Online Sparse Gaussian Process Training with Input Noise

Hildo Bijl
Delft Center for Systems and Control
Delft University of Technology
Delft, The Netherlands
H.J.BIJL@TUDELFT.NL

Thomas B. Schön
Department of Information Technology
Uppsala University
Uppsala, Sweden
THOMAS.SCHON@IT.UU.SE

Jan-Willem van Wingerden
Delft Center for Systems and Control
Delft University of Technology
Delft, The Netherlands
J.W.VANWINGERDEN@TUDELFT.NL

Michel Verhaegen
Delft Center for Systems and Control
Delft University of Technology
Delft, The Netherlands
M.VERHAEGEN@TUDELFT.NL

Abstract
Gaussian process regression traditionally has 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 measurements in constant runtime. A comparison has shown that it is more accurate than similar existing regression algorithms. In addition, the algorithm can be applied to non-linear black-box system modeling, where its performance is competitive with non-linear ARX models.

Keywords: Gaussian Processes, Regression, Machine Learning, Sparse Methods, System Identification

1. Introduction
Gaussian Process (GP) regression (see Rasmussen and Williams (2006) for an introduction) has been increasing in popularity recently. Fundamentally, it works similarly to other function approximation algorithms like neural networks and support vector machines. However, it has its basis in Bayesian probability theory. The result is that GP regression does not only give estimates of function outputs, but also variances – information about the uncertainty – of these estimates. This advantage has proven to be useful in a variety of applications, like those by Deisenroth and Rasmussen (2011) or Hensman et al. (2013).

Traditional GP regression is not without its limitations though. In fact, there are three important limitations, which we will outline here, also discussing the current state of the

©2016 Hildo Bijl, Thomas B. Schön, Jan-Willem van Wingerden, Michel Verhaegen.
literature on these limitations. The contribution of this paper is a new GP algorithm that
overcomes all three of these limitations simultaneously, which has not been done in this way
before.

1.1 Using stochastic input points

The first limitation is that the GP regression algorithm assumes that the input points are
deterministic. This assumption concerns both the measurement points \( x_m \) and the trial
points \( x_s \). For noisy (stochastic) trial points, we can work around the problem by applying
moment matching, as explained by Deisenroth (2010). This technique can subsequently
also be expanded for noisy measurement points using the ideas from Dallaire et al. (2009).

However, the problem with this technique is that it only adjusts the uncertainty (the
covariance) of points based on prior data and does not involve posterior data (derived from
measurement values) at all. As such, its performance is significantly limited. A method
which does include posterior data is given by Girard and Murray-Smith (2003), but this
method only works for noisy trial points and not for noisy measurement points. To the
best of our knowledge, there is only one previously available method which both includes
posterior data and works with noisy measurement points, which is the NIGP algorithm
by McHutchon and Rasmussen (2011). This is the method we will be expanding upon in
this paper.

It should be noted that there is also another way to take into account the effects of
noisy measurement points: by assuming the noise variance varies over the input space;
a feature called heteroscedasticity. This has been investigated quite in-depth by, among
others, Goldberg et al. (1997), Le and Smola (2005), Wang and Neal (2012) and Snelson and
Ghahramani (2006a), 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 our
particular problem.

1.2 Reducing the computational complexity

The second limitation of Gaussian process regression is its computational complexity. The
straightforward regression algorithm has a runtime of \( O(n^3) \), with \( n \) the number of measure-
ments. This makes it hard to combine GPs with the recent trend of using more and more
data. As a solution, sparse methods have been proposed. An overview of such methods
is given in Candela and Rasmussen (2005), summarizing various contributions from Smola
and Bartlett (2001), Csató and Opper (2002), Seeger et al. (2003) and Snelson and Ghahra-
man (2006b) into a comprehensive framework. With these methods, and particularly with
the FITC method which we will apply in this paper (see Candela and Rasmussen (2005))
the runtime can be reduced to being linear with respect to \( n \). More recent work on sparse
GPs involve the use of variational inference (Titsias, 2009; Hensman et al., 2013; Gal et al.,
2014).

An alternative to the sparse GPs is offered by the distributed Gaussian process con-
struction recently developed by Deisenroth and Ng (2015). Their construction makes use of
the full data set and instead reduces the computational complexity by distributing both the
computational load and the memory load amongst a large set of independent computational
units.
1.3 Online application

The third limitation is the difficulty with which GP regression algorithms can add new measurements. Traditionally, when a new measurement $n + 1$ is added to the existing set of $n$ measurement points, the GP regression algorithm would have to redo all computations to take this new measurement into account. For online applications, in which new measurement data continuously arrive, this seems unnecessarily inefficient.

And indeed, various online GP regression methods exist which approximate a GP through a relatively small subset of $n_u$ inducing input (basis) points. Many of these techniques (Csató and Opper, 2002; Candela and Rasmussen, 2005; Ranganathan et al., 2011; Kou et al., 2013; Hensman et al., 2013) are designed for sparse GP algorithms, but they do not work for any set of inducing input points. The exceptions are Huber (2013, 2014) and Bijl et al. (2015), which both give an online method of applying the FITC algorithm for any set of inducing input points.

1.4 Contribution and outline of the paper

In this paper the solutions to these three limitations will be combined into a single novel algorithm, called Sparse Online Noisy Input GP (SONIG). This algorithm will be capable of adding information obtained from noisy input points in an online way to Gaussian process predictions. To the best of our knowledge, an algorithm with these features has not been published before. In addition, we will subsequently expand this algorithm, enabling it to identify non-linear system dynamics. The resulting algorithm, called System Identification using Sparse Online GPs (SISOG), will be applied to a magneto-rheological fluid damper example.

This paper is set up as follows. We start by defining the notation we use in Section 2, as well as outlining the equations normally used in GP regression. In Section 3 we expand these methods, making them applicable to GP regression with noisy input points. This results in the main version of the SONIG algorithm. Section 4 subsequently outlines various ways to expand on this SONIG algorithm. Test results of the main SONIG algorithm and of an extension focused on system identification are then shown in Section 5. Section 6 finally gives conclusions and recommendations.

2. Overview of Gaussian process regression methods

In this section we will give an overview of the existing theory on GP regression. Specifically, we introduce the notation that we will use as well as the method of thought behind the regression algorithm. Section 2.1 mainly covers theory discussed by Rasmussen and Williams (2006) on regular GP regression. Section 2.2 discusses sparse GP with insights from Candela and Rasmussen (2005) and Section 2.3 looks at the online implementation of these algorithms using recent results from Bijl et al. (2015).

2.1 Regular Gaussian process regression

In GP regression we aim to approximate a function $f(x)$. To do this, we first obtain various measurements $(x_{m_1}, y_1), (x_{m_2}, y_2), \ldots, (x_{m_n}, y_n)$, with $x_m$ the measurement input point, $y$ the measured function output and $n$ the number of measurements. Here we assume that the
measurement outputs are corrupted by noise, such that \( y_i = f(x_{m_i}) + \epsilon \), with \( \epsilon \sim \mathcal{N}(0, \sigma_n^2) \) being Gaussian white noise with variance \( \sigma_n^2 \).

As shorthand notation, we merge all the measurement points \( x_{m_i} \) into a measurement set \( X_m \) and all output values \( y_i \) into an output vector \( y \). We now write the noiseless function output \( f(X_m) \) as \( f_m \), such that \( y = f_m + \epsilon \), with \( \epsilon \sim \mathcal{N}(0, \Sigma_n) \) and \( \Sigma_n = \sigma_n^2 I \).

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

\[
\begin{bmatrix} f_m^n \\ f_{*}^n \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} m(X_m) \\ m(X_{*}) \end{bmatrix}, \begin{bmatrix} k(X_m, X_m) & k(X_m, X_{*}) \\ k(X_{*}, X_m) & k(X_{*}, X_{*}) \end{bmatrix} \right) = \mathcal{N} \left( \begin{bmatrix} m_m \\ m_{*} \end{bmatrix}, \begin{bmatrix} K_{mm} & K_{ms} \\ K_{sm} & K_{ss} \end{bmatrix} \right),
\]

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

\[
k(x, x') = \alpha^2 \exp \left( -\frac{1}{2} (x - x')^T \Lambda^{-1} (x - x') \right),
\]

with \( \alpha \) a characteristic output length scale and \( \Lambda \) 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 (1) we can find the posterior distribution of both \( f_m \) and \( f_{*} \) given \( y \). This can be derived in a variety of ways, but they all result in the same expressions, being

\[
\begin{bmatrix} f_m^n \\ f_{*}^n \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} \mu_m^n \\ \mu_{*}^n \end{bmatrix}, \begin{bmatrix} \Sigma_{mm}^n & \Sigma_{ms}^n \\ \Sigma_{sm}^n & \Sigma_{ss}^n \end{bmatrix} \right),
\]

\[
\begin{bmatrix} \mu_m^n \\ \mu_{*}^n \end{bmatrix} = \begin{bmatrix} (K_{mm} + \Sigma_n^{-1})^{-1}(K_{mm}^{-1}m_m + \Sigma_n^{-1}y) \\ m_{*} + K_{sm}(K_{mm} + \Sigma_n)^{-1}(y - m_m) \end{bmatrix},
\]

\[
\begin{bmatrix} \Sigma_{mm}^n & \Sigma_{ms}^n \\ \Sigma_{sm}^n & \Sigma_{ss}^n \end{bmatrix} = \begin{bmatrix} (K_{mm}^{-1} + \Sigma_n^{-1})^{-1} & \Sigma_n(K_{mm} + \Sigma_n)^{-1}K_{ms} \\ K_{sm}(K_{mm} + \Sigma_n)^{-1}\Sigma_n & K_{ss} - K_{sm}(K_{mm} + \Sigma_n)^{-1}K_{ms} \end{bmatrix}.
\]

Note here that while we use \( m \) and \( K \) to denote properties of prior distributions, we use \( \mu \) and \( \Sigma \) for posterior distributions. The superscript \( n \) indicates these are posteriors taking \( n \) measurements into account.

### 2.2 Sparse Gaussian process regression

To reduce the computational complexity of predicting the function value \( f_{*} \) at trial points \( X_{*} \), we can also use sparse GP regression. The first step in this process is to introduce a set of so-called inducing input points \( X_u \). We choose these points ourselves, either evenly distributed over the input space, or with some more inducing input points in regions we
are specifically interested in. We then first predict the distribution of \( f_u \), identically to (3), through

\[
\begin{bmatrix}
  \mu_n^u \\
  \Sigma_n^u
\end{bmatrix} = \begin{bmatrix}
  \mu_n^u \\
  \Sigma_n^u
\end{bmatrix} = \begin{bmatrix}
  m_u + K_{um} (K_{mm} + \Sigma_n)^{-1} (y - m_m) \\
  K_{uu} (K_{mm} + \Sigma_n)^{-1} \Sigma_n
\end{bmatrix},
\]

(4)

In this expression, it is usually only the distribution \( f_u^n \sim N(\mu_u^n, \Sigma_u^n) \) that is useful. This is because, after we know the distribution of \( f_u \), we throw away all measurement data and only use this distribution to predict \( f_u \). (Mathematically, this comes down to assuming that \( f_m \) and \( f_u \) are conditionally independent, given \( f_u \).) Taking care not to use prior data twice, we then find that

\[
\begin{bmatrix}
  \mu_n^u \\
  \Sigma_n^u
\end{bmatrix} = \begin{bmatrix}
  m_u + K_{su} K_{uu}^{-1} (y - m_u) \\
  K_{uu} K_{uu}^{-1} (y - m_u)
\end{bmatrix},
\]

(5)

Given that the number of inducing input points \( n_u \) is much smaller than the number of measurements \( n \), this will result in a fast prediction for \( f_u \). However, predicting \( f_u \) is still slow. One way of making it faster is by first predicting (finding the distribution of) \( f_u \), only based on the first measurement \((x_{m1}, y_1)\). Then we do the same, only based on the second measurement, and so on. Finally, we merge all distributions of \( f_u \) which we have found together, taking care to use prior data only once. (Mathematically, this comes down to assuming that \( f_{m1} \) and \( f_{m2} \) are conditionally independent, given \( f_u \).) This will give us the FITC regression equations, in their (computationally) most efficient form

\[
\begin{bmatrix}
  \mu_n^m \\
  \Sigma_n^m
\end{bmatrix} = \begin{bmatrix}
  m_m + \Sigma_m^{-1} (y - m_m) \\
  \Lambda_{mm}^{-1} \Sigma_n
\end{bmatrix},
\]

\[
\begin{bmatrix}
  \mu_n^u \\
  \Sigma_n^u
\end{bmatrix} = \begin{bmatrix}
  m_u + \Sigma_u^{-1} (y - m_m) \\
  \Lambda_{mm}^{-1} \Sigma_u
\end{bmatrix},
\]

(6)

Here we have used the shorthand notation \( \Delta = K_{uu} + K_{um} (\Lambda_{mm} + \Sigma_n)^{-1} K_{mu} \), with \( \Delta \) being the function which sets all non-diagonal elements of the given matrix to zero. It is important to note that the above set of expressions does not equal (4), but is merely a computationally efficient approximation.
2.3 Sparse Online Gaussian process regression

The methods developed so far can also be implemented in an online fashion. Suppose we get a new measurement \((x_{m+1}, y_{m+1})\), whose notation we will shorten to \((x_{m+}, y_{+})\). To incorporate it, we first predict the posterior distribution of \(f_+ = f(x_{m+})\) using only the first \(n\) measurements. This results in

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

\[
\begin{bmatrix} \mu_{+u}^{n+1} 
 \mu_{+n}^{n+1} \end{bmatrix} = \begin{bmatrix} \mu_{+u}^n + \frac{\Sigma_{u+}^n}{\Sigma_{++}^n} (\mu_{+n}^n - \mu_{+u}^n) 
 \mu_{+n}^n + \frac{\Sigma_{++}^n}{\Sigma_{++}^n} \Sigma_{+u}^n (\mu_{+u}^n - \mu_{+n}^n) \end{bmatrix},
\]

\[
\begin{bmatrix} \Sigma_{uu}^{n+1} & \Sigma_{u+}^{n+1} 
 \Sigma_{+u}^{n+1} & \Sigma_{++}^{n+1} \end{bmatrix} = \begin{bmatrix} \Sigma_{uu}^n + \frac{\Sigma_{u+}^n}{\Sigma_{++}^n} \Sigma_{+u}^n + \frac{\Sigma_{++}^n}{\Sigma_{++}^n} \Sigma_{+u}^n 
 \Sigma_{+u}^n + \frac{\Sigma_{++}^n}{\Sigma_{++}^n} \Sigma_{+u}^n + \frac{\Sigma_{++}^n}{\Sigma_{++}^n} \Sigma_{++}^n \end{bmatrix}.
\]

Then we do the same using only our new measurement (using (4)). If we merge the results together in the proper way, we end up with

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

\[
\begin{bmatrix} \mu_{+u}^{n+1} 
 \mu_{+n}^{n+1} \end{bmatrix} = \begin{bmatrix} \mu_{+u}^n + \frac{\Sigma_{u+}^n}{\Sigma_{++}^n} (\mu_{+n}^n - \mu_{+u}^n) 
 \mu_{+n}^n + \frac{\Sigma_{++}^n}{\Sigma_{++}^n} \Sigma_{+u}^n (\mu_{+u}^n - \mu_{+n}^n) \end{bmatrix},
\]

\[
\begin{bmatrix} \Sigma_{uu}^{n+1} & \Sigma_{u+}^{n+1} 
 \Sigma_{+u}^{n+1} & \Sigma_{++}^{n+1} \end{bmatrix} = \begin{bmatrix} \Sigma_{uu}^n + \frac{\Sigma_{u+}^n}{\Sigma_{++}^n} \Sigma_{+u}^n + \frac{\Sigma_{++}^n}{\Sigma_{++}^n} \Sigma_{+u}^n 
 \Sigma_{+u}^n + \frac{\Sigma_{++}^n}{\Sigma_{++}^n} \Sigma_{+u}^n + \frac{\Sigma_{++}^n}{\Sigma_{++}^n} \Sigma_{++}^n \end{bmatrix}.
\]

This expression (or at least the part relating to \(f_u\)) is the update law for the FITC algorithm. It tells us exactly how the distribution of \(f_u^{n+1}\) (both \(\mu_u^{n+1}\) and \(\Sigma_u^{n+1}\)) depends on \(x_+\) and \(y_+\). And with this distribution, we can always predict new trial points \(f_u\) in an efficient way through (5).

3. Expanding the algorithm for stochastic measurement points

In this section we will expand on the existing theory to enable the online FITC algorithm to handle stochastic measurement points. Though we make use of many existing ideas, the combination of these ideas in this form has not been done before.

Consider the case where we know the distribution \(f_u^n \sim \mathcal{N} (\mu_u^n, \Sigma_{uu}^n)\). (Initially we have \(f_u^0 \sim \mathcal{N} (\mu_u^0, \Sigma_{uu}^0) = \mathcal{N} (m_u, K_{uu})\).) We now obtain a new measurement at some unknown input point \(x_+\). Also the true function output \(f_+ = f(x_+\) is unknown. Our measurements do give us values \(\hat{x}_+\) and \(\hat{f}_+\) approximating these. (Note that \(\hat{f}_+\) and \(y_+\) are exactly the same, but for the sake of uniform notation, we have renamed \(y_+\) to \(\hat{f}_+\).) This measurement basically tells us that \(x_+ \sim \mathcal{N}(\hat{x}_+, \Sigma_{++})\) and \(f_+ \sim \mathcal{N}(\hat{f}_+, \Sigma_{++})\). (Here \(\Sigma_{++}\) are the same.) We assume that the noise on the input and the output is independent, and hence that \(x_+\) and \(f_+\) are a priori not correlated.

The question now arises how we should calculate \(p(f_u^{n+1})\). Given \(x_+\), we can calculate the distribution of \(f_u^{n+1}\) using (8). This equation hence tells us \(p(f_u^{n+1} | x_+, \hat{x}_+, \hat{f}_+, f_u^n)\). It now follows that

\[
p(f_u^{n+1} | \hat{x}_+, \hat{f}_+, f_u^n) = \int p(f_u^{n+1} | x_+, \hat{x}_+, \hat{f}_+, f_u^n) p(x_+ | \hat{x}_+, \hat{f}_+, f_u^n) dx_+,
\]
where the first probability $p(f_{u+1}^n | x_+, \hat{x}_+, f_u^n)$ follows from (8). The second probability $p(x_+ | \hat{x}_+, \hat{f}_+, f_u^n)$ is the posterior distribution of $x_+$, given both $f_u^n$ and the new measurement. We will find this latter probability first in Section 3.1 and show how to implement it in Section 3.2. We then merge all ideas together into the resulting SONIG algorithm in Section 3.3.

### 3.1 The posterior distribution of the measurement point

The posterior distribution of $x_+$ can be found through Bayes’ theorem,

$$p(x_+ | \hat{f}_+, \hat{x}_+, f_u^n) = \frac{p(\hat{f}_+ | x_+, \hat{x}_+, f_u^n)p(x_+ | \hat{x}_+, f_u^n)}{p(\hat{f}_+ | \hat{x}_+, f_u^n)}.$$  \hspace{1cm} (10)

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

$$p(\hat{f}_+ | x_+, \hat{x}_+, f_u^n) = p(\hat{f}_+ | x_+, f_u^n) = \mathcal{N} \left( \mu^n_+, \Sigma^n_{++} + \Sigma_{+\mu} \right),$$  \hspace{1cm} (11)

where $\mu^n_+$ and $\Sigma^n_{++}$ follow from (7). Note that both these quantities depend on $x_+$ in a non-linear way. Because of this, the resulting probability $p(x_+ | \hat{f}_+, \hat{x}_+, f_u^n)$ will be non-Gaussian. To work around this problem, we have to make some simplifying assumptions. What we can do, similarly to Girard and Murray-Smith (2003), is to linearize the Gaussian process $\hat{f}_+$ (which depends on $x_+$) around a point $\bar{x}_+$. That is, we assume that

$$p(\hat{f}_+ | x_+, f_u^n) = \mathcal{N} \left( \mu^n_+ (\bar{x}_+) + \frac{d\mu^n_+ (\bar{x}_+)}{dx_+} (x_+ - \bar{x}_+), \Sigma^n_{++}(\bar{x}_+) + \Sigma_{+\mu} \right).$$  \hspace{1cm} (12)

(Note that we use the notation that a derivative of a scalar with respect to a vector is a row vector.) Hence, the mean varies linearly with $x_+$, but the covariance is constant everywhere. With this approximation, the solution for $p(x_+ | \hat{f}_+, \hat{x}_+, f_u^n)$ is Gaussian. It can be solved analytically, by manually working out all the matrix exponents, as

$$p(x_+ | \hat{f}_+, \hat{x}_+, f_u^n) = \mathcal{N} \left( \bar{x}_+^{n+1}, \Sigma_{++}^{n+1} \right),$$

$$\bar{x}_+^{n+1} = \hat{x}_+ + \Sigma_{++}^{n+1} \left( \frac{d\mu^n_+ (\bar{x}_+)}{dx_+} \right)^T \left( \Sigma^n_{++}(\bar{x}_+) + \Sigma_{+\mu} \right)^{-1}$$

$$\left( \frac{d\mu^n_+ (\bar{x}_+)}{dx_+} (\bar{x}_+ - \hat{x}_+) + \left( \hat{f}_+ - \mu^n_+(\bar{x}_+) \right) \right),$$

$$\Sigma_{++}^{n+1} = \left( \frac{d\mu^n_+ (\bar{x}_+)}{dx_+} \right)^T \left( \Sigma^n_{++}(\bar{x}_+) + \Sigma_{+\mu} \right)^{-1} \left( \frac{d\mu^n_+ (\bar{x}_+)}{dx_+} \right) + \Sigma_{+\mu}^{-1}. \hspace{1cm} (13)$$

The remaining question is which point $\bar{x}_+$ we should linearize around. The above equation is easiest to apply when we choose $\bar{x}_+ = \mu_+$, but when $\left( \hat{f}_+ - \mu^n_+(\bar{x}_+) \right)$ is large, this may result in inaccuracies due to the linearization. It is generally more accurate to choose $\bar{x}_+$ as $\bar{x}_+^{n+1}$. However, $\bar{x}_+^{n+1}$ is initially unknown, so it may be necessary to apply the above equation a few times, each time resetting $\bar{x}_+$ to the latest value of $\bar{x}_+^{n+1}$ that was found, to find the most accurate posterior distribution of $x_+$.
3.2 Updating the inducing input points

We are now ready to derive a solution to the integral in (9). To do so, we again need various assumptions. Here we use less strict assumptions than in the above derivation: we only assume that \( x_+ \) is independent of \( f_u \) and that \( \Sigma^2_{x+} \) and higher powers of \( \Sigma_{+,x} \) are negligible. Both assumptions are reasonable to make for most applications with reasonably accurate measurements.

To solve (9), we will not linearize the GP \( f_u(x_+) \), but instead approximate it by its full Taylor expansion around \( \hat{x}^{n+1} \). Element-wise, we write this as,

\[
f^{n+1}_{u_i}(x_+) = f^{n+1}_{u_i}(\hat{x}^{n+1}) + \frac{df^{n+1}_{u_i}(\hat{x}^{n+1})}{dx_+} (x_+ - \hat{x}^{n+1}) + \frac{1}{2} (x_+ - \hat{x}^{n+1})^T \frac{d^2 f^{n+1}_{u_i}(\hat{x}^{n+1})}{dx^2_+} (x_+ - \hat{x}^{n+1}) + \ldots \quad (14)
\]

Within this approximation, Girard and Murray-Smith (2003) made the further assumption that higher order derivatives like \( \frac{d^2 f_u}{dx^2} \) were negligible. We do not make this assumption, but our assumption that higher powers of \( \Sigma_{+,x} \) are negligible identically causes these higher order derivatives to drop out. As added bonus, this latter assumption is easier to justify.

It is important to realize here that not only \( x_+ \) but also \( f^{n+1}_u(\hat{x}^{n+1}_+) \) and its derivatives are Gaussian random variables. (The derivative of a Gaussian process is also a Gaussian process.) Hence, in order to find the expected value of \( f^{n+1}_u \), we have to take the expectation with respect to both the function values \( f^{n+1}_u \) and the input point \( x_+ \). If we look at the equation element-wise, the result will be

\[
E_{f_u x_+} [f^{n+1}_u] = E_{f_u} \left[ f^{n+1}_{u_i}(\hat{x}^{n+1}) \right] + E_{f_u} \left[ \frac{df^{n+1}_{u_i}(\hat{x}^{n+1})}{dx_+} \right] E_{x_+} \left[ x_+ - \hat{x}^{n+1}_+ \right] \\
+ \frac{1}{2} E_{x_+} \left[ (x_+ - \hat{x}^{n+1}_+)^T E_{f_u} \left[ \frac{d^2 f^{n+1}_{u_i}(\hat{x}^{n+1})}{dx^2_+} \right] (x_+ - \hat{x}^{n+1}_+) \right] + \ldots \\
= E_{f_u} \left[ f^{n+1}_{u_i}(\hat{x}^{n+1}) \right] + \frac{1}{2} \text{tr} \left( \frac{d^2 E_{f_u} [f^{n+1}_{u_i}(\hat{x}^{n+1})]}{dx^2_+} \Sigma_{+,x}^{n+1} \right) \\
= \mu_{u_i}^{n+1}(\hat{x}^{n+1}_+) + \frac{1}{2} \text{tr} \left( \frac{d^2 \mu_{u_i}^{n+1}(\hat{x}^{n+1}_+)}{dx^2_+} \Sigma_{+,x}^{n+1} \right). \quad (15)
\]

Note that, because of our assumption that \( \Sigma^2_{x+} \) and higher powers of \( \Sigma_{+,x} \) are negligible, all future terms drop out. Additionally, we have used our assumption that \( x_+ \) and \( f_u \) are independent. We have also used the fact that both the derivative and the expectation operators are linear operators, allowing us to interchange their order of application.

We can find the covariance matrix of \( f^{n+1}_u \) with a similar but much more lengthy procedure. For that reason, we have put the derivation in Appendix A and only present
the final result here element-wise as

\[ V_{f_+} \left[ f^{u+1}_u \right]_{i,j} = \sum_{u_i}^n \left( \frac{d\mu_i^{n+1} (\hat{x}^{n+1} + 1)}{dx_+} \right) \Sigma_{+x}^{n+1} \left( \frac{d\mu_i^{n+1} (\hat{x}^{n+1} + 1)}{dx_+} \right)^T + \frac{1}{2} \text{tr} \left( \left( \frac{d^2\Sigma_{u_i}^{n+1} (\hat{x}^{n+1} + 1)}{dx_+^2} \right) \Sigma_{+x}^{n+1} \right) \].

(16)

Finding all the necessary derivatives for the above equations can be a daunting task, especially for non-scalar inputs \( x_+ \), but the mathematics are relatively straightforward, so for this we refer to Appendix B.

It is interesting to compare the expressions we found with those used by McHutchon and Rasmussen (2011) in their NIGP algorithm. If we do, we find that for both (15) and (16), McHutchon and Rasmussen (2011) did not include the last term of the equation in their paper. That is, they did not take second derivatives into account. In their simulation code, which is available online, they did include the last term of (16) and found (correctly) that its effect was not significant. However, they still did not include the last term of (15), and in the test results section (see Section 5.1) we will find that this term actually is significant. As such, (15) and (16) also serve as an improvement with respect to the NIGP algorithm.

3.3 The SONIG algorithm

Using all the ideas developed this far, we can set up an online regression algorithm. The resulting Sparse Online Noisy Input GP (SONIG) algorithm is presented in Algorithm 1. This algorithm is computationally efficient, in the sense that a single updating step (incorporating one measurement) can be done in constant runtime with respect to the number of measurements already processed. The runtime does depend on the number of inducing input points through \( O(n_u^3) \). Hence, increasing the number of inducing input points does slow down the algorithm. Generally, \( n_u \) stays below a hundred though, preventing this from becoming a significant issue.

4. Extensions of the SONIG algorithm

In the previous section, we have presented the basic idea behind the SONIG algorithm. There are various further details that can be derived and implemented in the algorithm. For instance, the algorithm can deal with multi-output functions \( f(x) \) (Section 4.1), it can give us the posterior distribution of the output \( f_+ \) as well as its correlation with the input \( x_+ \) (Section 4.2), we can implement hyperparameter tuning (Section 4.3), we can add inducing input points online (Section 4.4) and we can make predictions \( f_* \) using stochastic trial points \( x_* \) (Section 4.5).

4.1 Multiple outputs

So far we have approximated functions \( f(x) \) with only one output. It is also possible to approximate functions \( f(x) \) with \( d_y > 1 \) outputs. The standard way in which this is done in GP regression algorithms (see for instance Deisenroth and Rasmussen (2011)) is by
Input:
A possibly expanding set of measurements \((x_{m1},y_1),\ldots,(x_{mn},y_n)\) in which both \(x_m\) 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 \(f_u\). Otherwise, initialize \(f_u\) as \(N(m_u,K_{uu})\).

Updating:
\[
\text{while there are unprocessed measurements } (x_{m_n+1},y_{n+1}) \text{ do}
\]
1. Apply (13) to find the posterior distribution of the measurement point \(x_{m_n+1}\) (written as \(x_+\)).
2. Use (15) and (16) to update the distribution of \(f_u\).
3. Optionally, use (17) and (18) to calculate the posterior distribution of the function value \(f(x_+\)) (written as \(f_+\)).
\[
\text{end}
\]

Prediction:
Apply (5) to find the distribution \(f_\ast\) for any set of deterministic trial points. For stochastic trial points, use the expansion from Section 4.5.

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

assuming that, given a deterministic input \(x\), all outputs \(f_1(x),\ldots,f_d(x)\) are independent. With this assumption, it is possible to keep a separate inducing input point distribution \(f^i_u \sim N(\mu^i_u,\Sigma^i_u)\) for each output \(f_i(x)\). Hence, each output is basically treated separately.

When using stochastic input points (as noted by Deisenroth and 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 \(f_1^u,\ldots,f_d^u\) caused by stochastic measurement points \(x_+ \sim N(\hat{x}_+,\Sigma_+)\). If we do, we can continue to treat each function output separately, giving our algorithm a runtime of \(O(n_u^3d_y)\). Because one of the focuses of this paper is to reduce the runtime of GP regression algorithms, we will apply the second option in this paper.

When each output is treated separately, each output also has its own hyperparameters. Naturally, the prior output covariance \(\alpha^2_i\) and the output noise \(\sigma^2_n\) can differ per output \(f_i\), but also the input length scales \(\Lambda_i\) may be chosen differently for each output. In fact, it is even possible to specify a fully separate covariance function \(k_i(x,x')\) 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 (13), the parameter \(\Sigma^i_u(\hat{x}_+)\) would not be a scalar anymore, but become a matrix. And because of our assumption that the outputs
are independent, it would be a diagonal matrix. Similarly, the derivative $d\mu^n_+ / dx_+$ would not be a row vector anymore. Instead, it would turn into the matrix $d\mu^n_+ / dx_+$. With these adjustments, (13) still holds and all other equations can be applied as usual.

### 4.2 The posterior distribution of the measured output

In Section 3.2 we found the posterior distribution for $f_u$. For some applications (like the SISOG algorithm presented in Section 5.2) we also need the posterior distribution of the measured function value $f_+$, even though we do not exactly know to which input it corresponds. We can find this element-wise, using the same methods, through

$$
E[f_+]_i = \mu_{n+1}^{+1}(\hat{x}^{n+1}_+) + \frac{1}{2} \text{tr} \left( \frac{d^2 \mu_{n+1}^{+1}(\hat{x}^{n+1}_+)}{dx^2_+} \Sigma^{n+1}_{++} \right),
$$

$$
V[f_+, f_+]_{i,j} = \Sigma^{n+1}_{++}(\hat{x}^{n+1}_+) + \frac{1}{2} \text{tr} \left( \frac{d \mu_{n+1}^{+1}(\hat{x}^{n+1}_+)}{dx_+} \right)^T \Sigma^{n+1}_{++} \left( \frac{d \mu_{n+1}^{+1}(\hat{x}^{n+1}_+)}{dx_+} \right) + \frac{1}{2} \text{tr} \left( \frac{d^2 \Sigma^{n+1}_{++}(\hat{x}^{n+1}_+)}{dx^2_+} \right) \Sigma^{n+1}_{++}.
$$

(17)

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

With the above expressions, we can now calculate the covariance between each of the entries of $f_+$. However, it may occur that we also need to know the posterior covariance between the function value $f_+$ and the function input $x_+$. Using the same method, we can find that

$$
V[f_+, x_+] = \left( \frac{d \mu_{n+1}^{+1}(\hat{x}^{n+1}_+)}{dx_+} \right) \Sigma^{n+1}_{++}.
$$

(18)

Now we know everything about the posterior distributions of $x_+$ and $f_+$. We can calculate their posterior covariances, after taking into account the data stored in the inducing input point distributions $f_u$. And all the time everything is approximated as Gaussian distributions, making all computations computationally very efficient.

### 4.3 Applying hyperparameter tuning to the algorithm

So far we have assumed that the hyperparameters of the Gaussian process were known a priori. When this is not the case, they need to be tuned first. Naturally, this could be done using expert knowledge of the system, but can it 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 $\Sigma_{++}$. However, this method has a computational complexity of $O(n^3)$, with $n$ still the number of measurements. As such, it can only be used for a small number of measurements and it cannot be used online. NIGP seems not to be applicable to our problem.

However, Chalupka et al. (2013) compare various GP regression algorithms, including methods to tune hyperparameters. One of their main conclusions is that the subset-of-data
(SoD) method – simply using a limited number of measurement points to tune hyperparameters – provides a good trade-off between computational complexity and prediction accuracy. As such, applying NIGP for hyperparameter tuning on a subset of data – say, the first few hundred data points – seems to be a decent choice.

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 the SoD method has given us an estimate of the hyperparameters, it will be an effective choice to use the online FITC algorithm with stochastic measurement 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 \( x_m \) 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.

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 \( f_u \) and \( f_{u+} \) satisfy, identically to (5),

\[
\begin{bmatrix}
  f_u \\
  f_{u+}
\end{bmatrix} \sim \mathcal{N}
\begin{bmatrix}
  \mu_u \\
  \mu_{u+}
\end{bmatrix},
\begin{bmatrix}
  \Sigma_{uu} & \Sigma_{uu+} \\
  \Sigma_{u+u} & \Sigma_{u+u+}
\end{bmatrix}
\]

\[
\begin{bmatrix}
  \Sigma_{uu}^n \\
  \Sigma_{u+u}^n \\
  \Sigma_{uu+}^n \\
  \Sigma_{u+u+}^n
\end{bmatrix} =
\begin{bmatrix}
  m_{u+} + K_{u+u}K_{uu}^{-1}(\mu_u - m_u) \\
  K_{u+u}K_{uu}^{-1}K_{uu+} \\
  K_{u+u}K_{uu}^{-1}K_{uu+} \\
  (K_{uu} - \Sigma_{uu}^n)K_{uu+}^{-1}K_{uu+}
\end{bmatrix}.
\]

(19)

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

Additionally, it is also possible to remove unimportant inducing input points. In this case, its entry can simply be removed from \( f_u \). And since it is possible to both add and delete 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 inducing input points are usually added in the SONIG algorithm is as follows. Whenever we incorporate a new measurement with posterior input distribution \( x_+ \sim \mathcal{N}(\hat{x}_+^{n+1}, \Sigma_{++}^{n+1}) \), we check if \( \hat{x}_+^{n+1} \) is already close to any existing inducing input point. To be more precise, we examine the normalized squared distance

\[
(\hat{x}_+^{n+1} - x_{ui})^T \Lambda^{-1} (\hat{x}_+^{n+1} - x_{ui}) \tag{20}
\]

for each inducing input point \( x_{ui} \). If there is no inducing input point whose normalized squared distance is below a given threshold (often chosen to be roughly 1, but tuned not to get too few or too many points), then it means that there is no inducing input point \( x_{ui} \) close to our new measurement \( \hat{x}_+^{n+1} \). As a result, we add \( \hat{x}_+^{n+1} \) to our set of inducing input points. This guarantees that each measurement is close to at least one inducing input point, which always allows the data from the measurement to be taken into account.
4.5 Predictions for stochastic trial points

Suppose we have a stochastic trial point \( x_* \sim \mathcal{N}(\hat{x}_*, \Sigma_{x*}) \) and we want to calculate the distribution of \( f_* = f(x_*) \). How does that work?

For deterministic trial points \( x_* \) we can use (5). For stochastic trial points, we need to integrate over this. 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 that, for the squared exponential covariance function, 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 \( f_k(x) \) can have its own covariance function \( k_k(\ldots) \), and as such its own hyperparameters \( \alpha_k \) and \( \Lambda_k \). (See Section 4.1.) Keeping this in mind, we now define the vectors \( q^k \) and matrices \( Q^{kl} \) element-wise as

\[
q^k_i = \int_X k_k(x_{ui}, x_*) p(x_*) \, dx_* = \alpha_k^2 \sqrt{\frac{\alpha_k^2}{|\Sigma_{x*}||\Sigma_{x*}^{-1} + \Lambda_k^{-1}|}} \exp \left( -\frac{1}{2} (x_{ui} - \hat{x}_*)^T (\Lambda_k + \Sigma_{x*})^{-1} (x_{ui} - \hat{x}_*) \right), \quad (21)
\]

\[
Q^{kl}_{ij} = \int_X k_k(x_{ui}, x_*) k_l(x_*, x_{uj}) p(x_*) \, dx_* = \alpha_k^2 \alpha_l^2 \sqrt{\frac{\alpha_k^2 \alpha_l^2}{|\Sigma_{x*}||\Sigma_{x*}^{-1} + \Lambda_k^{-1} + \Lambda_l^{-1}|}} \exp \left( -\frac{1}{2} (x_{ui} - x_{uj})^T (\Lambda_k + \Lambda_l)^{-1} (x_{ui} - x_{uj}) \right)
\]

\[
\exp \left( -\frac{1}{2} (\bar{x}^{kl}_{uij} - \hat{x}_*)^T \left( (\Lambda_k^{-1} + \Lambda_l^{-1})^{-1} + \Sigma_{x*} \right)^{-1} (\bar{x}^{kl}_{uij} - \hat{x}_*) \right), \quad (22)
\]

where we have defined

\[
\bar{x}^{kl}_{uij} = (\Lambda_k^{-1} + \Lambda_l^{-1})^{-1} \left( \Lambda_k^{-1} x_{ui} + \Lambda_l^{-1} x_{uj} \right). \quad (23)
\]

With these quantities, we can find that

\[
f_* \sim \mathcal{N}(\mu_*, \Sigma_*),
\]

\[
[\mu_*]_k = (q^k)^T (K_{uu}^{-1}) \mu_u^k,
\]

\[
[\Sigma_*]_{k,k} = \alpha_k^2 - \text{tr} \left( (K_{uu}^{-1})^{-1} (K_{uu}^{-1} - \Sigma_{uu}^{-1}) (K_{uu}^{-1})^{-1} Q^{kk} \right)
\]

\[
+ \left( \mu_u^k \right)^T (K_{uu}^{-1})^{-1} Q^{kk} (K_{uu}^{-1})^{-1} \mu_u^k - [\mu_*]_k^2,
\]

\[
[\Sigma_*]_{k,l} = \left( \mu_u^k \right)^T (K_{uu}^{-1})^{-1} Q^{kl} (K_{uu}^{-1})^{-1} \mu_u^l - [\mu_*]_k [\mu_*]_l, \quad (24)
\]

where the latter expression is for the non-diagonal terms of \( \Sigma_* \) (with \( k \neq l \)). Note that the first line of the above expression is in fact an approximation. In reality the distribution \( f_* \)
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 $f_*$ for stochastic trial points $x_*$. 

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 SONIG algorithm to 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 a variant of the SONIG algorithm to identify a 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 (2016)).

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 test. First, we randomly generate a sample function from a Gaussian process with a squared exponential covariance function (see (2)). This is done on the range $x \in [-5, 5]$, subject to the hyperparameters $\alpha = 1$ and $\Lambda = 1$. Subsequently, we take $n$ measurements on random points from the input range and distort both the input $x_m$ and the output $y$ with zero-mean Gaussian white noise with standard deviation $\sigma_x = 0.4$ and $\sigma_n = 0.1$, respectively. 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 $\sigma_x$.

4. The SONIG algorithm, starting with $\mu_u^0 = m_u$ and $\Sigma_{uu}^0 = K_{uu}$, using the hyperparameters given by (3). We use $n_u = 21$ evenly distributed inducing input points.

5. The same as (4), but now with more measurements (800 instead of 200). Because the SONIG algorithm is computationally a lot more efficient than the NIGP algorithm, the runtime of this is similar to that of (3), being roughly 2-3 seconds 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 measurements) to predict the distribution of the inducing input points, followed by the SONIG algorithm applied to the remainder (700) of the measurements, further updating the inducing input points. The runtime of this approach is again similar to that of (3), being 2-3 seconds.

7. The FITC algorithm, using the hyperparameters of (2). This serves as a reference case.
Table 1: Comparison of various GP regression algorithms, applied to noisy measurements of 200 randomly generated sample functions. For details, see the main text.

<table>
<thead>
<tr>
<th>Algorithm Description</th>
<th>Measurements</th>
<th>MSE</th>
<th>Mean variance</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) GPR with exact hyperparameters and no input noise</td>
<td>200</td>
<td>0.74 \times 10^{-3}</td>
<td>0.83 \times 10^{-3}</td>
<td>0.89</td>
</tr>
<tr>
<td>(2) GPR with tuned hyperparameters</td>
<td>200</td>
<td>24.0 \times 10^{-3}</td>
<td>6.1 \times 10^{-3}</td>
<td>3.9</td>
</tr>
<tr>
<td>(3) NIGP with its own hyperparameter tuning</td>
<td>200</td>
<td>22.5 \times 10^{-4}</td>
<td>4.9 \times 10^{-4}</td>
<td>4.5</td>
</tr>
<tr>
<td>(4) SONIG using the hyperparameters of (3)</td>
<td>200</td>
<td>14.4 \times 10^{-4}</td>
<td>6.9 \times 10^{-4}</td>
<td>2.1</td>
</tr>
<tr>
<td>(5) SONIG using the hyperparameters of (3)</td>
<td>800</td>
<td>10.4 \times 10^{-3}</td>
<td>2.1 \times 10^{-3}</td>
<td>5.0</td>
</tr>
<tr>
<td>(6) NIGP on a subset, followed by SONIG on the rest</td>
<td>100/700</td>
<td>10.3 \times 10^{-3}</td>
<td>2.0 \times 10^{-3}</td>
<td>5.1</td>
</tr>
<tr>
<td>(7) FITC, using the hyperparameters of (2)</td>
<td>800</td>
<td>17.4 \times 10^{-3}</td>
<td>1.6 \times 10^{-3}</td>
<td>10.6</td>
</tr>
</tbody>
</table>

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.

To get fair results, we do this whole process 200 times, each time for a different randomly generated function. In a small part of these cases, for some algorithms, Matlab causes numerical issues. For instance, it sometimes has problems properly inverting even a small covariance matrix. These issues are rare and unpredictable, but they significantly distort the results. To solve this, while keeping things fair, we discard the 10% worst results of each algorithm. The average of the remaining 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 measurements, the SONIG algorithm performs better 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 (15)). The NIGP algorithm does not, and if SONIG also does not, the performance of the two algorithms is comparable.

A second thing that can be noticed is that more measurements provide a higher accuracy. In particular, even the FITC algorithm (which does not take input noise into account) with 800 measurements performs better than the NIGP algorithm with 200 measurements. 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 algorithm still performs better than FITC.

Thirdly, it should be noted that the SONIG algorithms with and without an initial NIGP prediction of $f_u$ (algorithms (5) and (6)) have a similar performance. Hence, in the long run, it does not matter much whether or not we use the NIGP algorithm to make an initial prediction of the distribution of $f_u$.

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 (by roughly a factor 4 to 5).

Next, we will look at some plots. To be precise, we will examine algorithms (3) and (4) closer, but subject to only \( n = 30 \) measurements and \( n_u = 11 \) inducing input points. The predictions of the two algorithms, for a single random sample function, are shown in Figure 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 measurements, 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 measurement points together, the SONIG algorithm only uses data from previous measurement points while incorporating a new measurement 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 measurement 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

We will now show how the SONIG algorithm can be used in learning a model of a nonlinear dynamical system based on measured data. To this end we will make use of a nonlinear
Sparse Online Gaussian Process Training with Input Noise

autoregressive model which has exogenous (ARX) inputs of the following form

\[ y_k = \phi(y_{k-1}, \ldots, y_{k-n_y}, u_{k-1}, \ldots, u_{k-n_u}), \]  \(1\)

where \(\phi(\cdot)\) denotes some nonlinear function of past inputs \(u_{k-1}, \ldots, u_{k-n_u}\) and past outputs \(y_{k-1}, \ldots, y_{k-n_y}\). To fit (25) into the SONIG model, let us define

\[ x_k = (y_{k-1}, \ldots, y_{k-n_y}, u_{k-1}, \ldots, u_{k-n_u}), \]  \(2\)

as the input vector, implying that the output is given by \(y_k = \phi(x_k)\). We can now make use of the SONIG algorithm to approximate \(\phi\). While doing this, we keep track of the covariances between respective outputs \(y_k, y_{k-1}, \ldots\) and inputs \(u_{k-1}, u_{k-2}, \ldots\). When done properly this results in what we call the System Identification using Sparse Online GP (SISOG) algorithm, which is described in Algorithm 2.

We will apply this SISOG algorithm to model the dynamical behavior of a magneto-rheological fluid damper. The measured data for this example was provided by Wang et al. (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 et al. (2016) in their Reduced Rank GP-SSM (RR GP-SSM) algorithm.

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

\[ y_{k+1} = \phi(y_k, u_k, u_{k-1}, u_{k-2}). \]  \(3\)

Hyperparameters were initially tuned by passing a subset of the data to the NIGP algorithm, but were subsequently improved through manual tuning. They were set to

\[
\begin{align*}
\Lambda &= \text{diag}(70^2, 20^2, 10^2, 10^2), \\
\Sigma_{+x} &= \text{diag}(2^2, 0.1^2, 0.1^2, 0.1^2), \\
\alpha^2 &= 70^2, \\
\Sigma_{+f} &= 2^2.
\end{align*}
\]  \(4\)

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.

After all training measurements had been used, the SISOG 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 SISOG algorithm also calculated the variance of each prediction \(y_k\) to take this into account while predicting the next output using the techniques from Section 4.5. The resulting predictions can be seen in Figure 2.
Input:
A set of inputs $u_1, u_2, \ldots$ and outputs $y_1, y_2, \ldots$ of a 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 $y_{k+1}$ do

1. Set up $x_{k+1}$ (shortened to $x_+$) using its definition in (25). Find its prior distribution using known covariances between system outputs $y_k, \ldots, y_{k-(n_y-1)}$ and (if necessary) system inputs $u_k, \ldots, u_{k-(n_u-1)}$. Also find the prior distribution of the function output $y_{k+1}$ (denoted as $f_{k+1}$ or shortened as $f_+$).

2. Apply (13) to find the posterior distribution $\mathcal{N}(\hat{x}_{k+1}^+, \Sigma_{k+1}^+)$ of $x_+$.

Optionally, use this to update the posterior distribution of the system outputs $y_k, \ldots, y_{k-(n_y-1)}$ and system inputs $u_k, \ldots, u_{k-(n_u-1)}$.

3. Optionally, if $\hat{x}_{k+1}^+$ is far removed from any inducing input point, add it to the set of inducing inputs $X_u$ using (19). (Or rearrange the inducing input points in any desired way.)

4. Calculate the posterior distribution of the inducing input vector $f_u$ for each of the outputs of $\phi$ using (15) and (16).

5. Calculate the posterior distribution of $y_{k+1}$ using (17). Additionally, calculate the covariances between $y_{k+1}$ and each of the previous system outputs $y_k, \ldots, y_{k-(n_y-1)}$ and inputs $u_k, \ldots, u_{k-(n_u-1)}$ through (18).

end

Prediction:
For any deterministic set of previous outputs $y_k, \ldots, y_{k-(n_y-1)}$ and inputs $u_k, \ldots, u_{k-(n_u-1)}$, apply (5) to predict the next output $y_{k+1}$. For stochastic parameters, use the expansion from 4.5.

Algorithm 2: The System Identification using Sparse Online GP (SISOOG) algorithm: an application of the SONIG algorithm to identify a non-linear system in an online way.
A comparison of the SISOG algorithm with various other methods is shown in Table 2. This table shows that the SISOG algorithm can clearly outperform other black-box modeling approaches. 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 will degrade slightly, though in most cases it still is competitive with other identification algorithms.

Figure 2: Prediction of the output of the magneto-rheological fluid damper by the SISOG 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. As comparison, also the best non-linear 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 SISOG algorithm, indicating the errors are mostly caused by distortions in the training/evaluation data.

6. Conclusions and recommendations

We can conclude that the presented SONIG algorithm works as intended. Just like the FITC algorithm which it expands on, it is mainly effective when there are more measurements than the NIGP algorithm (or regular GP regression) can handle. The SONIG algorithm can
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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>RMSE</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear OE model (4th order)</td>
<td>27.1</td>
<td>The MathWorks Inc. (2015)</td>
</tr>
<tr>
<td>Hammerstein-Wiener (4th order)</td>
<td>27.0</td>
<td>The MathWorks Inc. (2015)</td>
</tr>
<tr>
<td>NLARX (3rd order, wavelet network)</td>
<td>24.5</td>
<td>The MathWorks Inc. (2015)</td>
</tr>
<tr>
<td>NLARX (3rd order, tree partition)</td>
<td>19.3</td>
<td>The MathWorks Inc. (2015)</td>
</tr>
<tr>
<td>RR GP-SSM</td>
<td>8.17</td>
<td>Svensson et al. (2016)</td>
</tr>
<tr>
<td>SISOG</td>
<td>7.12</td>
<td>This paper</td>
</tr>
</tbody>
</table>

Then include the additional measurements very efficiently – incorporating each measurement in constant runtime – giving a much higher accuracy than the NIGP algorithm could have given. 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 the form of the SISOG algorithm, to a non-linear black-box system identification problem. With the proper choice of hyperparameters and inducing input points, it outperformed existing state-of-the-art non-linear system identification algorithms.

Nevertheless, there are still many improvements which 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 (12) or the assumption that higher order terms of $\Sigma_+$ are negligible.

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

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 again 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. The work was also supported by the Swedish research Council (VR) via the project Probabilistic modeling of dynamical systems (Contract number: 621-
Appendix A. Derivation of the inducing input point covariance

In this appendix we derive update law (16) for the covariance matrix of \( f_n^{n+1} \) subject to a stochastic measurement point \( x_+ \sim \mathcal{N}(\hat{x}_+, \Sigma_{n+1}) \). (We drop the superscript \( n + 1 \) here for ease of notation.) This quantity is formally defined element-wise as

\[
V_{f_u x_+} [f_{u_i}(x_+), f_{u_j}(x_+)] = E_{f_u x_+} [(f_{u_i}(x_+) - E_{f_u x_+} [f_{u_i}(x_+)]) (f_{u_j}(x_+) - E_{f_u x_+} [f_{u_j}(x_+)])].
\]

The key to solving this lies in expanding it to

\[
E_{f_u x_+} \left[ \left( (f_{u_i}(x_+) - E_{f_u} [f_{u_i}(x_+)]) + (E_{f_u} [f_{u_i}(x_+)] - E_{f_u x_+} [f_{u_i}(x_+)]) \right) \left( (f_{u_j}(x_+) - E_{f_u} [f_{u_j}(x_+)]) + (E_{f_u} [f_{u_j}(x_+)] - E_{f_u x_+} [f_{u_j}(x_+)]) \right) \right],
\]

where the dots \((\ldots)\) denote the exact same as the previous term within brackets, but then with subscript \( j \) instead of \( i \). Expanding the brackets will lead to four terms. The two cross-terms of the form

\[
E_{f_u x_+} [(f_{u_i}(x_+) - E_{f_u} [f_{u_i}(x_+)]) (E_{f_u} [f_{u_j}(x_+)] - E_{f_u x_+} [f_{u_j}(x_+)])]
\]

will turn out to be zero. To see why, we first note that the second half of the above expression does not depend on the value of \( f_u \). When we then apply the expectation operator with respect to \( f_u \), the first half will become zero.

That leaves us with two terms, one of them being

\[
E_{f_u x_+} \left[ (f_{u_i}(x_+) - E_{f_u} [f_{u_i}(x_+)]) (f_{u_j}(x_+) - E_{f_u} [f_{u_j}(x_+)]) \right].
\]

If we expand \( f_{u_i} \) (and similarly \( f_{u_j} \)) using (14), we will find

\[
E_{f_u x_+} \left[ \left( (f_{u_i}(\hat{x}_+) - E_{f_u} [f_{u_i}(\hat{x}_+)]) + \left( \frac{df_{u_i}(\hat{x}_+)}{dx_+} - E_{f_u} \left[ \frac{df_{u_i}(\hat{x}_+)}{dx_+} \right] \right) \right) \left( (f_{u_j}(\hat{x}_+) - E_{f_u} [f_{u_j}(\hat{x}_+)]) + \left( \frac{df_{u_j}(\hat{x}_+)}{dx_+} - E_{f_u} \left[ \frac{df_{u_j}(\hat{x}_+)}{dx_+} \right] \right) \right) \right],
\]

The nice part is that each of the terms is either dependent on \( x_+ \) or on \( f_u \), but not on both. Hence, we can split the expectation, and after working out the results, neglecting higher orders of \( \Sigma_{n+1} \), we can reduce this term to

\[
V [f_{u_i}, f_{u_j}] + \text{tr} \left( \begin{bmatrix} \frac{df_{u_i}}{dx_+}, \frac{df_{u_j}}{dx_+} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} f_{u_i}, \frac{d^2 f_{u_i}}{dx_+^2} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} f_{u_j}, \frac{d^2 f_{u_j}}{dx_+^2} \end{bmatrix} \right) \Sigma_{n+1}.
\]

Here we have dropped the function notation \((\hat{x}_+)\) from \( f_{u_i}(\hat{x}_+) \) and the subscript \( f_u x_+ \) from \( V \) because of space constraints, and because it is clear with respect to what we take the function/covariance.

The above expression does leave us with the question of how to find the covariance of the derivatives of a GP with respect to some external variable \( x_+ \). Luckily, it can be shown
in a straightforward way (by directly taking the derivative of the definition of the variance operator) that

\[
\frac{dV}{dx_+} = V \left[ f_{u_i}, \frac{df_{u_j}}{dx_+} \right] + V \left[ f_{u_i}, f_{u_j} \right],
\]

\[
\frac{d^2V}{dx_+^2} = V \left[ f_{u_i}, \frac{d^2f_{u_j}}{dx_+^2} \right] + 2V \left[ \frac{df_{u_i}}{dx_+}, \frac{df_{u_j}}{dx_+} \right] + V \left[ \frac{d^2f_{u_i}}{dx_+^2}, f_{u_j} \right].
\]  

(35)

Next, we will look at the last of the four terms. It is defined as

\[
E_{f_+} \left[ (E_{f_+} [f_{u_i} (x_+)] - E_{f_+} [f_{u_i} (x_+)]) (E_{f_+} [f_{u_j} (x_+)] - E_{f_+} [f_{u_j} (x_+)]) \right],
\]

(36)

and working the above out further will eventually turn it into

\[
\left( \frac{dE_{f_+} [f_{u_i} (\hat{x}_+)]}{dx_+} \right) \Sigma_{++} \left( \frac{dE_{f_+} [f_{u_i} (\hat{x}_+)]}{dx_+} \right)^T.
\]

(37)

If we now merge all our results into one expression, we find that

\[
V \left[ f_{u_i} (x_+), f_{u_j} (x_+) \right] = V \left[ f_{u_i} (\hat{x}_+), f_{u_j} (\hat{x}_+) \right] + \left( \frac{dE_{f_+} [f_{u_i} (\hat{x}_+)]}{dx_+} \right) \Sigma_{++} \left( \frac{dE_{f_+} [f_{u_i} (\hat{x}_+)]}{dx_+} \right)^T
\]

\[
+ \frac{1}{2} \text{tr} \left( \left( \frac{d^2V}{dx_+^2} \left[ f_{u_i} (\hat{x}_+), f_{u_j} (\hat{x}_+) \right] \right) \Sigma_{++} \right),
\]

(38)

which equals (16).

Appendix B. Derivatives of prediction matrices

Equation (15) and (16) contain various derivatives of matrices. Using (7) and (8) we can find them. To do so, we first define the scalar quantity

\[
P = \Sigma_{++}^n + \sigma_n^2 = K_{++} + \sigma_n^2 - K_{+u} K_{uu}^{-1} (K_{uu} - \Sigma_{uu}^n) K_{uu}^{-1} K_{u+}.
\]

(39)
We also assume that \( m(\mathbf{x}) = 0 \) for ease of notation. (If not, this can of course be taken into account.) The derivatives of \( \mu_u^{n+1} \) and \( \Sigma_u^{n+1} \) can now be found element-wise through

\[
\frac{d\mu_u^{n+1}}{dx_{+j}} = \Sigma_u^{n} K_u^{-1} \left( \frac{dK_u + P^{-1}}{dx_{+j}} (y_u - K_u K_u^{-1} \mu_u^n) + K_u + \frac{dP^{-1}}{dx_{+j}} (y_u - K_u K_u^{-1} \mu_u^n) \right),
\]

\[
\frac{d^2 \mu_u^{n+1}}{dx_{+j} dx_{+k}} = \Sigma_u^{n} K_u^{-1} \left( \frac{d^2 K_u + P^{-1}}{dx_{+j} dx_{+k}} (y_u - K_u K_u^{-1} \mu_u^n) + \frac{dK_u + P^{-1}}{dx_{+j} dx_{+k}} (y_u - K_u K_u^{-1} \mu_u^n) \right)
\]

\[
\frac{d\Sigma_u^{n+1}}{dx_{+j}} = -\Sigma_u^{n} K_u^{-1} \left( \frac{dK_u + P^{-1}}{dx_{+j}} (y_u - K_u K_u^{-1} \mu_u^n) + \frac{dP^{-1}}{dx_{+j}} (y_u - K_u K_u^{-1} \mu_u^n) \right) K_u^{-1} \Sigma_u^{n},
\]

\[
\frac{d^2 \Sigma_u^{n+1}}{dx_{+j} dx_{+k}} = -\Sigma_u^{n} K_u^{-1} \left( \frac{d^2 K_u + P^{-1}}{dx_{+j} dx_{+k}} (y_u - K_u K_u^{-1} \mu_u^n) + \frac{dK_u + P^{-1}}{dx_{+j} dx_{+k}} (y_u - K_u K_u^{-1} \mu_u^n) \right) K_u^{-1} \Sigma_u^{n}.
\]

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 (2), we can derive

\[
\frac{dK_u +}{dx_{+}} = \alpha^2 \exp \left( -\frac{1}{2} (\mathbf{x}_{u+} - \mathbf{x}_{+})^T \Lambda^{-1} (\mathbf{x}_{u+} - \mathbf{x}_{+}) \right) (\mathbf{x}_{u+} - \mathbf{x}_{+})^T \Lambda^{-1},
\]

\[
\frac{d^2 K_u +}{dx_{+}^2} = \alpha^2 \exp \left( -\frac{1}{2} (\mathbf{x}_{u+} - \mathbf{x}_{+})^T \Lambda^{-1} (\mathbf{x}_{u+} - \mathbf{x}_{+}) \right) \left( \Lambda^{-1} (\mathbf{x}_{u+} - \mathbf{x}_{+}) (\mathbf{x}_{u+} - \mathbf{x}_{+})^T \Lambda^{-1} - \Lambda^{-1} \right),
\]

\[
\frac{dP^{-1}}{dx_{+}} = -P^{-2} \frac{dP}{dx_{+}} = 2P^{-2} \left( K_u K_u^{-1} (K_u - \Sigma_u^{n}) K_u^{-1} \frac{dK_u +}{dx_{+}} \right),
\]

\[
\frac{d^2 P^{-1}}{dx_{+}^2} = \frac{d}{dx_{+}} \left( -P^{-2} \frac{dP}{dx_{+}} \right) = 2P^{-3} \left( \frac{dP}{dx_{+}} \right)^T \left( \frac{dP}{dx_{+}} \right) - P^{-2} \frac{d^2 P}{dx_{+}^2},
\]

\[
\frac{dP}{dx_{+}} = -2K_u K_u^{-1} (K_u - \Sigma_u^{n}) K_u^{-1} \frac{dK_u +}{dx_{+}} + 2K_u K_u^{-1} (K_u - \Sigma_u^{n}) K_u^{-1} \frac{d^2 K_u +}{dx_{+}^2}.
\]
References


