# Online sparse Gaussian process regression using FITC and PITC approximations ⋆

**Hildo Bijl** * **Jan-Willem van Wingerden** * **Thomas B. Schön** ** **Michel Verhaegen** *

* *Delft Center for Systems and Control, Delft University of Technology, The Netherlands, {h.j.bijl,j.w.vanwingerden,m.verhaegen}@tudelft.nl*
** *Department of Information Technology, Uppsala University, Sweden, thomas.schon@it.uu.se*

Gaussian processes; Non-parametric regression; System identification.

**Abstract:** We provide a method which allows for online updating of sparse Gaussian Process (GP) regression algorithms for any set of inducing inputs. This method is derived both for the Fully Independent Training Conditional (FITC) and the Partially Independent Training Conditional (PITC) approximation, and it allows the inclusion of a new measurement point $\boldsymbol{x_{n+1}}$ in $\mathcal{O}(m^2)$ time, with $m$ denoting the size of the set of inducing inputs. Due to the online nature of the algorithms, it is possible to forget earlier measurement data, which means that also the memory space required is $\mathcal{O}(m^2)$, both for FITC and PITC. We show that this method is able to efficiently apply GP regression to a large data set with accurate results.

## 1. INTRODUCTION

Gaussian Process (GP) regression (see Rasmussen and Williams [2006]) is a regression method which is gaining in popularity. Through statistical methods, it provides an estimate of a function $f(\boldsymbol{x})$ based on a limited set of possibly noisy training points $(\boldsymbol{x}, y)$. GP regression is known for providing relatively accurate estimates with only few training data. In addition, it also provides data about the certainty of these estimates.

The most significant downside of GP regression is its computational complexity. Applying conventional GP regression requires $\mathcal{O}(n^3)$ runtime and $\mathcal{O}(n^2)$ memory space, where $n$ is the number of training points. This prevents GP regression from being applied in its original form to any data set larger than roughly a thousand data points.

The conventional way to work around this problem is to apply sparse GP regression, reducing the runtime to $\mathcal{O}(nm^2)$ and the memory space required to $\mathcal{O}(nm)$, where the user-defined $m$ is the size of the set of the so-called inducing input points. A higher number $m$ means a better accuracy, but also more computational requirements. A good overview of sparse GP regression is provided by Candela and Rasmussen [2005]. They show that sparse GP regression in its essence comes down to using a set of inducing inputs, while making assumptions on the prior distribution of the function values $f(\boldsymbol{x})$. By doing so, various other contributions by Smola and Bartlett [2001], Csató and Opper [2002], Seeger et al. [2003] and Snelson and Ghahramani [2006] are merged into a comprehensive framework. From this, it can be concluded that especially the Partially Independent Training Conditional (PITC) algorithm, being a generalization of the Fully Independent Training Conditional (FITC) algorithm, is a promising sparse GP algorithm. We will introduce both these algorithm later on.

Another downside of GP regression is that incorporating a new data point $\boldsymbol{x_{n+1}}$ into the data set $X$ is relatively inefficient. Traditionally, this would require the recalculation of the inverse of the $n \times n$ kernel matrix $K$, resulting in a runtime of $\mathcal{O}(n^3)$ per added measurement. As a result, several methods of applying sparse GP regression in an online way have been introduced. An early paper on this was written by Csató and Opper [2002], who (also see Candela and Rasmussen [2005]) use a Deterministic Training Conditional (DTC) approximation. Others expanded on this work, like Ranganathan et al. [2011] who update and downdate a Cholesky factorization of the kernel matrix $K$ to find the most efficient Subset of Data (SoD) approximation, and Kou et al. [2013], who use an FITC approximation but constrain themselves to inducing inputs chosen from the set of training inputs. Others use rather different techniques, like Hensman et al. [2013] who use stochastic variational inference. None of the algorithms presented in these papers are generally applicable to any set of inducing inputs, nor do they consider the PITC assumption.

The contribution of this paper is to generalize the sparse online GP regression techniques, allowing online updating of the FITC and PITC algorithms and giving a comprehensive overview. To accomplish this, we first use Section 2 to introduce the (sparse) GP regression algorithms mentioned in Table 1. Section 3 then examines how to apply online updating to these algorithms, with as little computational requirements as possible. In Section 4 we show an example of these new techniques being applied, while Section 5 offers conclusions and recommendations.

⋆ 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: 621-2013-5524).

Table 1. Computational requirements for the various examined algorithms, sorted by number of simplifying assumptions. ($\mathcal{O}$ is not written out explicitly.) The runtime is the total runtime of adding the first $n$ measurement points. $m$ is the number of inducing input points and $n_*$ the number of output points, where we assume that $m < n_* < n$. For the PITC algorithm we assume that the subsets $X_i$ do not grow larger than $m$ data points.

| | Preparation time | | Prediction time | | Memory | |
|---|---|---|---|---|---|---|
| Algorithm | Offline | Online | Offline | Online | Offline | Online |
| GP (2.1) | $n^3$ | $n^3$ | $n^2 n_*$ | $n^2 n_*$ | $n^2$ | $n^2$ |
| Sp. GP (2.2) | $n^3$ | $n^3$ | $mn_*^2$ | $mn_*^2$ | $n^2$ | $n^2$ |
| PITC (2.4) | $nm^2$ | $nm^2$ | $mn_*^2$ | $mn_*^2$ | $nm$ | $m^2$ |
| FITC (2.3) | $nm^2$ | $nm^2$ | $mn_*^2$ | $mn_*^2$ | $nm$ | $m^2$ |

## 2. SPARSE GAUSSIAN PROCESSES

This section provides a brief overview of sparse GP regression. We start with a brief introduction to GP regression, followed by the main assumption underlying its sparse version. Then we examine the FITC and PITC assumptions.

### *2.1 Gaussian processes*

In Gaussian process regression, we aim to approximate a function $f(\boldsymbol{x})$. To do so, we perform measurements $f_i = f(\boldsymbol{x}_i)$ at various training points $\boldsymbol{x_i}$, which we merge into a set $X$. (We ignore measurement noise, but Rasmussen and Williams [2006] show how it can be taken into account.) Using this data, we can predict the value $f(\boldsymbol{x_*})$ at some test point $\boldsymbol{x_*}$ or, more general, we can predict the values of $f(X_*)$ for a set of $n_*$ test points $X_*$.

To accomplish this, we assume a priori that the vectors $f(X)$ and $f(X_*)$ (often shortened to $\boldsymbol{f}$ and $\boldsymbol{f_*}$, respectively) have a known joint Gaussian distribution. That is,

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f_*} \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} m(X) \\ m(X_*) \end{bmatrix}, \begin{bmatrix} k(X,X) & k(X,X_*) \\ k(X_*,X) & k(X_*,X_*) \end{bmatrix} \right) = \mathcal{N}\left( \begin{bmatrix} m(X) \\ m(X_*) \end{bmatrix}, \begin{bmatrix} K_{ff} & K_{f*} \\ K_{*f} & K_{**} \end{bmatrix} \right), \tag{1}$$

where the mean function $m(\boldsymbol{x})$ and the covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$ are user-defined and depend only on a few hyperparameters. We assume these hyperparameters are known. If not, we refer to Chalupka et al. [2013] for a comparison of hyperparameter tuning methods and Huber [2014] for an additional hyperparameter tuning method using sigma points.

If we know $\boldsymbol{f}$, then our prior distribution $p(\boldsymbol{f}, \boldsymbol{f_*})$ tells us that the posterior distribution of $\boldsymbol{f_*}$ is given (see Rasmussen and Williams [2006]) by the normal distribution

$$p(\boldsymbol{f_*}|\boldsymbol{f}) = \frac{p(\boldsymbol{f}, \boldsymbol{f_*})}{p(\boldsymbol{f})} = \mathcal{N}(\boldsymbol{\mu_*}, \Sigma_*), \tag{2}$$

$$\boldsymbol{\mu_*} = m(X_*) + K_{*f} K_{ff}^{-1} (\boldsymbol{f} - m(X)), \tag{3}$$

$$\Sigma_* = K_{**} - K_{*f} K_{ff}^{-1} K_{f*}. \tag{4}$$

This is the main equation behind GP regression in its simplest form.

### *2.2 The inducing input assumption*

The problem with GP regression is that we need to find the inverse of the $n \times n$ matrix $K_{ff}$ at a cost of $\mathcal{O}(n^3)$. If we want to take all the available data into account, then there is no way of getting around this. The trick is therefore to reduce the amount of data, such that we retain most of our accuracy, but significantly reduce our calculation time.

There are two approximating assumptions through which this is generally accomplished. The first such assumption will be referred to as the *inducing input assumption*. The idea, as is also clearly outlined by Candela and Rasmussen [2005], is to use a set of $m$ inducing inputs $X_u$ with corresponding function values $\boldsymbol{f_u}$. We then assume that $\boldsymbol{f}$ and $\boldsymbol{f_*}$ are conditionally independent given $\boldsymbol{f_u}$. That is, we assume that

$$p(\boldsymbol{f}, \boldsymbol{f_*}|\boldsymbol{f_u}) = p(\boldsymbol{f}|\boldsymbol{f_u}) p(\boldsymbol{f_*}|\boldsymbol{f_u}). \tag{5}$$

Here we need to choose the positions of the inducing inputs ourselves. We could for instance set them equal to the test inputs $X_*$, spread them out evenly across our input range, tune them as is done by Snelson and Ghahramani [2006], or cleverly choose them from our training points as proposed by Cao et al. [2013]. How to select the positions of the inducing inputs is a research subject by itself, which we will not go into depth on here.

The inducing input assumption effectively adjusts the prior distribution $p(\boldsymbol{f}, \boldsymbol{f_*}, \boldsymbol{f_u})$. To find the new prior distribution, we can use the relation

$$p(\boldsymbol{f}, \boldsymbol{f_*}, \boldsymbol{f_u}) = p(\boldsymbol{f}|\boldsymbol{f_u}) p(\boldsymbol{f_*}|\boldsymbol{f_u}) p(\boldsymbol{f_u}). \tag{6}$$

All the probabilities in the above expression are Gaussian exponentials. The product of Gaussian exponentials is again a Gaussian exponential. So by inserting the exponentials (using (2)) and subsequently working out the results, we can find that the prior equals

$$p(\boldsymbol{f}, \boldsymbol{f_*}, \boldsymbol{f_u}) = \mathcal{N}\left( \begin{bmatrix} m(X) \\ m(X_*) \\ m(X_u) \end{bmatrix}, \begin{bmatrix} K_{ff} & Q_{f*} & K_{fu} \\ Q_{*f} & K_{**} & K_{*u} \\ K_{uf} & K_{u*} & K_{uu} \end{bmatrix} \right). \tag{7}$$

Here we have borrowed the notation $Q_{ab}$ from Candela and Rasmussen [2005]. That is, we define $Q_{ab}$, for any $a$ and $b$, as

$$Q_{ab} = K_{au} K_{uu}^{-1} K_{ub}. \tag{8}$$

So what we have done through our assumption is basically replace the covariance $K_{f*}$ with a different matrix $Q_{f*} = K_{fu} K_{uu}^{-1} K_{u*}$. This confirms that $\boldsymbol{f}$ and $\boldsymbol{f_*}$ can now only 'communicate' through $\boldsymbol{f_u}$.

Using this new prior, we can directly find the posterior distribution of $\boldsymbol{f_*}$. It equals

$$p(\boldsymbol{f_*}|\boldsymbol{f}) = \mathcal{N}(\boldsymbol{\mu_*}, \Sigma_*), \tag{9}$$

$$\boldsymbol{\mu_*} = m(X_*) + Q_{*f} K_{ff}^{-1} (\boldsymbol{f} - m(X)), \tag{10}$$

$$\Sigma_* = K_{**} - Q_{*f} K_{ff}^{-1} Q_{f*}. \tag{11}$$

We can obtain this distribution directly through $\boldsymbol{f}$ with the above equation. However, another option is to first find the posterior distribution of $\boldsymbol{f_u}$ as

$$p(\boldsymbol{f_u}|\boldsymbol{f}) = \mathcal{N}(\boldsymbol{\mu_u}, \Sigma_u), \tag{12}$$

$$\boldsymbol{\mu_u} = m(X_u) + K_{uf} K_{ff}^{-1} (\boldsymbol{f} - m(X)), \tag{13}$$

$$\Sigma_u = K_{uu} - K_{uf} K_{ff}^{-1} K_{fu}, \tag{14}$$

and then use this distribution to find the posterior distribution of $\boldsymbol{f_*}$. This can be done through marginalization

$$p(\boldsymbol{f_*}|\boldsymbol{f}) = \int p(\boldsymbol{f_*}|\boldsymbol{f_u})p(\boldsymbol{f_u}|\boldsymbol{f})\, d\boldsymbol{f_u}. \tag{15}$$

Because we are only dealing with Gaussian exponentials, the above integral can be solved analytically. We can hence find $\boldsymbol{\mu_*}$ and $\Sigma_*$, expressed in $\boldsymbol{\mu_u}$ and $\Sigma_u$, as

$$\boldsymbol{\mu_*} = m(X_*) + K_{*u}K_{uu}^{-1}(\boldsymbol{\mu_u} - m(X_u)), \tag{16}$$

$$\Sigma_* = K_{**} - K_{*u}K_{uu}^{-1}(K_{uu} - \Sigma_u)K_{uu}^{-1}K_{u*}. \tag{17}$$

Note that, if we insert the values for $\boldsymbol{\mu_u}$ and $\Sigma_u$, then the above expressions immediately reduce to (10) and (11).

### *2.3 The FITC assumption*

To get more computational gains, we need to make a second approximating assumption. We will make the same assumption here as was done by Snelson and Ghahramani [2006]. We assume that, given $\boldsymbol{f_u}$, also all function values $\boldsymbol{f}$ are independent with respect to each other. That is,

$$p(f_i, f_j|\boldsymbol{f_u}) = p(f_i|\boldsymbol{f_u})p(f_j|\boldsymbol{f_u}), \tag{18}$$

for every $i \neq j$. We call this assumption the *Fully Independent Training Conditional (FITC) assumption*, which is a term from Candela and Rasmussen [2005]. This assumption gives us a new effective prior distribution $p(\boldsymbol{f}, \boldsymbol{f_*}, \boldsymbol{f_u})$ which, similarly to (7), equals

$$\mathcal{N}\left(\begin{bmatrix} m(\boldsymbol{x_1}) \\ \vdots \\ m(\boldsymbol{x_n}) \\ m(X_*) \\ m(X_u) \end{bmatrix}, \begin{bmatrix} K_{f_1f_1} & \cdots & Q_{f_1f_n} & Q_{f_1*} & K_{f_1u} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ Q_{f_nf_1} & \cdots & K_{f_nf_n} & Q_{f_n*} & K_{f_nu} \\ Q_{*f_1} & \cdots & Q_{*f_n} & K_{**} & K_{*u} \\ K_{uf_1} & \cdots & K_{uf_n} & K_{u*} & K_{uu} \end{bmatrix}\right). \tag{19}$$

Do note that we only make this assumption for the elements of $\boldsymbol{f}$ and not for those of $\boldsymbol{f_*}$. After all, we do not need to invert $K_{**}$. So we have only replaced $K_{ff}$ by

$$\tilde{K}_{ff} = Q_{ff} + \text{diag}(K_{ff} - Q_{ff}) = Q_{ff} + \Lambda_{ff}, \tag{20}$$

$$\Lambda_{ff} = \text{diag}(K_{ff} - Q_{ff}), \tag{21}$$

while $K_{**}$ remains unchanged. The 'diag($P$)' function here denotes the diagonal matrix whose elements along the diagonal equal the corresponding elements of $P$. All other elements are set to zero.

With these assumptions, the posterior distribution of $\boldsymbol{f_*}$, being $p(\boldsymbol{f_*}|\boldsymbol{f}) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_{\boldsymbol{*}}, \tilde{\Sigma}_*)$, is described through

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{*}} = m(X_*) + Q_{*f}\tilde{K}_{ff}^{-1}(\boldsymbol{f} - m(X)), \tag{22}$$

$$\tilde{\Sigma}_* = K_{**} - Q_{*f}\tilde{K}_{ff}^{-1}Q_{f*}. \tag{23}$$

Similarly, our new distribution for $\boldsymbol{f_u}$ is given by

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{u}} = m(X_u) + K_{uf}\tilde{K}_{ff}^{-1}(\boldsymbol{f} - m(X)), \tag{24}$$

$$\tilde{\Sigma}_u = K_{uu} - K_{uf}\tilde{K}_{ff}^{-1}K_{fu}. \tag{25}$$

The problem is that we still need to invert an $n \times n$ matrix. Luckily, we can rewrite the above expressions through repeated use of the matrix inversion lemma (see for instance the work by Hager [1989] and Higham [2002]). It follows that we can also find $\tilde{\boldsymbol{\mu}}_{\boldsymbol{u}}$ and $\tilde{\Sigma}_u$ through

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{u}} = m(X_u) + \tilde{\Sigma}_u K_{uu}^{-1}K_{uf}\Lambda_{ff}^{-1}(\boldsymbol{f} - m(X)), \tag{26}$$

$$\tilde{\Sigma}_u = K_{uu}(K_{uu} + K_{uf}\Lambda_{ff}^{-1}K_{fu})^{-1}K_{uu}. \tag{27}$$

As a result, we again have

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{*}} = m(X_*) + K_{*u}K_{uu}^{-1}(\tilde{\boldsymbol{\mu}}_{\boldsymbol{u}} - m(X_u)), \tag{28}$$

$$\tilde{\Sigma}_* = K_{**} - K_{*u}K_{uu}^{-1}(K_{uu} - \tilde{\Sigma}_u)K_{uu}^{-1}K_{u*}. \tag{29}$$

In the above expressions, the only $n \times n$ matrix which we need to invert is $\Lambda_{ff}$, which is a diagonal matrix, resulting in a significant reduction in computational complexity; also see Table 1.

The algorithm which we have obtained so far is not new. It was proposed in a different form as 'Sparse Pseudo-input Gaussian Processes' (SPGP) by Snelson and Ghahramani [2006] and rewritten to a form similar to ours, called FITC GP, by Candela and Rasmussen [2005]. However, both these papers stopped with the above expressions, while we continue to look at how they can be implemented in an online fashion.

### *2.4 The PITC assumption*

The PITC algorithm (see e.g. [Candela and Rasmussen, 2005]) is an extension of the FITC algorithm. Instead of making the FITC assumption, we now make the *PITC assumption*. That is, we split the training set $X$ up into subsets $X_1, X_2, \ldots, X_p$, where all sets $X_i$ (with $1 \leq i \leq p$) together cover $X$, but $X_i \cap X_j = \emptyset$ for $i \neq j$. We split the output vector $\boldsymbol{f}$ in the same way into $\boldsymbol{f_1}, \ldots, \boldsymbol{f_p}$. The PITC assumption now is that, given $\boldsymbol{f_u}$, all outputs $\boldsymbol{f_i}$ and $\boldsymbol{f_j}$ are independent with respect to each other, but different outputs within a single subset output vector $\boldsymbol{f_i}$ are not independent with respect to each other.

With this assumption, most of our equations stay the same. In fact, if we write $K_{f_if_j}$ as

$$K_{f_if_j} = k(X_i, X_j), \tag{30}$$

then (19) still holds. Similarly to (20), we replace $K_{ff}$ by

$$\tilde{K}_{ff} = Q_{ff} + \text{blockdiag}\,(K_{ff} - Q_{ff}) = Q_{ff} + \tilde{\Lambda}_{ff}, \tag{31}$$

except that we now use the block-diagonal matrix $\tilde{\Lambda}_{ff}$ instead of the diagonal matrix $\Lambda_{ff}$. This matrix has $p$ matrices $\tilde{\Lambda}_{f_if_i}$ along its diagonal, where

$$\tilde{\Lambda}_{f_if_i} = K_{f_if_i} - Q_{f_if_i} = K_{f_if_i} - K_{f_iu}K_{uu}^{-1}K_{uf_i}. \tag{32}$$

Note that, if every set $X_i$ consists of a single vector $\boldsymbol{x_i}$, then the block-diagonal matrix $\tilde{\Lambda}_{ff}$ reduces back to the diagonal matrix $\Lambda_{ff}$ and PITC reduces back to FITC. On the other hand, if there is only one subset $X_i$ which equals $X$, then PITC reduces back to the algorithm of Section 2.2 with only the inducing input assumption.

The resulting PITC distribution for $\boldsymbol{f_u}$ is similarly to (26) and (27) given by

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{u}} = m(X_u) + \tilde{\Sigma}_u K_{uu}^{-1}K_{uf}\tilde{\Lambda}_{ff}^{-1}(\boldsymbol{f} - m(X)), \tag{33}$$

$$\tilde{\Sigma}_u = K_{uu}(K_{uu} + K_{uf}\tilde{\Lambda}_{ff}^{-1}K_{fu})^{-1}K_{uu}, \tag{34}$$

which is computationally efficient, because the only $n \times n$ matrix to be inverted is the block-diagonal matrix $\tilde{\Lambda}_{ff}$.

## 3. ONLINE UPDATING

The previous section considered already known theorems. This section presents new results. In particular, we will examine the case where we continuously get new measurement data. That is, we get a new sample $(\boldsymbol{x_{n+1}}, f_{n+1})$, causing $X$, $\boldsymbol{f}$ and $K_{ff}$ to grow. In the framework of the previous paragraph, all that we needed for prediction of $\boldsymbol{\mu_*}$ and $\Sigma_*$ were $\boldsymbol{\mu_u}$ and $\Sigma_u$. So that poses the question: how can we update $\boldsymbol{\mu_u}$ and $\Sigma_u$ to incorporate new measurement data in the most efficient way?

### *3.1 Updating for sparse GPs*

If we have only made the main inducing input assumption, we will need to keep track of all data of all our measurements. In particular, we will need to maintain the matrix inverse $K_{ff}^{-1}$. To incorporate our new measurement $(\boldsymbol{x_{n+1}}, f_{n+1})$, we can use the block matrix inversion theorem by Hager [1989], which states that

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B\Delta^{-1}CA^{-1} & -A^{-1}B\Delta^{-1} \\ \Delta^{-1}CA^{-1} & \Delta^{-1} \end{bmatrix}, \tag{35}$$

where $\Delta = (D - CA^{-1}B)$. We can apply this with

$$A = K_{f_{1:n}f_{1:n}} = K_{ff}, \qquad B = K_{f_{1:n}f_{n+1}} = K_{ff_+}, \tag{36}$$

$$C = K_{f_{n+1}f_{1:n}} = K_{f_+f}, \quad D = K_{f_{n+1}f_{n+1}} = K_{f_+f_+}, \tag{37}$$

where the subscript $\boldsymbol{f_{1:n}}$ (or short, $\boldsymbol{f}$) denotes the previously known data points $f_1$ up to $f_n$, while the subscript $f_{n+1}$ (or short, $f_+$) denotes the new data point. (We do not have a short notation for $f_{1:(n+1)}$.) The above expression now directly gives us an update law for $K_{ff}$ of runtime $\mathcal{O}(n^2)$ per update.

Subsequently, we can update $\Sigma_u$. An 'easy' way to do so would be to insert $K_{ff}^{-1}$ directly into (14), but this would result in a runtime of $\mathcal{O}(mn(m+n))$. A more efficient update rule can be derived if we apply (35) to (14) first. This would give us

$$\begin{aligned}\Sigma_u^{1:(n+1)} &= \Sigma_u^{1:n} - (K_{uf_+} - K_{uf}K_{ff}^{-1}K_{ff_+}) \\ &\quad (K_{f_+f_+} - K_{f_+f}K_{ff}^{-1}K_{ff_+})^{-1}(K_{f_+u} - K_{f_+f}K_{ff}^{-1}K_{fu}).\end{aligned} \tag{38}$$

This equation has a runtime of $\mathcal{O}((m+n)^2)$.

Finally, we can update $\boldsymbol{\mu_u}$. If we assume, for ease of notation, that $m(\boldsymbol{x}) = 0$, then we can find that

$$\begin{aligned}\boldsymbol{\mu_u^{1:(n+1)}} &= \boldsymbol{\mu_u^{1:n}} + (K_{uf_+} - K_{uf}K_{ff}^{-1}K_{ff_+}) \\ &\quad (K_{f_+f_+} - K_{f_+f}K_{ff}^{-1}K_{ff_+})^{-1}(f_+ - K_{f_+f}K_{ff}^{-1}\boldsymbol{f}).\end{aligned} \tag{39}$$

Again, this follows from applying (35) to (13). If we do want to incorporate a mean function $m(\boldsymbol{x})$ in the above expression, then this is possible by replacing $\boldsymbol{\mu_u}$ by $(\boldsymbol{\mu_u} - m(X_u))$ and $\boldsymbol{f}$ by $(\boldsymbol{f} - m(X))$.

### *3.2 Updating for FITC*

We will now apply online updating to the FITC algorithm. After $n$ measurements, the (assumed known) matrix $\tilde{\Sigma}_u$ equals (see (27))

$$\tilde{\Sigma}_u^{1:n} = K_{uu}\left(K_{uu} + K_{uf_{1:n}}\Lambda_{f_{1:n}f_{1:n}}^{-1}K_{f_{1:n}u}\right)^{-1}K_{uu}. \tag{40}$$

After adding the new measurement $f_{n+1}$, the matrix $\tilde{\Sigma}_u^{1:(n+1)}$ becomes

$$K_{uu}\left(K_{uu} + K_{uf_{1:(n+1)}}\Lambda_{f_{1:(n+1)}f_{1:(n+1)}}^{-1}K_{f_{1:(n+1)}u}\right)^{-1}K_{uu}. \tag{41}$$

We now again shorten the subscript $\boldsymbol{f_{1:n}}$ to $\boldsymbol{f}$ and $f_{n+1}$ to $f_+$ for ease of notation. If we also use that $\Lambda_{ff}$ is a diagonal matrix, we can expand the above relation for $\tilde{\Sigma}_u^{1:(n+1)}$ to

$$K_{uu}\left(K_{uu}+K_{uf}\Lambda_{ff}^{-1}K_{fu}+K_{uf_+}\Lambda_{f_+f_+}^{-1}K_{f_+u}\right)^{-1}K_{uu}. \tag{42}$$

At this point we introduce a theorem by Miller [1981]. It states that, when $A$ and $A+B$ are nonsingular matrices and $B$ is a rank 1 matrix, then

$$(A+B)^{-1} = A^{-1} - \frac{A^{-1}BA^{-1}}{1 + \mathrm{tr}(A^{-1}B)}. \tag{43}$$

We can use this theorem if we apply

$$\begin{aligned} A &= K_{uu}^{-1}\left(K_{uu} + K_{uf}\Lambda_{ff}^{-1}K_{fu}\right)K_{uu}^{-1} = (\tilde{\Sigma}_u^{1:n})^{-1}, \\ B &= K_{uu}^{-1}K_{uf_+}\Lambda_{f_+f_+}^{-1}K_{f_+u}K_{uu}^{-1} = P_{n+1}, \end{aligned} \tag{44}$$

with $P_{n+1}$ defined as shown above. Note that $P_{n+1}$ is indeed a rank one matrix because $\Lambda_{f_+f_+}$ is a scalar. Application of the theorem results in

$$\tilde{\Sigma}_u^{1:(n+1)} = \tilde{\Sigma}_u^{1:n} - \frac{\tilde{\Sigma}_u^{1:n}P_{n+1}\tilde{\Sigma}_u^{1:n}}{1 + \mathrm{tr}\left(\tilde{\Sigma}_u^{1:n}P_{n+1}\right)}. \tag{45}$$

This expression allows us to efficiently update $\tilde{\Sigma}_u$ upon the arrival of new measurement data.

For $\boldsymbol{\mu_u}$ we can derive an expression in a similar way. If we briefly assume that $m(\boldsymbol{x}) = 0$, for simplicity of notation, we can see from (26) that

$$\begin{aligned}\boldsymbol{\tilde{\mu}_u^{1:(n+1)}} &= \tilde{\Sigma}_u^{1:(n+1)}K_{uu}^{-1}K_{uf}^{1:(n+1)}\left(\Lambda_{ff}^{1:(n+1)}\right)^{-1}\boldsymbol{f}_{1:(n+1)} \\ &= \tilde{\Sigma}_u^{1:(n+1)}(\tilde{\Sigma}_u^{1:n})^{-1}\boldsymbol{\tilde{\mu}_u^{1:n}} \\ &\quad + \tilde{\Sigma}_u^{1:(n+1)}K_{uu}^{-1}K_{uf_+}\Lambda_{f_+f_+}^{-1}f_+.\end{aligned} \tag{46}$$

From this it follows that the update law for $\boldsymbol{\tilde{\mu}_u^n}$ is given by

$$\begin{aligned}\boldsymbol{\tilde{\mu}_u^{1:(n+1)}} &= \left(I - \frac{\tilde{\Sigma}_u^{1:n}P_{n+1}}{1 + \mathrm{tr}\left(\tilde{\Sigma}_u^{1:n}P_{n+1}\right)}\right)\boldsymbol{\tilde{\mu}_u^{1:n}} \\ &\quad + \tilde{\Sigma}_u^{1:(n+1)}K_{uu}^{-1}K_{uf_+}\Lambda_{f_+f_+}^{-1}f_+.\end{aligned} \tag{47}$$

In case we do have a non-zero mean function $m(\boldsymbol{x})$, we can incorporate it by replacing $\boldsymbol{\tilde{\mu}_u}$ by $(\boldsymbol{\tilde{\mu}_u} - m(X_u))$ and $f_+$ by $(f_+ - m(\boldsymbol{x_{n+1}}))$.

The online FITC algorithm has the same computational complexity as the offline FITC algorithm (see Table 1). The online algorithm has as advantage that it does not need to remember previous measurement data. After all, all data is stored and maintained within $\boldsymbol{\tilde{\mu}_u}$ and $\tilde{\Sigma}_u$. As a result, it only requires $\mathcal{O}(m^2)$ memory space, instead of $\mathcal{O}(nm)$ for the offline algorithm.

Once $\boldsymbol{\tilde{\mu}_u}$ and $\tilde{\Sigma}_u$ are known, the posterior distribution of $\boldsymbol{f_*}$ can directly be found through (28) and (29) in $\mathcal{O}(mn_*(m+n_*))$ time, with $n_*$ still the number of test points in $X_*$.

### *3.3 Updating for PITC*

Next, we examine how a similar update scheme can be set up for the PITC algorithm. For this, we consider the case where we add a measurement $n+1$ to the subset $X_p$, which will grow from size $n_p$ to $n_p+1$. Note that, instead of $X_p$, we could have chosen any subset $X_i$ of $X$ with $1 \leq i \leq p$.

Similarly to (40), we know that $\tilde{\Sigma}_u^{1:n}$ for PITC equals

$$\tilde{\Sigma}_u^{1:n} = K_{uu}\left(K_{uu} + \Sigma_{i=1}^{p}K_{uf_i}\tilde{\Lambda}_{f_if_i}^{-1}K_{f_iu}\right)^{-1}K_{uu}, \tag{48}$$

where we sum over the matrices $\tilde{\Lambda}_{f_i f_i}$ corresponding to each subset $X_i$. If we apply the above for $n+1$ measurements instead of $n$, we can rewrite the result to

$$\tilde{\Sigma}_u^{1:(n+1)} = \left( \left( \tilde{\Sigma}_u^{1:n} \right)^{-1} + \tilde{P}_{n+1} \right)^{-1}, \tag{49}$$

where we have defined $\tilde{P}_{n+1}$ as

$$\begin{aligned} \tilde{P}_{n+1} = K_{uu}^{-1} \Big( & K_{u f_{p,1:(n_p+1)}} \tilde{\Lambda}^{-1}_{f_{p,1:(n_p+1)} f_{p,1:(n_p+1)}} K_{f_{p,1:(n_p+1)} u} \\ & - K_{u f_{p,1:n_p}} \tilde{\Lambda}^{-1}_{f_{p,1:n_p} f_{p,1:n_p}} K_{f_{p,1:n_p} u} \Big) K_{uu}^{-1}. \end{aligned} \tag{50}$$

Hence, we see that, when adding a measurement $f_{n+1}$ to $\boldsymbol{f_p}$, we only need data from $X_p$. The other points do not influence the update of $\tilde{\Sigma}_u$.

At this point, you may have lost track of what the notation $\tilde{\Lambda}_{f_{p,1:(n_p+1)} f_{p,1:(n_p+1)}}$ means and what the matrix exactly consists of. The subscript $f_{p,1:(n_p+1)}$ indicates that we consider all measurements $\boldsymbol{f_p}$ from the subset $X_p$, including the new measurement $f_{n+1}$ which we just added. As a result, $\tilde{\Lambda}_{f_{p,1:(n_p+1)} f_{p,1:(n_p+1)}}$ equals

$$\begin{bmatrix} \tilde{\Lambda}_{f_p f_p} & \tilde{\Lambda}_{f_p f_+} \\ \tilde{\Lambda}_{f_+ f_p} & \tilde{\Lambda}_{f_+ f_+} \end{bmatrix} = \begin{bmatrix} K_{f_p f_p} - Q_{f_p f_p} & K_{f_p f_+} - Q_{f_p f_+} \\ K_{f_+ f_p} - Q_{f_+ f_p} & K_{f_+ f_+} - Q_{f_+ f_+} \end{bmatrix}, \tag{51}$$

where we have again shortened the subscript $f_{p,1:n_p}$ to $f_p$, while our new measurement is denoted by $f_+$.

When we look at definition (50), we can detect two problems to solve. The first is that we need to know $\tilde{\Lambda}^{-1}_{f_p f_p}$. An efficient solution is to keep track of $\tilde{\Lambda}^{-1}_{f_p f_p}$ as long as new measurements are added. (If no new measurements are added to a subset $X_i$, we may forget $\tilde{\Lambda}_{f_i f_i}$.) An update law for $\tilde{\Lambda}^{-1}_{f_p f_p}$ can be found directly through (35).

A second problem is that, even when $\tilde{\Lambda}^{-1}_{f_p f_p}$ is known, it still takes $\mathcal{O}(m n_p (m + n_p))$ time to evaluate (50). If we instead apply (35) to rewrite it, we can find that

$$\begin{aligned} \tilde{P}_{n+1} = K_{uu}^{-1} & \left( K_{u f_+} - K_{u f_p} \tilde{\Lambda}^{-1}_{f_p f_p} \tilde{\Lambda}_{f_p f_+} \right) \Big( \tilde{\Lambda}_{f_+ f_+} \\ & - \tilde{\Lambda}_{f_+ f_p} \tilde{\Lambda}^{-1}_{f_p f_p} \tilde{\Lambda}_{f_p f_+} \Big)^{-1} \left( K_{f_+ u} - \tilde{\Lambda}_{f_+ f_p} \tilde{\Lambda}^{-1}_{f_p f_p} K_{f_p u} \right) K_{uu}^{-1}, \end{aligned} \tag{52}$$

which only takes $\mathcal{O}\left((m + n_p)^2\right)$ time to evaluate.

We should note at this point that $\tilde{P}_{n+1}$ is a rank one matrix. Hence, we can again apply the theorem by Miller [1981]. Similarly to (45), this results in

$$\tilde{\Sigma}_u^{1:(n+1)} = \tilde{\Sigma}_u^{1:n} - \frac{\tilde{\Sigma}_u^{1:n} \tilde{P}_{n+1} \tilde{\Sigma}_u^{1:n}}{1 + \operatorname{tr}\left( \tilde{\Sigma}_u^{1:n} \tilde{P}_{n+1} \right)}. \tag{53}$$

Deriving an update law for $\tilde{\boldsymbol{\mu}}_{\boldsymbol{u}}^{\boldsymbol{1:(n+1)}}$ is done similarly to what was done in (46). However, we also need to apply some methods which we used in deriving (39). The resulting expression will be

$$\begin{aligned} \tilde{\boldsymbol{\mu}}_{\boldsymbol{u}}^{\boldsymbol{1:(n+1)}} = & \left( I - \frac{\tilde{\Sigma}_u^{1:n} \tilde{P}_{n+1}}{1 + \operatorname{tr}\left( \tilde{\Sigma}_u^{1:n} \tilde{P}_{n+1} \right)} \right) \tilde{\boldsymbol{\mu}}_{\boldsymbol{u}}^{\boldsymbol{1:n}} \\ & + \tilde{\Sigma}_u^{1:(n+1)} K_{uu}^{-1} (K_{u f_+} - K_{u f_p} \tilde{\Lambda}^{-1}_{f_p f_p} \tilde{\Lambda}_{f_p f_+}) \\ & (\tilde{\Lambda}_{f_+ f_+} - \tilde{\Lambda}_{f_+ f_p} \tilde{\Lambda}^{-1}_{f_p f_p} \tilde{\Lambda}_{f_p f_+})^{-1} (f_+ - \tilde{\Lambda}_{f_+ f_p} \tilde{\Lambda}^{-1}_{f_p f_p} \boldsymbol{f_p}). \end{aligned} \tag{54}$$

The above expressions have a few interesting properties. First of all, if we define all subsets $X_i$ such that they have size $n_i = 1$, then we basically use FITC instead of PITC. As a result, $\tilde{\Lambda}_{f_p f_+}$ and $\tilde{\Lambda}_{f_+ f_p}$ become zero. This turns the PITC matrix $\tilde{P}_{n+1}$ from (52) into the FITC matrix $P_{n+1}$ from (44). All other PITC expressions turn into their respective FITC counterparts as well. On the other hand, if we only have one subset $X_i$, which hence equals $X$, then we basically have not made any second sparse GP assumption. As a result, $\tilde{\Lambda}_{f_p f_p}$ becomes $K_{ff} - Q_{ff}$, and it follows that (53) and (54) turn into (38) and (39), respectively, though proving this would be a lengthy process.

## 4. EXAMPLE

In this section we will show the effectiveness of the derived algorithms at identifying a simple system. For small deterministic systems, few data samples are required to learn the system dynamics and hence regular GP regression is applicable. For very large systems, many data samples are required, but also many inducing input points (a large value of $m$) are needed, which makes sparse GP regression less applicable. As such, the ideal example to show the effectiveness of sparse GP regression is to use a small system involving large uncertainty, such that a large data set is required to learn the system dynamics.

The example system is borrowed from Huber [2014] and is described by

$$x(k+1) = \frac{x(k)}{2} + \frac{25 x(k)}{1 + x(k)^2} \cos(x(k)) + \epsilon, \tag{55}$$

where, to get a large uncertainty, we have set $\epsilon \sim \mathcal{N}(0, 10)$. We start at $x(0) = 0$ and calculate the state $x$ at each subsequent timestep. These state transitions are then used in various different GP regression algorithms to learn the above state transition function. We use both regular GP, FITC and PITC, where the latter two have $m = 31$ inducing input points, uniformly distributed over the interval $[-7.5, 7.5]$. We use the squared exponential covariance function with all hyperparameters set to unity.

The three algorithms are compared by calculating the Mean Squared Error (MSE) with respect to the exact state transition function (without noise) on the interval $[-7.5, 7.5]$. We do this comparison in two ways. First we do an 'equal data' comparison, where we give all three algorithms the same $n = 4\,000$ data points. (This is the maximum number of data points before the regular GP algorithm runs into memory problems.) Naturally, FITC and PITC run faster than regular GP, so to keep things fair, we also do an 'equal runtime' comparison, where we give the regular GP algorithm $n = 4\,000$ data points and subsequently give FITC and PITC the same runtime, letting these online algorithms add data points until their time is up. The results of all these experiments, obtained with Matlab on a simple home PC, are shown in Table 2.

From Table 2 we can see that, given the same amount of data, regular GP regression is more accurate than PITC, which in turn is more accurate than FITC. This corresponds to the amount of simplifying assumptions that are made. However, the differences are small. In fact, due to the randomness involved in the simulations, occasionally regular GP regression performed worse than the other two algorithms.

Table 2. Comparison of the accuracy of the different algorithms for various numbers of data points. The results shown are averaged results of 300 independent simulations. The PITC algorithm used subsets of size $m = 31$. After each $m$ data points, all prior data points were forgotten.

| Algorithm | Runtime | Data points | MSE |
|---|---|---|---|
| GP | 4.32 s | 4 000 | 0.102 |
| PITC | 1.40 s | 4 000 | 0.104 |
| | 4.32 s | 12 017 | 0.035 |
| FITC | 0.43 s | 4 000 | 0.106 |
| | 1.23 s | 12 017 | 0.036 |
| | 4.32 s | 42 668 | 0.011 |

On the flip side, FITC and PITC execute significantly faster than regular GP regression. Because of the online nature of these algorithms, the additional available runtime can be used to incorporate more data points. When this is done, both FITC and PITC perform much better than regular GP regression. In addition, FITC performs better than PITC for the simple reason that it can incorporate more data in the same amount of time, even though it uses that data slightly less efficiently.

## 5. CONCLUSIONS AND RECOMMENDATIONS

In this paper we have derived expressions which allow for an efficient online implementation of the FITC and PITC algorithms. These algorithms are in particular effective when a large set of data needs to be learned, for instance to approximate the system dynamics of a highly stochastic system. Given the same amount of data, the FITC and PITC algorithms have a slightly lower accuracy than regular GP regression. However, because of their faster runtime, they can incorporate much more data in the same time, while using less memory. As a result we have shown that, when computational requirements are the limiting factor, FITC and PITC result in much more accurate estimates than regular GP regression.

Furthermore, due to the online way in which these algorithms have been derived in this paper, it is possible to incorporate additional data as long as there is still available runtime. In addition, incorporating such data does not put any additional requirements on memory, since all data is contained within the parameters $\tilde{\boldsymbol{\mu}}_{\boldsymbol{u}}$ and $\tilde{\Sigma}_u$. This makes the online versions of the FITC and PITC algorithms very suitable for application in systems with large sets of measurement data, especially when new measurement data is constantly being added.

There are still various subjects for future research, to improve these algorithms further. For instance, no fine-tuning has been done of either the hyperparameters or the positions of the inducing inputs. It would be interesting to see if the methods used by Snelson and Ghahramani [2006] can be applied in an online way, without having to start incorporating data from scratch. Expanding on this, also the online addition of inducing input points (increasing $m$), whenever this would result in a better performance, would be an interesting subject for future research.

Another subject for future research is to find an optimal or near-optimal way of forming subsets $X_i$ within the PITC algorithm. It is expected that putting highly correlated measurements into the same subset will increase the algorithm accuracy. After all, then the off-diagonal terms in $K_{f_i f_i}$ are likely to have higher values, which means less data is lost due to the PITC assumption. Currently, we have lumped $m$ subsequent measurements into a single subset, assuming that subsequent measurements are correlated. However, through some sort of clustering algorithm, it may be possible to increase the accuracy of the PITC algorithm even further.

## REFERENCES

Joaquin Q. Candela and Carl E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

Yanshuai Cao, Marcus A. Brubaker, David Fleet, and Aaron Hertzmann. Efficient optimization for sparse Gaussian process regression. In *Advances in Neural Information Processing Systems 26*, pages 1097–1105, 2013.

Krzysztof Chalupka, Christopher K. I. Williams, and Iain Murray. A framework for evaluating approximation methods for Gaussian process regression. *Journal of Machine Learning Research (JMLR)*, 14:333–350, 2013.

Legel Csató and Manfred Opper. Sparse online Gaussian processes. *Neural Computation*, 14(3):641–669, 2002.

William W. Hager. Updating the inverse of a matrix. *SIAM Rev.*, 31(2):221–239, jun 1989.

James Hensman, Fusi Nicoló, and Neil D. Lawrence. Gaussian processes for big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, 2013.

Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002.

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

Peng Kou, Feng Gao, and Xiaohong Guan. Sparse online warped Gaussian process for wind power probabilistic forecasting. *Applied Energy*, 108:410–428, August 2013.

Kenneth S. Miller. On the inverse of the sum of matrices. *Mathematics Magazine*, 54(2):67–72, 1981.

Ananth Ranganathan, Ming-Hsuan Yang, and Jeffrey Ho. Online sparse Gaussian process regression and its applications. *IEEE Transactions on Image Processing*, 20(2): 391–404, February 2011.

Carl E. Rasmussen and Christopher K.I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Matthias Seeger, Christopher K.I. Williams, and Neil D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *In Workshop on AI and Statistics 9*, 2003.

Alex J. Smola and Peter Bartlett. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems 13*, pages 619–625. MIT Press, 2001.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264. MIT Press, 2006.