IFAC Journal of Systems and Control 2 (2017) 1–11

Contents lists available at ScienceDirect

IFAC Journal of Systems and Control

journal homepage: www.elsevier.com/locate/ifacsc

System identification through online sparse Gaussian process regression with input noise

Hildo Bijl[a,*], Thomas B. Schön[b], Jan-Willem van Wingerden[a], Michel Verhaegen[a]

[a] Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands
[b] Department of Information Technology, Uppsala University, Uppsala, Sweden

ARTICLE INFO

Article history:
Received 15 May 2017
Revised 13 August 2017
Accepted 22 September 2017
Available online 29 September 2017

Keywords:
Nonlinear system identification
Gaussian processes
Regression
Machine learning
Sparse methods

ABSTRACT

There has been a growing interest in using non-parametric regression methods like Gaussian Process (GP) regression for system identification. GP regression does traditionally have three important downsides: (1) it is computationally intensive, (2) it cannot efficiently implement newly obtained measurements online, and (3) it cannot deal with stochastic (noisy) input points. In this paper we present an algorithm tackling all these three issues simultaneously. The resulting Sparse Online Noisy Input GP (SONIG) regression algorithm can incorporate new noisy measurements in constant runtime. A comparison has shown that it is more accurate than similar existing regression algorithms. When applied to nonlinear black-box system modeling, its performance is competitive with existing nonlinear ARX models.

1. Introduction

The Gaussian Process (GP) (Rasmussen & Williams, 2006) has established itself as a standard model for nonlinear functions. It offers a representation that is non-parametric and probabilistic. The *non-parametric* nature of the GP means that it does not rely on any particular parametric functional form to be postulated. The fact that the GP is a *probabilistic* model means that it is able to take uncertainty into account in every aspect of the model.

The nonlinear system identification problem amounts to learning a nonlinear mathematical model based on data that is observed from a dynamical phenomenon under study. Recently there has been a growing interest of using the GP to this end and it has in fact allowed researchers to successfully revisit the linear system identification problem and establish new and significantly improved results on the impulse estimation problem (Chen, Ohlsson, & Ljung, 2012; Pillonetto, Chiuso, & De Nicolao, 2011; Pillonetto & De Nicolao, 2010). There are also older results on nonlinear ARX type models (Kocijan, Girard, Banko, & Murray-Smith, 2005) and new results on the nonlinear state space models based on the GP (Frigola, Chen, & Rasmussen, 2014; Frigola, Lindsten, Schön, & Rasmussen, 2013; Svensson & Schön, 2017). We also mention the nice overview by Kocijan (2016).

When the basic GP model is used for regression, it results in a computational complexity that is too high to be practically useful, stochastic inputs cannot be used and it cannot be used in an online fashion. These three fundamental problems of basic GP regression have been addressed in many different ways, which we return to in Section 2. The nonlinear system identification problem typically requires us to solve these three problems simultaneously. This brings us to our two main contributions of this paper. (1) We derive an algorithm allowing us to—in an online fashion—include stochastic training points to one of the classic sparse GP models, the so-called FITC (Fully Independent Training Conditional) algorithm. (2) We adapt the new algorithm to the nonlinear system identification problem, resulting in an online algorithm for nonlinear system identification. The experimental results show that the our new algorithm is indeed competitive compared to existing solutions.

The system identification formulation takes inspiration from the nonlinear autoregressive model with exogenous (ARX) inputs of the following form

$$\boldsymbol{y_k} = \boldsymbol{\phi}(\boldsymbol{y_{k-1}}, \ldots, \boldsymbol{y_{k-n_y}}, \boldsymbol{u_{k-1}}, \ldots, \boldsymbol{u_{k-n_u}}), \tag{1}$$

where $\boldsymbol{\phi}(\cdot)$ denotes some nonlinear function of past inputs $\boldsymbol{u_{k-1}}, \ldots, \boldsymbol{u_{k-n_u}}$ and past outputs $\boldsymbol{y_{k-1}}, \ldots, \boldsymbol{y_{k-n_y}}$ to the system. To this end we develop a non-parametric and probabilistic GP model which takes the following vector

$$\boldsymbol{x_k} = (\boldsymbol{y_{k-1}}, \ldots, \boldsymbol{y_{k-n_y}}, \boldsymbol{u_{k-1}}, \ldots, \boldsymbol{u_{k-n_u}}), \tag{2}$$

as its input vector. The crucial part behind our solution from a system identification point of view is that we continuously

* Corresponding author.
E-mail addresses: h.j.bijl@tudelft.nl (H. Bijl), thomas.schon@it.uu.se (T.B. Schön), j.w.vanwingerden@tudelft.nl (J.-W. van Wingerden), m.verhaegen@tudelft.nl (M. Verhaegen).

https://doi.org/10.1016/j.ifacsc.2017.09.001
2468-6018/

keep track of the covariances between respective output estimates $\boldsymbol{y_k}, \ldots, \boldsymbol{y_{k-n_y}}$ and inputs $\boldsymbol{u_k}, \ldots, \boldsymbol{u_{k-n_u}}$. Every time we incorporate new training data, the respective means and covariances of these parameters are further refined.

The paper is organized as follows. Section 2 starts by examining three important existing problems within GP regression, giving a quick summary of the solutions discussed in literature. In Section 3 we expand on these methods, enabling GP regression to be applied in an online manner using noisy input points. This results in the basic version of the new algorithm. Section 4 subsequently outlines various ways of extending this algorithm, allowing it to be applied to system identification problems. Experimental results, first for the basic algorithm (Algorithm 1) and then for its system identification set-up (Algorithm 2) are shown in Section 5. Section 6 finally gives conclusions and recommendations for future work along this direction.

2. Backgrounds and limitations of Gaussian process regression

Gaussian process regression is a powerful regression method but, like any method, it has its limitations. In this section we look at its background, what exact limitations it has and what methods are available in literature to tackle these. Also the assumptions made and the notation used is introduced.

2.1. Regular Gaussian process regression

GP regression (Rasmussen & Williams, 2006) is about approximating a function $f(\boldsymbol{x})$ through a number of n training points (measurements) $(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), \ldots, (\boldsymbol{x_n}, y_n)$. Here, $\boldsymbol{x}$ denotes the training input point and y the measured function output. (For now we assume scalar outputs. Section 4.1 looks at the multi-output case.) We assume that the training outputs are corrupted by noise, such that $y_i = f(\boldsymbol{x_i}) + \varepsilon$, with $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$ being Gaussian white noise with zero mean and variance σ_n^2.

As shorthand notation, we merge all the training points $\boldsymbol{x_i}$ into a training set X and all corresponding output values y_i into an output vector $\boldsymbol{y}$. We now write the noiseless function output $f(X)$ as $\boldsymbol{f}$, such that $\boldsymbol{y} = \boldsymbol{f} + \boldsymbol{\varepsilon}$, with $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_n)$ and $\Sigma_n = \sigma_n^2 I$.

Once we have the training data, we want to predict the function value $f(\boldsymbol{x_*})$ at a specific test point $\boldsymbol{x_*}$. Equivalently, we can also predict the function values $\boldsymbol{f_*} = f(X_*)$ at a whole set of test points X_*. To accomplish this using GP regression, we assume that $\boldsymbol{f_*}$ and $\boldsymbol{f}$ have a prior joint Gaussian distribution given by

$$\begin{bmatrix} \boldsymbol{f^0} \\ \boldsymbol{f_*^0} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(X) \\ m(X_*) \end{bmatrix}, \begin{bmatrix} k(X,X) & k(X,X_*) \\ k(X_*,X) & k(X_*,X_*) \end{bmatrix} \right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{m} \\ \boldsymbol{m_*} \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix} \right), \tag{3}$$

where in the second part of the equation we have introduced another shorthand notation. Note here that $m(\boldsymbol{x})$ is the prior mean function for the Gaussian process and $k(\boldsymbol{x}, \boldsymbol{x'})$ is the prior covariance function. The superscript 0 in $\boldsymbol{f^0}$ and $\boldsymbol{f_*^0}$ also denotes that we are referring to the prior distribution: no training points have been taken into account yet. In this paper we make no assumptions on the prior mean/kernel functions, but our examples will apply a zero mean function $m(\boldsymbol{x}) = 0$ and a squared exponential covariance function

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \alpha^2 \exp\left(-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{x'})^T \Lambda^{-1} (\boldsymbol{x} - \boldsymbol{x'}) \right), \tag{4}$$

with α a characteristic output length scale and Λ a diagonal matrix of characteristic input squared length scales. For now we assume that these hyperparameters are known, but in Section 4.3 we look at ways to tune them.

From (3) we can find the posterior distribution of both $\boldsymbol{f}$ and $\boldsymbol{f_*}$ given $\boldsymbol{y}$ as

$$\begin{aligned} \begin{bmatrix} \boldsymbol{f^n} \\ \boldsymbol{f_*^n} \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu^n} \\ \boldsymbol{\mu_*^n} \end{bmatrix}, \begin{bmatrix} \Sigma^n & \Sigma_*^n \\ (\Sigma_*^n)^T & \Sigma_{**}^n \end{bmatrix} \right), \\ \begin{bmatrix} \boldsymbol{\mu^n} \\ \boldsymbol{\mu_*^n} \end{bmatrix} &= \begin{bmatrix} \left(K^{-1} + \Sigma_n^{-1}\right)^{-1} \left(K^{-1}\boldsymbol{m} + \Sigma_n^{-1}\boldsymbol{y}\right) \\ \boldsymbol{m_*} + K_*^T (K + \Sigma_n)^{-1} (\boldsymbol{y} - \boldsymbol{m}) \end{bmatrix}, \\ \begin{bmatrix} \Sigma^n & \Sigma_*^n \\ (\Sigma_*^n)^T & \Sigma_{**}^n \end{bmatrix} &= \begin{bmatrix} \left(K^{-1} + \Sigma_n^{-1}\right)^{-1} & \Sigma_n (K + \Sigma_n)^{-1} K_* \\ K_*^T (K + \Sigma_n)^{-1} \Sigma_n & K_{**} - K_*^T (K + \Sigma_n)^{-1} K_* \end{bmatrix}. \end{aligned} \tag{5}$$

Note here that, while we use $\boldsymbol{m}$ and K to denote properties of prior distributions, we use $\boldsymbol{\mu}$ and Σ for posterior distributions. The superscript n indicates these are posteriors taking n training points into account, and while a star $*$ subscript denotes a parameter of the test set, an omitted subscript denotes a training parameter.

2.2. Sparse Gaussian process regression

An important limitation of Gaussian process regression is its computational complexity of $\mathcal{O}(n^3)$, with n the number of training points. This can be tackled through parallel computing (Deisenroth & Ng, 2015; Gal, van der Wilk, & Rasmussen, 2014) but a more common solution is to use the so-called sparse methods. An overview of these is given by Candela and Rasmussen (2005), summarizing various contributions (Csató & Opper, 2002; Seeger, Williams, & Lawrence, 2003; Smola & Bartlett, 2001; Snelson & Ghahramani, 2006a) into a comprehensive framework. With these methods, and particularly with the FITC method that we will apply in this paper, the runtime can be reduced to being linear with respect to n, with only a limited reduction in how well the available data is being used.

All these sparse methods make use of so-called inducing input points X_u to reduce the computational complexity. Such inducing input points are also used in the more recent work on variational inference (Titsias, 2009; Titsias & Lawrence, 2010). However, as pointed out by McHutchon (2014), these points are now not used for the sake of computational speed but merely as 'information storage'. McHutchon (2014) also noted that the corresponding methods have a large number of parameters to optimize, making the computation of the derivatives rather slow. Furthermore, even though variations have been developed which do allow the application of variational inference to larger data sets (Damianou, Titsias, & Lawrence, 2016; Gal et al., 2014; Hensman, Nicoló, & Lawrence, 2013), online methods generally require simpler and faster methods, like the NIGP (Noisy Input GP) method from McHutchon and Rasmussen (2011), which is what we will focus on.

To apply sparse GP regression, we first find the posterior distribution of the inducing outputs $\boldsymbol{f_u}$ at the corresponding inducing input points X_u. This can be done in $\mathcal{O}(n^3)$ time through (5) (replacing $\boldsymbol{f_*}$ by $\boldsymbol{f_u}$) or in $\mathcal{O}(nn_u^2)$ time through the FITC regression equation

$$\begin{aligned} \begin{bmatrix} \boldsymbol{f^n} \\ \boldsymbol{f_u^n} \end{bmatrix} &\sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu^n} \\ \boldsymbol{\mu_u^n} \end{bmatrix}, \begin{bmatrix} \Sigma^n & \Sigma_u^n \\ (\Sigma_u^n)^T & \Sigma_{uu}^n \end{bmatrix} \right), \\ \begin{bmatrix} \boldsymbol{\mu^n} \\ \boldsymbol{\mu_u^n} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{m} + \Sigma^n \Sigma_n^{-1} (\boldsymbol{y} - \boldsymbol{m}) \\ \boldsymbol{m_u} + (\Sigma_u^n)^T \left(\Lambda_n^{-1} + \Sigma_n^{-1}\right) (\boldsymbol{y} - \boldsymbol{m}) \end{bmatrix}, \\ \Sigma^n &= \left(\Lambda_n^{-1} + \Sigma_n^{-1}\right)^{-1} + \Sigma_n (\Lambda_n + \Sigma_n)^{-1} K_u \Delta^{-1} K_u^T (\Lambda_n + \Sigma_n)^{-1} \Sigma_n, \\ \Sigma_u^n &= \Sigma_n (\Lambda_n + \Sigma_n)^{-1} K_u \Delta^{-1} K_{uu}, \\ \Sigma_{uu}^n &= K_{uu} \Delta^{-1} K_{uu}. \end{aligned} \tag{6}$$

Here we have used the shorthand notation $\Delta = K_{uu} + K_u^T (\Lambda_n + \Sigma_n)^{-1} K_u$ and $\Lambda_n = \mathrm{diag}\left(K - K_u^T K_{uu}^{-1} K_u\right)$, with diag be-

ing the function that sets all non-diagonal elements of the given matrix to zero. The FITC regression equation is an approximation, based on the assumption that all function values $f(\boldsymbol{x_1}), \ldots, f(\boldsymbol{x_n})$ are (a priori) independent given $\boldsymbol{f_u}$.

Once we know the distribution of $\boldsymbol{f_u}$, we calculate the posterior distribution of $\boldsymbol{f_*}$. Mathematically, this method is equivalent to assuming that $\boldsymbol{f}$ and $\boldsymbol{f_*}$ are conditionally independent, given $\boldsymbol{f_u}$. It follows that

$$\begin{bmatrix} \boldsymbol{f_u^n} \\ \boldsymbol{f_*^n} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu_u^n} \\ \boldsymbol{\mu_*^n} \end{bmatrix}, \begin{bmatrix} \Sigma_{uu}^n & \Sigma_{u*}^n \\ \Sigma_{*u}^n & \Sigma_{**}^n \end{bmatrix} \right),$$

$$\begin{bmatrix} \boldsymbol{\mu_u^n} \\ \boldsymbol{\mu_*^n} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu_u^n} \\ \boldsymbol{m_*} + K_{*u} K_{uu}^{-1} (\boldsymbol{\mu_u^n} - \boldsymbol{m_u}) \end{bmatrix},$$

$$\begin{bmatrix} \Sigma_{uu}^n & \Sigma_{u*}^n \\ \Sigma_{*u}^n & \Sigma_{**}^n \end{bmatrix} = \begin{bmatrix} \Sigma_{uu}^n & \Sigma_{uu}^n K_{uu}^{-1} K_{u*} \\ K_{*u} K_{uu}^{-1} \Sigma_{uu}^n & K_{**} - K_{*u} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}^n) K_{uu}^{-1} K_{u*} \end{bmatrix}. \tag{7}$$

2.3. Online Gaussian process regression

A second limitation of GP regression is the difficulty with which it can incorporate new training points. For regular GP regression, a new measurement $n+1$ can be added to the existing set of n training points through a matrix update, resulting in an $\mathcal{O}(n^2)$ runtime. For sparse methods using inducing input (basis) points this can generally be done more efficiently (Candela & Rasmussen, 2005; Csató & Opper, 2002; Hensman et al., 2013; Kou, Gao, & Guan, 2013; Ranganathan, Yang, & Ho, 2011). The main downside is that most methods set requirements on these inducing input points. However, the FITC and the PITC (Partially Independent Training Conditional) methods can be set up in an online way without such constraints (Bijl, van Wingerden, Schön, & Verhaegen, 2015; Huber, 2013; 2014). We briefly summarize the resulting algorithm.

Suppose that we know the distribution of $\boldsymbol{f_u}$, given the first n training points. This is written as $\boldsymbol{f_u^n}$. Next, consider a new measurement $(\boldsymbol{x_{n+1}}, y_{n+1})$, whose notation we will shorten to $(\boldsymbol{x_+}, y_+)$. To incorporate it, we first predict the posterior distribution of $f_+ = f(\boldsymbol{x_+})$ based on only the first n training points. Identically to (7), this results in

$$\begin{bmatrix} \boldsymbol{f_u^n} \\ f_+^n \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu_u^n} \\ \mu_+^n \end{bmatrix}, \begin{bmatrix} \Sigma_{uu}^n & \Sigma_{u+}^n \\ \Sigma_{+u}^n & \Sigma_{++}^n \end{bmatrix} \right),$$

$$\begin{bmatrix} \boldsymbol{\mu_u^n} \\ \mu_+^n \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu_u^n} \\ m_+ + K_{+u} K_{uu}^{-1} (\boldsymbol{\mu_u^n} - \boldsymbol{m_u}) \end{bmatrix},$$

$$\begin{bmatrix} \Sigma_{uu}^n & \Sigma_{u+}^n \\ \Sigma_{+u}^n & \Sigma_{++}^n \end{bmatrix} = \begin{bmatrix} \Sigma_{uu}^n & \Sigma_{uu}^n K_{uu}^{-1} K_{u+} \\ K_{+u} K_{uu}^{-1} \Sigma_{uu}^n & K_{++} - K_{+u} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}^n) K_{uu}^{-1} K_{u+} \end{bmatrix}. \tag{8}$$

If we subsequently incorporate the new measurement, identically to (5), then we get

$$\begin{bmatrix} \boldsymbol{f_u^{n+1}} \\ f_+^{n+1} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu_u^{n+1}} \\ \mu_+^{n+1} \end{bmatrix}, \begin{bmatrix} \Sigma_{uu}^{n+1} & \Sigma_{u+}^{n+1} \\ \Sigma_{+u}^{n+1} & \Sigma_{++}^{n+1} \end{bmatrix} \right),$$

$$\begin{bmatrix} \boldsymbol{\mu_u^{n+1}} \\ \mu_+^{n+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu_u^n} + \Sigma_{u+}^n \left(\Sigma_{++}^n + \sigma_n^2 \right)^{-1} \left(y_+ - \mu_+^n \right) \\ \sigma_n^2 \left(\Sigma_{++}^n + \sigma_n^2 \right)^{-1} \mu_+^n + \Sigma_{++}^n \left(\Sigma_{++}^n + \sigma_n^2 \right)^{-1} y_+ \end{bmatrix},$$

$$\begin{bmatrix} \Sigma_{uu}^{n+1} & \Sigma_{u+}^{n+1} \\ \Sigma_{+u}^{n+1} & \Sigma_{++}^{n+1} \end{bmatrix} = \begin{bmatrix} \Sigma_{uu}^n - \Sigma_{u+}^n (\Sigma_{++}^n + \sigma_n^2)^{-1} \Sigma_{+u}^n & \Sigma_{u+}^n (\Sigma_{++}^n + \sigma_n^2)^{-1} \sigma_n^2 \\ \sigma_n^2 (\Sigma_{++}^n + \sigma_n^2)^{-1} \Sigma_{+u}^n & \sigma_n^2 (\Sigma_{++}^n + \sigma_n^2)^{-1} \Sigma_{++}^n \end{bmatrix}. \tag{9}$$

This expression (or at least the part relating to $\boldsymbol{f_u}$) is the update law for the FITC algorithm. It tells us exactly how the distribution of $\boldsymbol{f_u^{n+1}}$ (both $\boldsymbol{\mu_u^{n+1}}$ and Σ_{uu}^{n+1}) depends on $\boldsymbol{x_+}$ and y_+. With this distribution, we can subsequently always find the distribution of new test point outputs $\boldsymbol{f_*}$ in an efficient way through (7).

2.4. Using stochastic input points

The third limitation is that the GP regression algorithm assumes that the input points are deterministic. This assumption concerns both the training (measurement) points $\boldsymbol{x}$ and the test points $\boldsymbol{x_*}$. For noisy (stochastic) test points $\boldsymbol{x_*}$, we can work around the problem by applying moment matching (Deisenroth, 2010). This technique can subsequently also be expanded for noisy training points (Dallaire, Besse, & Chaib-draa, 2009), but the effectiveness is limited because the method integrates over all possible *a priori* functions, and not over all possible *a posteriori* functions. There are methods that include posterior distributions (Girard & Murray-Smith, 2003) but these often only work for noisy test points and not for noisy training points. The NIGP algorithm (McHutchon & Rasmussen, 2011) is the only previously existing algorithm that we are aware of that is include posterior distributions at the same time as it can deal with noisy training points. This is the method we will be expanding upon in this paper.

It should be noted that the variational methods mentioned earlier can also deal with stochastic input points, up to a certain degree. However, as also mentioned before, they cannot do so as computationally efficient as the NIGP algorithm or the algorithm that we will develop, so their applicability to online system identification remains limited. Additionally, it is also possible to take into account the effects of noisy training points by assuming that the noise variance varies over the input space; a feature called heteroscedasticity. This has been investigated quite in-depth (Goldberg, Williams, & Bishop, 1997; Le & Smola, 2005; Snelson & Ghahramani, 2006b; Wang & Neal, 2012) but it would give more degrees of freedom to the learning algorithm than would be required, and as a result these methods have a reduced performance for the problems we consider. We will not consider these methods further in this paper.

3. Expanding the algorithm for stochastic training points

This section contains our main contribution: enabling the FITC algorithm to handle stochastic training points in an online way. From a computational point of view, the novel update laws given here are simple and efficient, relative to other methods.

3.1. The online stochastic measurement problem

Consider the case where we know the distribution $\boldsymbol{f_u^n} \sim \mathcal{N}(\boldsymbol{\mu_u^n}, \boldsymbol{\Sigma_{uu}^n})$ (initially we have $\boldsymbol{f_u^0} \sim \mathcal{N}\left(\boldsymbol{\mu_u^0}, \Sigma_{uu}^0\right) = \mathcal{N}(\boldsymbol{m_u}, K_{uu})$) and we obtain a new measurement at some unknown input point $\boldsymbol{x_+}$. As before, the true function output $f_+ = f(\boldsymbol{x_+})$ is also unknown. Our measurement only gives us values $\hat{\boldsymbol{x}}_+$ and $\hat{f}_+$ approximating these and hence tells us that $\boldsymbol{x_+} \sim \mathcal{N}(\hat{\boldsymbol{x}}_+, \Sigma_{+_x})$ and $f_+ \sim \mathcal{N}(\hat{f}_+, \Sigma_{+_f})$. (Note that $\hat{f}_+$ and y_+ are identical, and so are Σ_{+_f} and σ_n^2. For the sake of uniform notation, we have renamed them.) We assume that the noise on the input and the output is independent, and hence that $\boldsymbol{x_+}$ and f_+ are a priori not correlated.

Our main goal is to find the posterior distribution $\boldsymbol{f_u^{n+1}}$, given this stochastic training point. This can be done through

$$p(\boldsymbol{f_u^{n+1}} | \hat{\boldsymbol{x}}_+, \hat{f}_+, \boldsymbol{f_u^n}) = \int_X p(\boldsymbol{f_u^{n+1}} | \boldsymbol{x_+}, \hat{\boldsymbol{x}}_+, \hat{f}_+, \boldsymbol{f_u^n}) p(\boldsymbol{x_+} | \hat{\boldsymbol{x}}_+, \hat{f}_+, \boldsymbol{f_u^n}) \, d\boldsymbol{x_+}. \tag{10}$$

In the integral, the first probability $p(\boldsymbol{f_u^{n+1}} | \boldsymbol{x_+}, \hat{\boldsymbol{x}}_+, \hat{f}_+, \boldsymbol{f_u^n})$ is the update law for $\boldsymbol{f_u^{n+1}}$ if $\boldsymbol{x_+}$ was known exactly. It directly follows

from (9). The second probability $p(\boldsymbol{x}_+|\hat{\boldsymbol{x}}_+, \hat{f}_+, \boldsymbol{f_u^n})$ is the posterior distribution of $\boldsymbol{x}_+$, given both $\boldsymbol{f_u^n}$ and the new measurement. Since this latter term is more difficult to deal with, we examine it first.

3.2. The posterior distribution of the training point

The posterior distribution of $\boldsymbol{x}_+$ can be found through Bayes' theorem,

$$p(\boldsymbol{x}_+|\hat{f}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n}) = \frac{p(\hat{f}_+|\boldsymbol{x}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})\, p(\boldsymbol{x}_+|\hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})}{p(\hat{f}_+|\hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})}. \tag{11}$$

Here $p(\boldsymbol{x}_+|\hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n}) = p(\boldsymbol{x}_+|\hat{\boldsymbol{x}}_+) = \mathcal{N}(\hat{\boldsymbol{x}}_+, \Sigma_{+x})$ and $p(\hat{f}_+|\hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})$ equals an unknown constant (i.e., not depending on $\boldsymbol{x}_+$). Additionally,

$$p(\hat{f}_+|\boldsymbol{x}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n}) = p(\hat{f}_+|\boldsymbol{x}_+, \boldsymbol{f_u^n}) = \mathcal{N}\big(\mu_+^n, \Sigma_{++}^n + \Sigma_{+_f}\big), \tag{12}$$

where μ_+^n and Σ_{++}^n follow from (8). Note that both these quantities depend on $\boldsymbol{x}_+$ in a nonlinear way. Because of this, the resulting probability $p(\boldsymbol{x}_+|\hat{f}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})$ will be non-Gaussian. To work around this problem, we have to make some simplifying assumptions. Similarly to Girard and Murray-Smith (2003), we linearize[1] the Gaussian process $\hat{f}_+$ (which depends on $\boldsymbol{x}_+$) around a point $\bar{\boldsymbol{x}}_+$. That is, we assume that

$$p(\hat{f}_+|\boldsymbol{x}_+, \boldsymbol{f_u^n}) = \mathcal{N}\left(\mu_+^n(\bar{\boldsymbol{x}}_+) + \frac{d\mu_+^n(\bar{\boldsymbol{x}}_+)}{d\boldsymbol{x}_+}(\boldsymbol{x}_+ - \bar{\boldsymbol{x}}_+),\, \Sigma_{++}^n(\bar{\boldsymbol{x}}_+) + \Sigma_{+_f}\right). \tag{13}$$

In other words, we assume that the mean varies linearly with $\boldsymbol{x}_+$, while the covariance is constant everywhere. This is a necessary assumption for $p(\boldsymbol{x}_+|\hat{f}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n})$ to have a Gaussian solution. It follows as

$$\begin{aligned}
p(\boldsymbol{x}_+|\hat{f}_+, \hat{\boldsymbol{x}}_+, \boldsymbol{f_u^n}) &= \mathcal{N}\big(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}, \Sigma_{+_x}^{n+1}\big),\\
\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}} &= \hat{\boldsymbol{x}}_+ + \Sigma_{+_x}^{n+1}\left(\left(\frac{d\mu_+^n(\bar{\boldsymbol{x}}_+)}{d\boldsymbol{x}_+}\right)^T\big(\Sigma_{++}^n(\bar{\boldsymbol{x}}_+) + \Sigma_{+_f}\big)^{-1}\right.\\
&\qquad \left.\left(\frac{d\mu_+^n(\bar{\boldsymbol{x}}_+)}{d\boldsymbol{x}_+}(\bar{\boldsymbol{x}}_+ - \hat{\boldsymbol{x}}_+) + \big(\hat{f}_+ - \mu_+^n(\bar{\boldsymbol{x}}_+)\big)\right)\right),\\
\Sigma_+^{n+1} &= \left(\left(\frac{d\mu_+^n(\bar{\boldsymbol{x}}_+)}{d\boldsymbol{x}_+}\right)^T\big(\Sigma_{++}^n(\bar{\boldsymbol{x}}_+) + \Sigma_{+_f}\big)^{-1}\right.\\
&\qquad \left.\left(\frac{d\mu_+^n(\bar{\boldsymbol{x}}_+)}{d\boldsymbol{x}_+}\right) + \Sigma_{+_x}^{-1}\right)^{-1}.
\end{aligned} \tag{14}$$

We are left to choose a linearization point $\bar{\boldsymbol{x}}_+$. The above equation is easiest to apply when we choose $\bar{\boldsymbol{x}}_+ = \hat{\boldsymbol{x}}_+$, but when $\big(\hat{f}_+ - \mu_+^n(\bar{\boldsymbol{x}}_+)\big)$ is large, this may result in inaccuracies due to the linearization. It is generally more accurate to choose $\bar{\boldsymbol{x}}_+$ as $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$. However, $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$ is initially unknown, so it may be necessary to apply the above equation multiple times, each time resetting $\bar{\boldsymbol{x}}_+$ to the latest value of $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$ that was found, to find the most accurate posterior distribution of $\boldsymbol{x}_+$.

3.3. Updating the inducing input point function values

Using the above, we can solve (10). This is done by approximating the GP $\boldsymbol{f_u}(\boldsymbol{x}_+)$ by its Taylor expansion around $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$. Elementwise we write this as

$$\begin{aligned}
f_{u_i}^{n+1}(\boldsymbol{x}_+) &= f_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}) + \frac{df_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\big(\boldsymbol{x}_+ - \hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}\big)\\
&\quad + \frac{1}{2}\big(\boldsymbol{x}_+ - \hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}\big)^T\frac{d^2 f_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+^2}\big(\boldsymbol{x}_+ - \hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}\big) + \ldots.
\end{aligned} \tag{15}$$

Within this Taylor expansion, Girard and Murray-Smith (2003) made the assumption that higher order derivatives like $\frac{d^2\boldsymbol{f_u^n}}{d\boldsymbol{x}_+^2}$ are negligible, remaining with just a linearization of the GP. We do not make this assumption, but instead assume that Σ_{+x}^2 and higher powers of Σ_{+x} are negligible. (If the uncertainties in $\boldsymbol{x}_+$ are so large that this assumption does not hold, then any form of Gaussian process regression is likely to fail.) This assumption is not only more loose—resulting in an extra term in (15)—but it is also easier to verify.

An additional assumption we need to make is that $\boldsymbol{x}_+$ is independent of $\boldsymbol{f_u}$. This is reasonable, as $\boldsymbol{x}_+$ is only contaminated by Gaussian white noise. Applying this, we can solve (10) through both (14) and (15). The result equals

$$\begin{aligned}
\boldsymbol{f_u^{n+1}} &\sim \mathcal{N}\big(\boldsymbol{\mu_u^{n+1}}, \Sigma_{uu}^{n+1}\big),\\
\mu_{u_i}^{n+1} &= \mu_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}) + \frac{1}{2}\mathrm{tr}\left(\frac{d^2\mu_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+^2}\Sigma_{+_x}^{n+1}\right),\\
\Sigma_{u_iu_j}^{n+1} &= \Sigma_{u_iu_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}) + \left(\frac{d\mu_{u_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\right)\Sigma_{+_x}^{n+1}\left(\frac{d\mu_{u_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\right)^T\\
&\quad + \frac{1}{2}\mathrm{tr}\left(\left(\frac{d^2\Sigma_{u_iu_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+^2}\right)\Sigma_{+_x}^{n+1}\right).
\end{aligned} \tag{16}$$

Here, the functions $\mu_{u_i}^{n+1}(\boldsymbol{x}_+)$ and $\Sigma_{u_iu_j}^{n+1}(\boldsymbol{x}_+)$ (for a given point $\boldsymbol{x}_+$) are given by (9), combined with (8). Finding all the derivatives of these parameters can be a daunting task, especially for non-scalar inputs $\boldsymbol{x}_+$, but the mathematics are relatively straightforward, so for this we refer to the Appendix.

It is interesting to compare expression (16) with what was used by McHutchon and Rasmussen (2011) in their NIGP algorithm. They did not include the term involving $d^2\mu_{u_i}^{n+1}/d\boldsymbol{x}_+^2$. Later on, in Section 5.1, we will find that exactly this term causes the new algorithm to perform better than the NIGP algorithm. As such, the above update law (16) also serves as an improvement with respect to the NIGP algorithm.

3.4. The SONIG algorithm

Applying the equations developed so far is done through the Sparse Online Noisy Input GP (SONIG) algorithm, outlined in Algorithm 1. This algorithm is computationally efficient, in the sense that a single updating step (incorporating one training point) can be done in constant runtime with respect to the number of training points already processed. The runtime does depend on the number of inducing input points through $\mathcal{O}(n_u^3)$, just like it does for all sparse GP regression algorithms.

4. Extensions of the SONIG algorithm

In the previous section we have presented the basic idea behind the SONIG algorithm. There are various further extensions that can be derived and implemented in the algorithm. For instance, the algorithm can deal with multi-output functions $\boldsymbol{f}(\boldsymbol{x})$ (Section 4.1), it can give us the posterior distribution of the output $\boldsymbol{f}_+$ as well as its correlation with the input $\boldsymbol{x}_+$ (Section 4.2), we can implement hyperparameter tuning (Section 4.3), we can add inducing input

[1] Applying moment-matching like in Section 4.5 does not work here, since we would have to integrate over the inverse of a matrix $k(\boldsymbol{x}, \boldsymbol{x})$ that depends on the integration parameter $\boldsymbol{x}$.

Algorithm 1: The Sparse Online Noisy Input GP (SONIG) algorithm: an online version of the FITC algorithm capable of dealing with stochastic (noisy) training points.

Input:
A possibly expanding set of training points $(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_n}, y_n)$ in which both $\boldsymbol{x}$ and y are distorted by Gaussian white noise.

Preparation:
Either choose the hyperparameters based on expert knowledge, or apply the NIGP hypertuning methods of McHutchon and Rasmussen (2011) on a subset of the data (a few hundred points) to find the hyperparameters.
Optionally, apply the NIGP regression methods on this subset of data to obtain an initial distribution of $\boldsymbol{f_u}$.
Otherwise, initialize $\boldsymbol{f_u}$ as $\mathcal{N}(\boldsymbol{m_u}, K_{uu})$.

Updating:
while *there are unprocessed training points* $(\boldsymbol{x_{n+1}}, y_{n+1})$ **do**
1. Apply (14) to find the posterior distribution of the training point $\boldsymbol{x_{n+1}}$ (written as $\boldsymbol{x_+}$).
2. Use (16) to update the distribution of $\boldsymbol{f_u}$.
3. Optionally, use (17) and (18) to calculate the posterior distribution of the function value $\boldsymbol{f}(\boldsymbol{x_+})$ (written as $\boldsymbol{f_+}$).

end

Prediction:
Apply (7) to find the distribution $\boldsymbol{f_*}$ for any set of deterministic test points. For stochastic test points, use the expansion from Section 4.5.

points online (Section 4.4) and we can make predictions $\boldsymbol{f_*}$ using stochastic test points $\boldsymbol{x_*}$ (Section 4.5). Many of these extensions are necessary to apply the SONIG algorithm for system identification. The resulting system identification algorithm is summarized in Algorithm 2.

4.1. Multiple outputs

So far we have approximated functions $f(\boldsymbol{x})$ with only one output. It is also possible to approximate functions $\boldsymbol{f}(\boldsymbol{x})$ with $d_y > 1$ outputs. A common way in which this is done in GP regression algorithms (Álvarez, Rosasco, & Lawrence, 2012; Deisenroth & Rasmussen, 2011) is by assuming that, given a deterministic input $\boldsymbol{x}$, all outputs $f_1(\boldsymbol{x}), \ldots, f_{d_y}(\boldsymbol{x})$ are independent. With this assumption, it is possible to keep a separate inducing input point distribution $\boldsymbol{f_u^i} \sim \mathcal{N}\left(\boldsymbol{\mu_u^i}, \Sigma_u^i\right)$ for each output $f_i(\boldsymbol{x})$. Hence, each output is basically treated separately.

When using stochastic input points (again, see Deisenroth & Rasmussen, 2011) the outputs do become correlated. We now have two options. If we take this correlation into account, we have to keep track of the joint distribution of the vectors $\boldsymbol{f_u^1}, \ldots, \boldsymbol{f_u^{d_y}}$, effectively merging them into one big vector. This results in a vector of size $n_u d_y$, giving our algorithm a computational complexity of $\mathcal{O}(n_u^3 d_y^3)$. Alternatively, we could also neglect the correlation between the inducing input point distributions $\boldsymbol{f_u^i}$ caused by stochastic training points $\boldsymbol{x_+} \sim \mathcal{N}(\hat{\boldsymbol{x}}_+, \Sigma_{+_x})$. If we do, we can continue to treat each function output separately, giving our algorithm a runtime of $\mathcal{O}(n_u^3 d_y)$. Because one of our aims in this paper is to reduce the runtime of GP regression algorithms, we will apply the second option.

When each output is treated separately, each output also has its own hyperparameters. Naturally, the prior output covariance α_i^2 and the output noise $\sigma_{n_i}^2$ can differ per output f^i, but also the input length scales Λ_i may be chosen differently for each output. In fact, it is even possible to specify a fully separate covariance function $k_i(\boldsymbol{x}, \boldsymbol{x}')$ per output f^i, though in this paper we stick with the squared exponential covariance function.

Naturally, there are a few equations which we should adjust slightly in the case of multivariate outputs. In particular, in equation (14), the parameter $\Sigma_{++}^n(\bar{\boldsymbol{x}}_+)$ would not be a scalar anymore, but become a matrix. Due to our assumption that the outputs are independent, it would be a diagonal matrix. Similarly, the derivative $d\mu_+^n/d\boldsymbol{x}_+$ would not be a row vector anymore. Instead, it would turn into the matrix $d\boldsymbol{\mu_+^n}/d\boldsymbol{x}_+$. With these adjustments, (14) still holds and all other equations can be applied as usual.

4.2. The posterior distribution of the measured output

In Section 3.3 we found the posterior distribution for $\boldsymbol{f_u}$. For some applications (like the system identification set-up presented in Section 5.2) we also need the posterior distribution of the measured function value $\boldsymbol{f_+}$, even though we do not exactly know to which input it corresponds. We can find this element-wise, using the same methods, through

$$\mathbb{E}[\boldsymbol{f_+}]_i = \mu_{+_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}) + \frac{1}{2}\mathrm{tr}\left(\frac{d^2\mu_{+_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+^2}\Sigma_{+_x}^{n+1}\right),$$

$$\begin{aligned}\mathbb{V}[\boldsymbol{f_+}, \boldsymbol{f_+}]_{i,j} = {} & \Sigma_{+_i+_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}) \\ & + \left(\frac{d\mu_{+_i}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\right)\Sigma_{+_x}^{n+1}\left(\frac{d\mu_{+_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\right)^T \\ & + \frac{1}{2}\mathrm{tr}\left(\left(\frac{d^2\Sigma_{+_i+_j}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+^2}\right)\Sigma_{+_x}^{n+1}\right). \end{aligned} \tag{17}$$

Note here that $\Sigma_{++}^{n+1}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})$ is (by assumption) a diagonal matrix, simplifying the above equation for non-diagonal terms. As such, the covariance between two different function outputs f_{+_i} and f_{+_j} only depends on the second term in the above expression.

It may occur that we also need to know the posterior covariance between the function value $\boldsymbol{f_+}$ and the function input $\boldsymbol{x_+}$. Using the same method, we can find that

$$\mathbb{V}[\boldsymbol{f_+}, \boldsymbol{x_+}] = \left(\frac{d\boldsymbol{\mu_+^{n+1}}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}})}{d\boldsymbol{x}_+}\right)\Sigma_{+_x}^{n+1}. \tag{18}$$

This allows us to find the joint posterior distribution of $\boldsymbol{x_+}$ and $\boldsymbol{f_+}$.

4.3. Applying hyperparameter tuning to the algorithm

So far we have assumed that the hyperparameters of the Gaussian process are known a priori. When this is not the case, they need to be tuned first. While this could be done using expert knowledge of the system, it can also be done automatically.

McHutchon and Rasmussen (2011), with their NIGP method, offer an effective method of tuning the hyperparameters, which also tells us the input noise variance Σ_{+_x}. However, this method has a computational complexity of $\mathcal{O}(n^3)$, with n still the number of training points. As such, it can only be used for a small number of measurements and it cannot be used online. Hence NIGP does not seem to be applicable to our problem.

However, Chalupka, Williams, and Murray (2013) compare various GP regression algorithms, including methods to tune hyperparameters. One of their main conclusions is that the subset-of-data (SoD) method provides a good trade-off between computational complexity and prediction accuracy. When using the SoD method, we do not apply hyperparameter tuning to our full data set, of

possibly tens of thousands of input–output pairs. Instead, we randomly take a subset (say, a few hundred) of these data points and tune the hyperparameters only based on this selection. For this latter step any suitable hyperparameter tuning algorithm can be used, although we use the NIGP method described by McHutchon and Rasmussen (2011).

Chalupka et al. (2013) also conclude that, for the regression problem with known hyperparameters, the FITC algorithm provides a very good trade-off between complexity and accuracy. So after having tuned the hyperparameters, it will be an effective choice to use the online FITC algorithm with stochastic training points (that is, the SONIG algorithm) as our regression method.

4.4. Adjusting the set of inducing input points online

When using an online GP regression algorithm, it is often not known in advance what kind of measured input points $\boldsymbol{x}$ the system will get. As such, choosing the inducing input points X_u in advance is not always possible, nor wise. Instead, we can adjust the inducing input points while the algorithm is running. There are ways to fully tune the set of inducing input points, like using the latent variable methods by Titsias (2009) and Titsias and Lawrence (2010), but those methods require the optimization of many parameters, resulting in a computationally complex procedure. To keep the required computations limited, we have to opt for a simpler method and add/remove inducing input points based on areas of the input space we are interested in or have data at.

Suppose that we denote the current set of inducing input points by X_u, and that we want to add an extra set of inducing input points X_{u_+}. In this case, given all the data that we have, the distributions of $\boldsymbol{f_u}$ and $\boldsymbol{f_{u_+}}$ satisfy, identically to (7),

$$\begin{bmatrix} \boldsymbol{f_u} \\ \boldsymbol{f_{u_+}} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu_u} \\ \boldsymbol{\mu_{u_+}} \end{bmatrix}, \begin{bmatrix} \Sigma_{uu} & \Sigma_{uu_+} \\ \Sigma_{u_+u} & \Sigma_{u_+u_+} \end{bmatrix} \right),$$

$$\begin{bmatrix} \boldsymbol{\mu_u} \\ \boldsymbol{\mu_{u_+}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu_u} \\ \boldsymbol{m_{u_+}} + K_{u_+u}K_{uu}^{-1}(\boldsymbol{\mu_u} - \boldsymbol{m_u}) \end{bmatrix},$$

$$\begin{bmatrix} \Sigma_{uu}^n & \Sigma_{uu_+}^n \\ \Sigma_{u_+u}^n & \Sigma_{u_+u_+}^n \end{bmatrix} = \begin{bmatrix} \Sigma_{uu}^n & \Sigma_{uu}^n K_{uu}^{-1} K_{uu_+} \\ K_{u_+u}K_{uu}^{-1}\Sigma_{uu}^n & K_{u_+u_+} - K_{u_+u}K_{uu}^{-1}(K_{uu} - \Sigma_{uu}^n)K_{uu}^{-1}K_{uu_+} \end{bmatrix}. \tag{19}$$

With this combined set of old and new inducing input points, we can then continue incorporating new training points without losing any data.

Additionally, it is possible to remove unimportant inducing input points when desired. An inducing input point can be 'unimportant' when it does not provide much information (it contributes little to the log-likelihood) or when it provides information we are not interested in, for instance when it lies in a part of the input space we do not care about. In this case, its entry can simply be removed from $\boldsymbol{f_u}$. Since it is possible to both add and remove inducing input points, it is naturally also possible to shift them around (first add new points, then remove old points) whenever deemed necessary.

The way in which we add inducing input points in the SONIG algorithm is as follows. Whenever we incorporate a new training point with posterior input distribution $\boldsymbol{x}_+ \sim \mathcal{N}(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}, \Sigma_{+x}^{n+1})$, we check if $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$ is already close to any existing inducing input point. To be more precise, we examine the normalized squared distance

$$\left(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}} - \boldsymbol{x_{u_i}}\right)^T \Lambda^{-1} \left(\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}} - \boldsymbol{x_{u_i}}\right) \tag{20}$$

for each inducing input point $\boldsymbol{x_{u_i}}$. If there is no inducing input point whose normalized squared distance is below a given threshold (often chosen to be roughly 1, but tuned to get a satisfactory number of points), then it means that there is no inducing input point $\boldsymbol{x_{u_i}}$ close to our new training point $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$. As a result, we add $\hat{\boldsymbol{x}}_+^{\boldsymbol{n+1}}$ to our set of inducing input points. This guarantees that each training point is close to at least one inducing input point, which always allows the data from the measurement to be taken into account.

Of course there is also a variety of other methods to choose the inducing input points, like distributing them in advance or tuning them along with the hyperparameters. The above method has shown to result in a good trade-off between accuracy and computational complexity for many problems, but it always depends on the problem at hand which inducing input point selection method works best.

4.5. Predictions for stochastic test points

For deterministic test points $\boldsymbol{x_*}$ we can simply make use of (7) to compute predictions. However, for a stochastic test point $\boldsymbol{x_*} \sim \mathcal{N}(\hat{\boldsymbol{x}}_*, \Sigma_{*_x})$ it is more challenging, since we have to calculate the distribution of $\boldsymbol{f_*} = \boldsymbol{f}(\boldsymbol{x_*})$, requiring us to solve an integration problem. This will not result in a Gaussian distribution, so once more we will apply moment matching. Previously, we had to make additional assumptions, to make sure that the mean vector and the covariance matrix could be solved for analytically. This time we do not have to. Deisenroth (2010) showed, based on work by Candela, Girard, Larsen, and Rasmussen (2003) and Girard, Rasmussen, Candela, and Murray-Smith (2003), that for the squared exponential covariance function (and also various other functions) the mean vector and the covariance matrix can be calculated analytically. We can apply the same ideas in our present setting.

For our results, we will first define some helpful quantities. When doing so, we should note that in theory every output $f_k(\boldsymbol{x})$ can have its own covariance function $k_k(\ldots)$, and as such its own hyperparameters α_k and Λ_k. (See Section 4.1.) Keeping this in mind, we now define the vectors $\boldsymbol{q^k}$ and matrices Q^{kl} element-wise as

$$\begin{aligned} q_i^k &= \int_X k_k(\boldsymbol{x_{u_i}}, \boldsymbol{x_*})\, p(\boldsymbol{x_*})\, d\boldsymbol{x_*} \\ &= \frac{\alpha_k^2}{\sqrt{|\Sigma_{*_x}||\Sigma_{*_x}^{-1} + \Lambda_k^{-1}|}} \\ &\quad \exp\left(-\frac{1}{2}\left(\boldsymbol{x_{u_i}} - \hat{\boldsymbol{x}}_*\right)^T (\Lambda_k + \Sigma_{*_x})^{-1} \left(\boldsymbol{x_{u_i}} - \hat{\boldsymbol{x}}_*\right)\right), \end{aligned} \tag{21}$$

$$\begin{aligned} Q_{ij}^{kl} &= \int_X k_k(\boldsymbol{x_{u_i}}, \boldsymbol{x_*}) k_l(\boldsymbol{x_*}, \boldsymbol{x_{u_j}})\, p(\boldsymbol{x_*})\, d\boldsymbol{x_*} \\ &= \frac{\alpha_k^2 \alpha_l^2}{\sqrt{|\Sigma_{*_x}||\Sigma_{*_x}^{-1} + \Lambda_k^{-1} + \Lambda_l^{-1}|}} \\ &\quad \exp\left(-\frac{1}{2}\left(\boldsymbol{x_{u_i}} - \boldsymbol{x_{u_j}}\right)^T (\Lambda_k + \Lambda_l)^{-1} \left(\boldsymbol{x_{u_i}} - \boldsymbol{x_{u_j}}\right)\right) \\ &\quad \exp\left(-\frac{1}{2}\left(\bar{\boldsymbol{x}}_{\boldsymbol{u_{ij}}}^{\boldsymbol{kl}} - \hat{\boldsymbol{x}}_*\right)^T \left(\left(\Lambda_k^{-1} + \Lambda_l^{-1}\right)^{-1} + \Sigma_{*_x}\right)^{-1} \left(\bar{\boldsymbol{x}}_{\boldsymbol{u_{ij}}}^{\boldsymbol{kl}} - \hat{\boldsymbol{x}}_*\right)\right), \end{aligned} \tag{22}$$

where we have defined

$$\bar{\boldsymbol{x}}_{\boldsymbol{u_{ij}}}^{\boldsymbol{kl}} = \left(\Lambda_k^{-1} + \Lambda_l^{-1}\right)^{-1} \left(\Lambda_k^{-1} \boldsymbol{x_{u_i}} + \Lambda_l^{-1} \boldsymbol{x_{u_j}}\right). \tag{23}$$

With these quantities, we can find that

$$\begin{aligned} \boldsymbol{f_*} &\sim \mathcal{N}(\boldsymbol{\mu_*}, \Sigma_*), \\ [\boldsymbol{\mu_*}]_k &= \left(\boldsymbol{q^k}\right)^T \left(K_{uu}^k\right)^{-1} \boldsymbol{\mu_u^k}, \\ [\Sigma_*]_{k,k} &= \alpha_k^2 - \mathrm{tr}\left(\left(K_{uu}^k\right)^{-1} \left(K_{uu}^k - \Sigma_u^k\right) \left(K_{uu}^k\right)^{-1} Q^{kk}\right) \end{aligned}$$

$$+ \left(\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{k}}\right)^T \left(K_{uu}^k\right)^{-1} Q^{kk} \left(K_{uu}^k\right)^{-1} \boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{k}} - [\boldsymbol{\mu}_*]_k^2,$$

$$[\Sigma_*]_{k,l} = \left(\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{k}}\right)^T \left(K_{uu}^k\right)^{-1} Q^{kl} \left(K_{uu}^l\right)^{-1} \boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{l}} - [\boldsymbol{\mu}_*]_k [\boldsymbol{\mu}_*]_l, \tag{24}$$

where the latter expression is for the non-diagonal terms of Σ_* (with $k \neq l$). Note that the first line of the above expression is in fact an approximation. In reality the distribution $\boldsymbol{f}_*$ is not Gaussian. The other two lines, however, are the analytic mean vector and covariance matrix. With these quantities, we can accurately predict the distribution of the output $\boldsymbol{f}_*$ for stochastic test points $\boldsymbol{x}_*$.

5. Experimental results

In this section we apply the developed algorithm to test problems and compare its performance to existing state of the art solutions. First we apply the basic SONIG algorithm (Algorithm 1) to approximate a sample function, allowing us to compare its performance to other regression algorithms. The results of this are discussed in Section 5.1. Then we apply the SONIG algorithm with all the extensions from Section 4 (Algorithm 2) to identify a time-invariant[2] magneto-rheological fluid damper, the outcome of which is reported in Section 5.2. All code for these examples, as well as for using the SONIG algorithm in general, is available on GitHub, see Bijl (2017).

5.1. Evaluating the SONIG algorithm through a sample function

To compare the SONIG algorithm with other algorithms, we have set up a basic single-input single-output GP experiment. First, we randomly generate a sample function from a Gaussian process with a squared exponential covariance function (see (4)). This is done on the range $x \in [-5, 5]$, subject to the hyperparameters $\alpha = 1$ and $\Lambda = 1$. Subsequently, we take n training points at random places in the input range and distort both the input x and the output y with zero-mean Gaussian white noise with standard deviation $\sigma_x = 0.4$ and $\sigma_n = 0.1$, respectively. We use $n = 200$ unless mentioned otherwise. To this data set, we then apply the following algorithms.

(1) GP regression without any input noise and with the exact hyperparameters, given above. This serves as a reference case: all other algorithms get noisy input points and tuned hyperparameters.
(2) GP regression with input noise and with hyperparameters tuned through the maximum-likelihood method.
(3) The NIGP algorithm of McHutchon and Rasmussen (2011). This algorithm has its own method of tuning hyperparameters, including σ_x.
(4) The SONIG algorithm, starting with $\boldsymbol{\mu}_{\boldsymbol{u}}^{\boldsymbol{0}} = \boldsymbol{m}_{\boldsymbol{u}}$ and $\Sigma_{uu}^0 = K_{uu}$, using the hyperparameters given by (3). We use $X_u = \{-5, -4.5, -4, \ldots, 5\}$, resulting in $n_u = 21$ evenly distributed inducing input points.
(5) The same as (4), but now with more training points (800 instead of 200). Because the SONIG algorithm is computationally more efficient than the NIGP algorithm, the runtime of this is similar to that of (3), being roughly 2–3 s when using Matlab, although this of course does depend on the exact implementation of the algorithms.
(6) NIGP applied on a subset of data (100 training points) to predict the distribution of the inducing input points, followed by the SONIG algorithm applied to the remainder (700) of the training set, further updating the inducing input points. The runtime of this approach is again similar to that of (3), being 2–3 s.
(7) The FITC algorithm, using the hyperparameters of (2). This serves as a reference case.

[2] The SONIG algorithm can also be applied to systems that slowly vary in time. In this case old data needs to be 'forgotten', for instance by slowly increasing Σ_{uu} as time passes.

Algorithm 2: The steps required to identify nonlinear systems with measurement noise in an online way using the SONIG method.

Input:
A set of inputs $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots$ and outputs $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots$ of a time-invariant system that is to be identified. Both the input and the output can be disturbed by noise.

Preparation:
Define hyperparameters, either through the NIGP algorithm or by using expert knowledge about the system. Optionally, also define an initial set of inducing input points X_u.

Updating:
while *there are unprocessed measurements* $\boldsymbol{y}_{\boldsymbol{k+1}}$ **do**
1. Set up $\boldsymbol{x}_{\boldsymbol{k+1}}$ (shortened to $\boldsymbol{x}_+$) using its definition in (1). Find its prior distribution using known covariances between system outputs $\boldsymbol{y}_{\boldsymbol{k}}, \ldots, \boldsymbol{y}_{\boldsymbol{k-(n_y-1)}}$ and (if necessary) system inputs $\boldsymbol{u}_{\boldsymbol{k}}, \ldots, \boldsymbol{u}_{\boldsymbol{k-(n_u-1)}}$. Also find the prior distribution of the function output $\boldsymbol{y}_{\boldsymbol{k+1}}$ (denoted as $\boldsymbol{f}_{\boldsymbol{k+1}}$ or shortened as $\boldsymbol{f}_+$).
2. Apply (14) to find the posterior distribution $\mathcal{N}\left(\hat{\boldsymbol{x}}_+^{\boldsymbol{k+1}}, \Sigma_{+x}^{k+1}\right)$ of $\boldsymbol{x}_+$. Optionally, use this to update the posterior distribution of the system outputs $\boldsymbol{y}_{\boldsymbol{k}}, \ldots, \boldsymbol{y}_{\boldsymbol{k-(n_y-1)}}$ and system inputs $\boldsymbol{u}_{\boldsymbol{k}}, \ldots, \boldsymbol{u}_{\boldsymbol{k-(n_u-1)}}$.
3. Optionally, if $\hat{\boldsymbol{x}}_+^{\boldsymbol{k+1}}$ is far removed from any inducing input point, add it to the set of inducing inputs X_u using (19). (Or rearrange/tune the inducing input points in any desired way.)
4. Calculate the posterior distribution of the inducing input vector $\boldsymbol{f}_{\boldsymbol{u}}$ for each of the outputs of $\boldsymbol{\phi}$ using (16).
5. Calculate the posterior distribution of $\boldsymbol{y}_{\boldsymbol{k+1}}$ using (17). Additionally, calculate the covariances between $\boldsymbol{y}_{\boldsymbol{k+1}}$ and each of the previous system outputs $\boldsymbol{y}_{\boldsymbol{k}}, \ldots, \boldsymbol{y}_{\boldsymbol{k-(n_y-1)}}$ and inputs $\boldsymbol{u}_{\boldsymbol{k}}, \ldots, \boldsymbol{u}_{\boldsymbol{k-(n_u-1)}}$ through (18).
end

Prediction:
For any deterministic set of previous outputs $\boldsymbol{y}_{\boldsymbol{k}}, \ldots, \boldsymbol{y}_{\boldsymbol{k-(n_y-1)}}$ and inputs $\boldsymbol{u}_{\boldsymbol{k}}, \ldots, \boldsymbol{u}_{\boldsymbol{k-(n_u-1)}}$, apply (7) to predict the next output $\boldsymbol{y}_{\boldsymbol{k+1}}$. For stochastic outputs and inputs, use the expansion from Section 4.5.

For all these algorithms, we examine both the Mean Squared Error (MSE) of the resulting prediction and the mean variance given by the regression algorithm. The latter is basically the estimate by the regression algorithm of the MSE. By comparing it with the real MSE, we learn about the integrity of the algorithm. As such, the ratio between these two is an indication of the algorithm integrity. We do this whole process 400 times, each time for a different randomly generated sample function from the Gaussian process. The average of the results is subsequently shown in Table 1.

There are many things that can be noticed from Table 1. First of all, it is that for the given type of functions, and for an equal number of training points, the SONIG algorithm performs better

Table 1
Comparison of various GP regression algorithms, applied to noisy measurements of 400 randomly generated sample functions. For details, see the main text.

	n	MSE	Mean variance	Ratio
(1) GPR with exact hyperparameters and no input noise	200	$0.87 \cdot 10^{-3}$	$0.85 \cdot 10^{-3}$	1.02
(2) GPR with tuned hyperparameters	200	$28.0 \cdot 10^{-3}$	$8.3 \cdot 10^{-3}$	3.4
(3) NIGP with its own hyperparameter tuning	200	$26.2 \cdot 10^{-3}$	$5.6 \cdot 10^{-3}$	4.7
(4) SONIG using the hyperparameters of (3)	200	$21.5 \cdot 10^{-3}$	$8.1 \cdot 10^{-3}$	2.7
(5) SONIG using the hyperparameters of (3)	800	$12.5 \cdot 10^{-3}$	$2.2 \cdot 10^{-3}$	5.6
(6) NIGP on a subset, followed by SONIG on the rest	100/700	$16.5 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	7.1
(7) FITC, using the hyperparameters of (2)	800	$19.5 \cdot 10^{-3}$	$2.7 \cdot 10^{-3}$	7.1

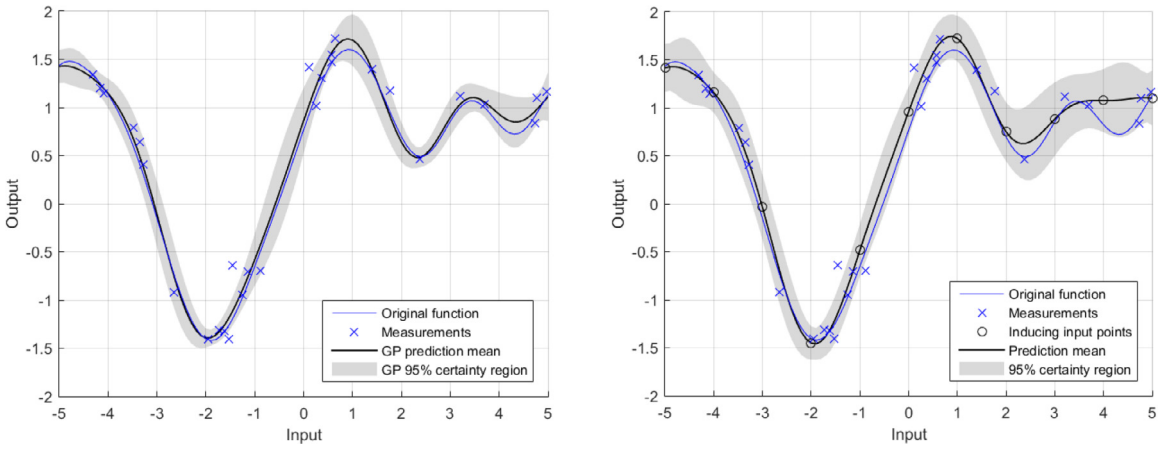

Fig. 1. Predictions of the NIGP algorithm (left) and the SONIG algorithm (right) after $n = 30$ training points have been incorporated. Exact conditions are described in the main text.

than the NIGP algorithm. This is surprising, because the SONIG algorithm can be seen as a computationally efficient approximation of the NIGP algorithm. Further experiments have shown that this is mostly because the SONIG term takes into account the second derivative of the mean in its approximation; see $\mu_{u_i}^{n+1}$ from (16). The NIGP algorithm does not, and if SONIG also does not (detailed experiment results not included here for sake of brevity) the performance of the two algorithms is comparable.

A second thing that can be noticed is that more training points provide a higher accuracy. In particular, even the FITC algorithm (which does not take input noise into account) with 800 training points performs better than the NIGP or SONIG algorithms with 200 training points. It should be noted here that part of the reason is the type of function used: for functions with a steeper slope, it is expected that the NIGP and SONIG algorithms still perform better than FITC.

Finally, it is interesting to note that all algorithms, with the exception of regular GP regression with the exact hyperparameters, are much more optimistic about their predictions than is reasonable. That is, the ratio between the MSE and the mean variance is way larger than the value of 1 which it should have. Ideally, the predicted variance of all algorithms would be significantly higher.

Next, we will look at some plots. To be precise, we will examine algorithms (3) and (4) closer, but subject to only $n = 30$ training points and with $X_u = \{-5, -4, -3, \ldots, 5\}$, giving us $n_u = 11$ inducing input points. The predictions of the two algorithms, for a single random sample function, are shown in Fig. 1.

The most important thing that can be noticed here is that (for both methods) the posterior standard deviation varies with the slope of the to-be-approximated function. When the input is near -2, and the function is nearly flat, the standard deviation is small (well below 0.1). However, when the input is near -3 or $-1/2$, the standard deviation is larger (near 0.2). This is what can be expected, because measurements in these steep regions are much more affected/distorted by the noise, and hence provide less information.

A second thing to be noticed is the difference between the two methods. Especially for $x > 2$, where there are relatively few training points, the SONIG algorithm gives much higher variances. There are two reasons for this. The first is inherent to sparse algorithms. (The FITC algorithm would show a similar trend.) The second reason is inherent to the SONIG algorithm. Whereas regular GP regression (and similarly the NIGP algorithm) uses all training points together, the SONIG algorithm only uses data from previous training points while incorporating a new training point. As a result, when there are relatively few measurements in a certain region, and many of these measurements appear early in the updating process, the accuracy in that region can be expected to be slightly lower. However, as more training points are incorporated, which can be done very efficiently, the problem will quickly disappear.

5.2. Identifying the dynamics of a magneto-rheological fluid damper

In the next experiment we will apply the developed system identification algorithm (Algorithm 2) to a practical problem. In particular, we model the dynamical behavior of a magneto-rheological fluid damper. The measured data for this example was provided by Wang, Sano, Chen, and Huang (2009) and supplied through The MathWorks Inc. (2015), which also discusses various system identification examples using the techniques from Ljung (1999). This example is a common benchmark in system identification applications. It has for instance been used more recently in the context of Gaussian Process State Space Models (GP-SSM) by Svensson and Schön (2017) in their Reduced Rank GP-SSM (RR GP-SSM) algorithm.

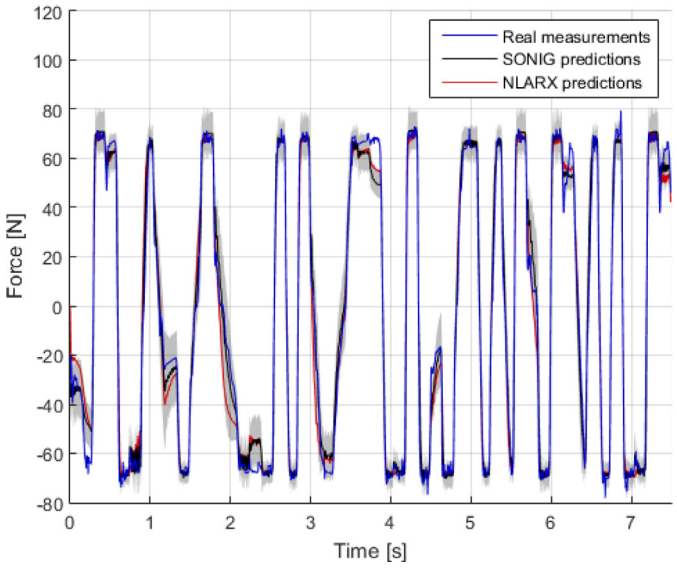

Fig. 2. Prediction of the output of the magneto-rheological fluid damper by the SONIG algorithm (black) compared to the real output (blue). The grey area represents the 95% uncertainty region as given by the algorithm. It shows that in the transition regions (like near $t = 2$s) which the algorithm is less well trained on, the uncertainty is larger. It also shows (for instance near $t = 4.6$ s) that the uncertainty may grow over time. This is a result of the algorithm taking into account its own uncertainty in making future predictions. As comparison, also the best nonlinear ARX model predictions from The MathWorks Inc. (2015) (red) are plotted. It is interesting to note that this model makes very similar errors as the SONIG algorithm, indicating the errors are mostly caused by distortions in the training/evaluation data. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

This example has 3499 measurements provided, sampled every $\Delta t = 0.05$ s. We will use the first 2000 measurements (10 s) for training and the next 1499 measurements (7.5 s) for evaluation. The MathWorks Inc. (2015) recommended to use one past output and three past inputs to predict subsequent outputs. Based on this, we learn a black-box model of the following functional form

$$y_{k+1} = \phi(y_k, u_k, u_{k-1}, u_{k-2}). \tag{25}$$

Hyperparameters were tuned by passing a subset of the data to the NIGP algorithm. Rounded off for simplicity (which did not affect performance) they equaled

$$\Lambda = \operatorname{diag}\left(70^2, 20^2, 10^2, 10^2\right), \qquad \alpha^2 = 70^2,$$
$$\Sigma_{+_x} = \operatorname{diag}\left(2^2, 0.1^2, 0.1^2, 0.1^2\right), \qquad \Sigma_{+_f} = 2^2. \tag{26}$$

After processing a measurement y_{k+1}, the SONIG algorithm provided us with a posterior distribution of y_{k+1}, y_k, u_k, u_{k-1} and u_{k-2}. The marginal posterior distribution of y_{k+1}, u_k and u_{k-1} was then used as prior distribution while incorporating the next measurement. Inducing input points were added online, as specified in Section 4.4, which eventually gave us 32 inducing input points. This is a low number, and as a result, the whole training was done in only a few (roughly 10) seconds. As a result, we did not need to remove or shift inducing input points in any way.

After all training measurements had been used, the SONIG algorithm was given the input data for the remaining 1499 measurements, but not the output data. It had to predict this output data by itself, using each prediction y_k to predict the subsequent y_{k+1}. While doing so, the algorithm also calculated the variance of each prediction y_k, taking this into account while predicting the next output using the techniques from Section 4.5. The resulting predictions can be seen in Fig. 2.

A comparison of the algorithm with various other methods is shown in Table 2. We also added in regular GP regression and NIGP, applied to the ARX model (1), as comparison. This table shows that the SONIG algorithm, when applied in its system identification set-up, can clearly outperform other black-box modeling approaches. It is better than regular GP regression at taking into account uncertainties and better than NIGP mainly due to the reasons explained before. It should be noted here, however, that this is all subject to the proper tuning of hyperparameters and the proper choice of inducing input points. With different hyperparameters or inducing input point selection strategies, the performance of the SONIG algorithm will degrade slightly, though it is still likely to outperform other identification algorithms.

Table 2
Comparison of various system identification models and algorithms when applied to data from the magneto-rheological fluid damper. All algorithms were given 2000 measurements for training and 1499 measurements for evaluation.

Algorithm	RMSE	Source
Linear OE model (4th order)	27.1	The MathWorks Inc. (2015)
Hammerstein–Wiener (4th order)	27.0	The MathWorks Inc. (2015)
NLARX (3rd order, wavelet network)	24.5	The MathWorks Inc. (2015)
NLARX (3rd order, tree partition)	19.3	The MathWorks Inc. (2015)
NIGP	10.2	This paper
GP regression	9.87	This paper
NLARX (3rd order, sigmoid network)	8.24	The MathWorks Inc. (2015)
RR GP-SSM	8.17	Svensson, Solin, Särkkä, and Schön (2016)
SONIG	**7.12**	This paper

6. Conclusions and recommendations

We can conclude that the presented SONIG algorithm works as intended. Just like the FITC algorithm that it expands upon, it is mainly effective when there are more measurements than the NIGP algorithm (or regular GP regression) can handle. The SONIG algorithm can then include the additional measurements very efficiently—incorporating each training point in constant runtime—resulting in a higher accuracy than what the NIGP algorithm could have achieved. However, even when this is not the case, the SONIG algorithm has on average a better performance than the NIGP algorithm, though it still needs the NIGP algorithm for hyperparameter tuning.

Though the SONIG algorithm can be used for any type of regression problem, it has been successfully applied, in its system identification set-up, to a nonlinear black-box system identification problem. With the proper choice of hyperparameters and inducing input points, it outperformed existing state-of-the-art nonlinear system identification algorithms.

Nevertheless, there are still many improvements that can be made to the SONIG algorithm. For instance, to improve the accuracy of the algorithm, we can look at reducing some of the approximating assumptions, like the linearization assumption (13) or the assumption that higher order terms of Σ_+ are negligible.

Another way to improve the accuracy of the algorithm is to increase the number of inducing input points, but this will slow down the algorithm. To compensate, we could look into updating only the few nearest inducing input points (with the highest covariance) when incorporating a new training point. Experience has shown that updates hardly affect inducing inputs far away from the training point (with a low covariance) so this could lead to more efficient updates.

A final possible improvement would concern the addition of a smoothing step in the algorithm. Currently, early measurements are used to provide more accuracy for later measurements, but not vice versa. If we also walk back through the measurements, like in a smoothing algorithm, a higher accuracy might be obtained.

Acknowledgments

This research is supported by the Dutch Technology Foundation STW, which is part of the Netherlands Organisation for Scientific Research (NWO), and which is partly funded by the Ministry of Economic Affairs (Project number: 12173, SMART-WIND). The work was also supported by the Swedish Research Council (VR) via the projects *NewLEADS – New Directions in Learning Dynamical Systems* and *Probabilistic modeling of dynamical systems* (Contract numbers: 621-2016-06079 and 621-2013-5524) and by the Swedish Foundation for Strategic Research (SSF) via the project *ASSEMBLE* (Contract number: RIT15-0012). We would also like to thank Marc Deisenroth for fruitful discussion and in particular for pointing us to the NIGP algorithm.

Appendix. Derivatives of prediction matrices

The SONIG update law (16) contains various derivatives of matrices. Using (8) and (9) we can find them. To do so, we first define the scalar quantity

$$P = \Sigma^n_{++} + \sigma_n^2 = K_{++} + \sigma_n^2 - K_{+u}K_{uu}^{-1}(K_{uu} - \Sigma^n_{uu})K_{uu}^{-1}K_{u+}. \tag{27}$$

We also assume that $m(\boldsymbol{x}) = 0$ for ease of notation. (If not, this can of course be taken into account.) The derivatives of $\boldsymbol{\mu_u^{n+1}}$ and Σ^{n+1}_{uu} can now be found element-wise through

$$\begin{aligned}
\frac{d\boldsymbol{\mu_u^{n+1}}}{dx_{+_j}} &= \Sigma^n_{uu}K_{uu}^{-1}\Bigg(\frac{dK_{u+}}{dx_{+_j}}P^{-1}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu_u^n}\right)\\
&\quad + K_{u+}\frac{dP^{-1}}{dx_{+_j}}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu_u^n}\right) - K_{u+}P^{-1}\frac{dK_{+u}}{dx_{+_j}}K_{uu}^{-1}\boldsymbol{\mu_u^n}\Bigg),\\
\frac{d^2\boldsymbol{\mu_u^{n+1}}}{dx_{+_j}dx_{+_k}} &= \Sigma^n_{uu}K_{uu}^{-1}\Bigg(\frac{d^2K_{u+}}{dx_{+_j}dx_{+_k}}P^{-1}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu_u^n}\right)\\
&\quad + \frac{dK_{u+}}{dx_{+_j}}\frac{dP^{-1}}{dx_{+_k}}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu_u^n}\right) - \frac{dK_{u+}}{dx_{+_j}}P^{-1}\frac{dK_{+u}}{dx_{+_k}}K_{uu}^{-1}\boldsymbol{\mu_u^n}\\
&\quad + \frac{dK_{u+}}{dx_{+_k}}\frac{dP^{-1}}{dx_{+_j}}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu_u^n}\right)\\
&\quad + K_{u+}\frac{d^2P^{-1}}{dx_{+_j}dx_{+_k}}\left(y_+ - K_{+u}K_{uu}^{-1}\boldsymbol{\mu_u^n}\right)\\
&\quad - K_{u+}\frac{dP^{-1}}{dx_{+_j}}\frac{dK_{+u}}{dx_{+_k}}K_{uu}^{-1}\boldsymbol{\mu_u^n} - \frac{dK_{u+}}{dx_{+_k}}P^{-1}\frac{dK_{+u}}{dx_{+_j}}K_{uu}^{-1}\boldsymbol{\mu_u^n}\\
&\quad - K_{u+}\frac{dP^{-1}}{dx_{+_k}}\frac{dK_{+u}}{dx_{+_j}}K_{uu}^{-1}\boldsymbol{\mu_u^n} - K_{u+}P^{-1}\frac{d^2K_{+u}}{dx_{+_j}dx_{+_k}}K_{uu}^{-1}\boldsymbol{\mu_u^n}\Bigg),\\
\frac{d\Sigma^{n+1}_{uu}}{dx_{+_j}} &= -\Sigma^n_{uu}K_{uu}^{-1}\Bigg(\frac{dK_{u+}}{dx_{+_j}}P^{-1}K_{+u} + K_{u+}\frac{dP^{-1}}{dx_{+_j}}K_{+u}\\
&\quad + K_{u+}P^{-1}\frac{dK_{+u}}{dx_{+_j}}\Bigg)K_{uu}^{-1}\Sigma^n_{uu},\\
\frac{d^2\Sigma^{n+1}_{uu}}{dx_{+_j}dx_{+_k}} &= -\Sigma^n_{uu}K_{uu}^{-1}\Bigg(\frac{d^2K_{u+}}{dx_{+_j}dx_{+_k}}P^{-1}K_{+u} + \frac{dK_{u+}}{dx_{+_j}}\frac{dP^{-1}}{dx_{+_k}}K_{+u}\\
&\quad + \frac{dK_{u+}}{dx_{+_j}}P^{-1}\frac{dK_{+u}}{dx_{+_k}} + \frac{dK_{u+}}{dx_{+_k}}\frac{dP^{-1}}{dx_{+_j}}K_{+u} + K_{u+}\frac{d^2P^{-1}}{dx_{+_j}dx_{+_k}}K_{+u}\\
&\quad + K_{u+}\frac{dP^{-1}}{dx_{+_j}}\frac{dK_{+u}}{dx_{+_k}} + \frac{dK_{u+}}{dx_{+_k}}P^{-1}\frac{dK_{+u}}{dx_{+_j}} + K_{u+}\frac{dP^{-1}}{dx_{+_k}}\frac{dK_{+u}}{dx_{+_j}}\\
&\quad + K_{u+}P^{-1}\frac{d^2K_{+u}}{dx_{+_j}dx_{+_k}}\Bigg)K_{uu}^{-1}\Sigma^n_{uu}.
\end{aligned} \tag{28}$$

These expressions contain various additional derivatives. To find them, we need to choose a covariance function. (The above expressions are valid for any covariance function.) If we use the squared exponential covariance function of (4), we can derive

$$\begin{aligned}
\frac{dK_{u_i+}}{d\boldsymbol{x}_+} &= \alpha^2\exp\left(-\frac{1}{2}(\boldsymbol{x_{u_i}} - \boldsymbol{x}_+)^T\Lambda^{-1}(\boldsymbol{x_{u_i}} - \boldsymbol{x}_+)\right)(\boldsymbol{x_{u_i}} - \boldsymbol{x}_+)^T\Lambda^{-1},\\
\frac{d^2K_{u_i+}}{d\boldsymbol{x}_+^2} &= \alpha^2\exp\left(-\frac{1}{2}(\boldsymbol{x_{u_i}} - \boldsymbol{x}_+)^T\Lambda^{-1}(\boldsymbol{x_{u_i}} - \boldsymbol{x}_+)\right)\\
&\quad \left(\Lambda^{-1}(\boldsymbol{x_{u_i}} - \boldsymbol{x}_+)(\boldsymbol{x_{u_i}} - \boldsymbol{x}_+)^T\Lambda^{-1} - \Lambda^{-1}\right),\\
\frac{dP^{-1}}{d\boldsymbol{x}_+} &= -P^{-2}\frac{dP}{d\boldsymbol{x}_+} = 2P^{-2}\left(K_{+u}K_{uu}^{-1}(K_{uu} - \Sigma^n_{uu})K_{uu}^{-1}\frac{dK_{u+}}{d\boldsymbol{x}_+}\right),\\
\frac{d^2P^{-1}}{d\boldsymbol{x}_+^2} &= \frac{d}{d\boldsymbol{x}_+}\left(-P^{-2}\frac{dP}{d\boldsymbol{x}_+}\right) = 2P^{-3}\left(\frac{dP}{d\boldsymbol{x}_+}\right)^T\left(\frac{dP}{d\boldsymbol{x}_+}\right) - P^{-2}\frac{d^2P}{d\boldsymbol{x}_+^2},\\
\frac{dP}{d\boldsymbol{x}_+} &= -2K_{+u}K_{uu}^{-1}(K_{uu} - \Sigma^n_{uu})K_{uu}^{-1}\frac{dK_{u+}}{d\boldsymbol{x}_+},\\
\frac{d^2P}{d\boldsymbol{x}_+^2} &= -2\frac{dK_{+u}}{d\boldsymbol{x}_+}K_{uu}^{-1}(K_{uu} - \Sigma^n_{uu})K_{uu}^{-1}\frac{dK_{u+}}{d\boldsymbol{x}_+}\\
&\quad -2K_{+u}K_{uu}^{-1}(K_{uu} - \Sigma^n_{uu})K_{uu}^{-1}\frac{d^2K_{u+}}{d\boldsymbol{x}_+^2}.
\end{aligned} \tag{29}$$

References

Álvarez, M. A., Rosasco, L., & Lawrence, N. D. (2012). Kernels for vector-valued functions: A review. *Foundations and Trends in Machine Learning, 4*(3), 195–266.

Bijl, H. (2017). SONIG source code. https://github.com/HildoBijl/SONIG/tree/v1.0.

Bijl, H., van Wingerden, J.-W., Schön, T. B., & Verhaegen, M. (2015). Online sparse Gaussian process regression using FITC and PITC approximations. In *Proceedings of the IFAC symposium on system identification (SYSID), Beijing, China*.

Candela, J. Q., Girard, A., Larsen, J., & Rasmussen, C. E. (2003). Propagation of uncertainty in Bayesian kernel models – Application to multiple-step ahead forecasting. In *Proceedings of advances in neural information processing systems* (pp. 701–704). MIT Press.

Candela, J. Q., & Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research, 6*, 1939–1959.

Chalupka, K., Williams, C. K. I., & Murray, I. (2013). A framework for evaluating approximation methods for Gaussian process regression. *Machine Learning Research, 14*, 333–350.

Chen, T., Ohlsson, H., & Ljung, L. (2012). On the estimation of transfer functions, regularizations and Gaussian processes–revisited. *Automatica, 48*(8), 1525–1535.

Csató, L., & Opper, M. (2002). Sparse online Gaussian processes. *Neural Computation, 14*(3), 641–669.

Dallaire, P., Besse, C., & Chaib-draa, B. (2009). Learning Gaussian process models from uncertain data. In *Proceedings of the 16th international conference on neural information processing*.

Damianou, A. C., Titsias, M. K., & Lawrence, N. D. (2016). Variational inference for latent variables and uncertain inputs in Gaussian processes. *Journal of Machine Learning Research, 17*(42), 1–62.

Deisenroth, M. P. (2010). *Efficient reinforcement learning using Gaussian processes*. Karlsruhe Institute of Technology Ph.D. thesis..

Deisenroth, M. P., & Ng, J. W. (2015). Distributed Gaussian processes. In *Proceedings of the international conference on machine learning (ICML)*. Lille, France.

Deisenroth, M. P., & Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the international conference on machine learning (ICML), Bellevue, Washington, USA* (pp. 465–472). ACM Press.

Frigola, R., Chen, Y., & Rasmussen, C. E. (2014). Variational Gaussian process state-space models. In *Proceedings of advances in neural information processing systems (NIPS)*.

Frigola, R., Lindsten, F., Schön, T. B., & Rasmussen, C. E. (2013). Bayesian inference and learning in Gaussian process state-space models with particle MCMC. In *Proceedings of advances in neural information processing systems (NIPS): 26*. Lake Tahoe, NV, USA.

Gal, Y., van der Wilk, M., & Rasmussen, C. E. (2014). Distributed variational inference in sparse Gaussian process regression and latent variable models. In *Proceedings of advances in neural information processing systems (NIPS)*.

Girard, A., & Murray-Smith, R. (2003). Learning a Gaussian Process Model with Uncertain Inputs. *Technical Report 144*. Department of Computing Science, University of Glasgow.

Girard, A., Rasmussen, C. E., Candela, J. Q., & Murray-Smith, R. (2003). Gaussian process priors with uncertain inputs – Application to multiple-step ahead time series forecasting. In *Proceedings of advances in neural information processing systems* (pp. 529–536). MIT Press.

Goldberg, P. W., Williams, C. K. I., & Bishop, C. M. (1997). Regression with input-dependent noise: A Gaussian process treatment. In *Proceedings of advances in neural information processing systems (NIPS): vol. 10* (pp. 493–499). MIT Press.

Hensman, J., Nicoló, F., & Lawrence, N. D. (2013). Gaussian processes for big data. In *Proceedings of the twenty-ninth conference on uncertainty in artificial intelligence (UAI), Bellevue, Washington, USA*.

Huber, M. F. (2013). Recursive Gaussian process regression. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing (ICASSP), Vancouver, Canada* (pp. 3362–3366).

Huber, M. F. (2014). Recursive Gaussian process: On-line regression and learning. *Pattern Recognition Letters, 45*, 85–91.

Kocijan, J. (2016). *Modelling and control of dynamic systems using Gaussian process models*. Basel, Switzerland: Springer.

Kocijan, J., Girard, A., Banko, B., & Murray-Smith, R. (2005). Dynamic systems identification with Gaussian processes. *Mathematical and Computer Modelling of Dynamical Systems, 11*(4), 411—424.

Kou, P., Gao, F., & Guan, X. (2013). Sparse online warped Gaussian process for wind power probabilistic forecasting. *Applied Energy, 108*, 410–428.

Le, Q. V., & Smola, A. J. (2005). Heteroscedastic Gaussian process regression. In *Proceedings of the international conference on machine learning (ICML), Bonn, Germany*.

Ljung, L. (1999). *System identification: Theory for the user*. Upper Saddle River, NJ, USA: Prentice Hall.

McHutchon, A. (2014). *Nonlinear modelling and control using Gaussian processes*. Churchill College Ph.D. thesis.

McHutchon, A., & Rasmussen, C. E. (2011). Gaussian process training with input noise. In *Proceedings of advances in neural information processing systems (NIPS), Granada, Spain* (pp. 1341–1349).

Pillonetto, G., Chiuso, A., & De Nicolao, G. (2011). Prediction error identification of linear systems: a nonparametric Gaussian regression approach. *Automatica, 47*(2), 291—305.

Pillonetto, G., & De Nicolao, G. (2010). A new kernel-based approach for linear system identification. *Automatica, 46*(1), 81—93.

Ranganathan, A., Yang, M.-H., & Ho, J. (2011). Online sparse Gaussian process regression and its applications. *IEEE Transactions on Image, 20*(2), 391–404.

Rasmussen, C. E., & Williams, C. K. (2006). *Gaussian processes for machine learning*. MIT Press.

Seeger, M., Williams, C. K., & Lawrence, N. D. (2003). Fast forward selection to speed up sparse Gaussian process regression. In *Proceedings of the ninth international workshop on artificial intelligence and statistics (AISTATS)*. Key West, FL, USA.

Smola, A. J., & Bartlett, P. (2001). Sparse greedy Gaussian process regression. In *Proceedings of advances in neural information processing systems (NIPS), Vancouver, Canada* (pp. 619–625).

Snelson, E., & Ghahramani, Z. (2006a). Sparse Gaussian processes using pseudo-inputs. In *Proceedings of advances in neural information processing systems (NIPS)* (pp. 1257–1264).

Snelson, E., & Ghahramani, Z. (2006b). Variable noise and dimensionality reduction for sparse Gaussian processes. In *Proceedings of the twenty-second conference on uncertainty in artificial intelligence (UAI), Cambridge, Massachusetts, USA*.

Svensson, A., & Schön, T. B. (2017). A flexible state space model for learning nonlinear dynamical systems. *Automatica, 80*, 189–199.

Svensson, A., Solin, A., Särkkä, S., & Schön, T. B. (2016). Computationally efficient Bayesian learning of Gaussian process state space models. In *Proceedings of the 19th international conference on artificial intelligence and statistics (AISTATS), Cadiz, Spain*.

The MathWorks Inc. (2015). Nonlinear modeling of a magneto-rheological fluid damper.Example file provided by Matlab ® R2015b System Identification Toolbox™. Available at http://mathworks.com/help/ident/examples/nonlinear-modeling-of-a-magneto-rheological-fluid-damper.html.

Titsias, M. K. (2009). Variational learning of inducing variables in sparse Gaussian processes. In *Proceedings of the international conference on artificial intelligence and statistics (AISTATS)*. Clearwater Beach, FL, USA.

Titsias, M. K., & Lawrence, N. D. (2010). Bayesian Gaussian process latent variable model. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics (AISTATS 13)* (pp. 844–851).

Wang, C., & Neal, R. M. (2012). Gaussian process regression with heteroscedastic or non-Gaussian residuals. *Technical Report*. arXiv.org

Wang, J., Sano, A., Chen, T., & Huang, B. (2009). Identification of Hammerstein systems without explicit parameterization of nonlinearity. *International Journal of Control, 82*(5), 937–952.