FIFA-GP:
A Python Package for Fast Gaussian Process Regression

Alessandro Zito, Graham Tierney, Vittorio Orlandi

April 2021

Abstract

Gaussian processes (GP) are a powerful, non-parametric tool used to capture complex dependencies in data. They provide highly-flexible models, straight-forward estimation procedures, and uncertainty quantification not available to many other methods. However, their use has been hindered due to high computational complexity and storage demands. Sampling requires inversion and Cholesky decomposition of the covariance matrix that scale at a rate of $\mathcal{O}(n^3)$, and storage of covariance matrix that scales at a rate of $\mathcal{O}(n^2)$. This paper outlines the **F**ast **I**ncreased **F**idelity **A**pproximate **GP** (FIFA-GP) developed by Drs. Kelly Moran and Matthew Wheeler to sample a GP in $\mathcal{O}(n \log^2 n)$ time with $\mathcal{O}(n \log n)$ storage demands. While other methods have shown bounds on the approximation error to the *prior* on a GP, this is the first approximation method that demonstrates a bound for the Kullback-Leibler divergence between the approximation and the *posterior* distribution that can be made arbitrarily small. We implement the method in C++ and make the code executable in Python using **pybind11**. We then compare the FIFA-GP sampler to several other Python GP estimation procedures. Our package is available on TestPyPi as **FIFA-GP** and installable via `pip`. The code is hosted on Github at https://github.com/vittorioorlandi/STA663_FIFA_GP.

1 Introduction

Gaussian process (GP) regression is a nonparametric method for modeling smooth functions. Specifically, GPs impose a complex dependency between each observation in the data by means of a $n \times n$ kernel matrix, K, which acts as covariance matrix for the process. Such approach has several advantages over traditional supervised machine learning methods, as it provides a highly flexible functional form to capture trends in the data. Moreover, the relative simplicity of the distributions involved in the process allow for easy to derive uncertainty measures. Nevertheless, their application is limited to low dimensional datasets, as they come with an expensive computational burden: sampling from a GP requires the inversion of K along with computing a square-root decomposition. These operations scale at $\mathcal{O}(n^3)$, which makes Gaussian processes inapplicable when n is moderate to large. This is particularly problematic in Bayesian settings, where GPs are commonly estimated via Markov chain Monte Carlo methods. In a Gibbs sampling framework, for example, model estimates are obtained by iteratively drawing one sample at random from the full conditional distribution of each parameter involved until convergence. This severely limits the applicability of Gaussian processes, as each iteration carries a computational cost of $\mathcal{O}(n^3)$. To address this computational bottleneck, many methods have been developed, both frequentist [8, 11, 16] and Bayesian [5, 7, 4]. Most of these methods are based on approximations to the covariance function of the GP. However, the approximations to the covariance may not be accurate, as in some cases the approximation diverges from the truth. Furthermore, even if the approximations are provably close to the truth, the resulting algorithms may not be efficient in practice despite their theoretical efficiency, or only if there is certain structure in the problem, such as the existence of a good low dimensional approximating subspace to the whole space. These methods can also struggle with capturing *both* local and global structure in the covariance matrix. Lastly, in the case of Bayesian methods, no previous approach bounds the distance between the posterior targeted by the approximate GP and the true posterior, the object of interest in most Bayesian analyses.

One recent solution to this problem, the **F**ast **I**ncreased **F**idelity **A**pproximate **GP** (FIFA-GP) has been proposed by [10]. In particular, the authors exploit the Hierarchical Off-Diagonal Low Rank (HODLR)

approximation method for the covariance matrix of the full conditional associated with the mean of a Gaussian process. This severely reduces the computational costs associated with the matrix operations involved from $\mathcal{O}(n^3)$ to $\mathcal{O}(n \log^2 n)$, thus widening the applicability of GPs in large n settings.

In this paper, we both provide a summary of the work in [10], and we describe our novel Python implementation of it, which embeds code written in C++ via the package **pybind11**. First, we carefully review the general setting of a Gaussian process, highlighting the computational bottlenecks in Gibbs estimation. Then, we report the details of the approximation algorithm proposed by [10] and describe our implementation. Finally, we compare the HODLR approximation procedure with standard Python tools for GPs in terms of accuracy and computation time.

2 Gaussian Process Models

A Gaussian process can be thought of as a prior over functions of the form $f : \mathbb{R}^d \to \mathbb{R}$ where each finite sample from f at the points $\{x_i \in \mathbb{R}^d\}_{i=1}^n$ has a joint multivariate normal distribution and d is the dimension of the input space. When the function is observed with noise, the model can be expressed as $y_i = f(\mathbf{x_i}) + \epsilon_i$ where $\epsilon_i \sim N(0, \tau^{-1})$ and $f(\cdot) \sim N(m(\cdot), k(\cdot, \cdot))$ and with $m(\cdot)$ and $k(\cdot, \cdot)$ defined as the mean and covariance functions of the process, respectively. In this expression, k can be any symmetric covariance function. A common choice for it is the so-called square-exponential kernel, defined

$$k(\mathbf{x_i}, \mathbf{x_j}) = \sigma^2 \exp\left[-\frac{1}{2}(\mathbf{x_i} - \mathbf{x_j})^T \Omega (\mathbf{x_i} - \mathbf{x_j})\right],$$

where $\Omega = \text{diag}(1/\rho_1^2, 1/\rho_2^2, \ldots, 1/\rho_d^2)$ is a diagonal matrix of lengthscale parameters $\rho_j > 0$ controlling the relative importance of the j'th input feature and $\sigma^2 > 0$ is the global variance of the kernel. We denote the collection of hyperparameters defining the kernel as $\Theta = \{\sigma^2, \rho_1, \ldots, \rho_d\}$ and define $K_{nn}(\Theta)$ as the covariance matrix associated with a dataset of n observations, with $[K_{nn}(\Theta)]_{ij} = k(\mathbf{x_i}, \mathbf{x_j})$. Notice that $m(\cdot)$ is commonly set equal to zero *a priori*. This ensures that the entire behaviour of the process is a priori captured by K alone. A compact representation of the model is thus

$$\mathbf{y} \mid \mathbf{f}, X, \Theta \sim N(\mathbf{f}, K_{nn}(\Theta) + \tau^{-1} I_n)$$

with $\mathbf{f}$ denoting the realized values of the GP and $\mathbf{y}$ the observed outcome.

The flexibility of this framework is apparent when compared with the standard setup of an ordinary least squares (OLS) regression. In particular, under OLS, one has that

$$\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta} \sim N(\mathbf{X}\boldsymbol{\beta}, \tau^{-1} I_n),$$

where $\mathbf{X}$ is the same design matrix as in a GP regression and $\boldsymbol{\beta}$ is the vector of regression coefficients. In other words, the mean $\mathbf{X}\boldsymbol{\beta}$ is a linear function of the input features, whose observations are assumed independent and identically distributed. Instead, in a Gaussian process regression one can model any smooth function in the mean and capture the dependence between observations through the covariance kernel, which ensures that points which are close in the input space have a higher correlation than points that are far apart.

3 Estimation

In this section we describe a method to estimate the posterior distribution of the parameters, namely $p(\mathbf{f}, \Theta, \tau \mid \mathbf{y}, \mathbf{X})$, in a Bayesian setting. In particular, we focus on the case of a squared exponential kernel with form

$$k(\mathbf{x_i}, \mathbf{x_j}) = \sigma^2 \exp(\rho ||\mathbf{x_i} - \mathbf{x_j}||_2^2).$$

Note that there is a singular lengthscale parameter ρ, giving all features equal importance. To ease computations and exploit the conditional conjugacy property of the normal model, we let

$$\tau \sim Ga(a_1/2, b_1/2), \qquad \sigma^{-2} \sim Ga(a_2/2, b_2/2).$$

be the prior distribution for the likelihood precision and the kernel precision, respectively. Furthermore, we adopt a discrete uniform prior for the lengthscale ρ, with

$$Pr(\rho = s_i) = 1/r, \quad s_i \in \{s_1, \ldots, s_r\}.$$

Such a choice is common in the literature [6, 14, 15]. On the other hand, standard Python implementations of Gaussian processes may assume a continuous prior over the lengthscale. These differences are noted when we compare FIFA-GP to other alternatives.

The advantage of choosing such a prior structure for our model is that the full conditional distributions for all parameters are analytically tractable. Using a similar notation as Section 4.1 in [10], all quantities can be estimated by iteratively sampling from the following distributions:

$$\mathbf{f}|- \sim N(K_{\sigma^2,\rho}(K_{\sigma^2,\rho} + \tau^{-1}I)^{-1}\mathbf{y}, K_{\sigma^2,\rho}(\tau K_{\sigma^2,\rho} + I)^{-1}), \tag{1}$$

$$\tau|- \sim Ga\left(\frac{a_1 + n}{2}, \frac{b_1 + (\mathbf{y} - \mathbf{f})^T(\mathbf{y} - \mathbf{f})}{2}\right), \tag{2}$$

$$\sigma^{-2}|- \sim Ga\left(\frac{a_2 + n}{2}, \frac{b_2 + \sigma^2\mathbf{f}^T K_{\sigma^2,\rho}^{-1}\mathbf{f}}{2}\right), \tag{3}$$

$$Pr(\rho = s_i) \propto \det(K_{\sigma^2,s_i})^{-1/2} \exp\left\{-\frac{1}{2}\mathbf{f}^T K_{\sigma^2,s_i}^{-1}\mathbf{f}\right\}. \tag{4}$$

The notation $K_{\sigma^2,\rho}$ is to make explicit the dependence of the covariance matrix on specific hyperparameter values. Morevoer, note that the conditional distribution for σ^2 has a σ^2 in the parameter values. This is not an error and appears due to notational convenience from $K_{\sigma^2,\rho}$ having a multiplicative factor of σ^2 in its definition. Despite its analytic convenience, it is immediately apparent that such a Gibbs sampler requires several expensive matrix operations. Namely, sampling from equation (1) requires computing the inverse of the $n \times n$ matrix $(\tau K_{\sigma^2,\rho} + I)$ and the Cholesky decomposition of $K_{\sigma^2,\rho}(\tau K_{\sigma^2,\rho} + I)^{-1}$. Moreover, the full conditionals in equation (3) and (4) require the inversion of the kernel and the calculation of its determinant. All these operations are come with a computational cost of $\mathcal{O}(n^3)$, which hinder the use of GPs on large datasets.

4 Matrix Approximation

This section describes the HODLR approximation strategy performed in [10]. The key ideas here are to use specific low-rank approximations to full matrices whose structure makes those operations easier to compute and still captures the advantages of GP regression.

Suppose some $m \times n$ matrix A is closely approximated by a rank p matrix A_p such that $A \approx A_p = UV^T$ where U is $m \times p$ and V is $n \times p$. Storage of the matrix A is $\mathcal{O}(mn)$ while storage of the two matrices U and V is $\mathcal{O}(\max(m, n)p)$. Beyond storage gains, certain low-rank approximations, such as those used in this paper, can improve the speed at which linear algebra operations can be computed.

The broad class of $\mathcal{H}$-matrices relies on approximating a full, non-sparse matrix based on a hierarchical tree-based grouping of rows and columns with low-rank components. Specifically, the FIFA-GP algorithm we describe here relies on approximating the covariance matrix K with Hierarchical Off-Diagonal Low Rank (HODLR) matrices.

A HODLR matrix approximation for a covariance matrix in a GP is constructed as follows. Given a tree that partitions the data, the original matrix is partitioned into a block diagonal structure as given in Figure 1. The diagonal entries, covariance structure among observations in the same leaf of the partition tree, are preserved exactly, while the relationships between leaves are stored with less precision the farther away the nodes are. This structure is equivalently captured by recursively splitting the matrix into a 2×2 block partition, then splitting again the diagonal blocks into a 2×2 partition, and so on. This recursive construction defines the level of the HODLR approximation as the number of recursions. This approximation structure is captured by a factorization given in Figure 2.

This view of the HODLR matrix shows how it is well-suited to approximating a GP kernel. Local structure is preserved exactly in the diagonal blocks, and global structure is still measured with the low-rank

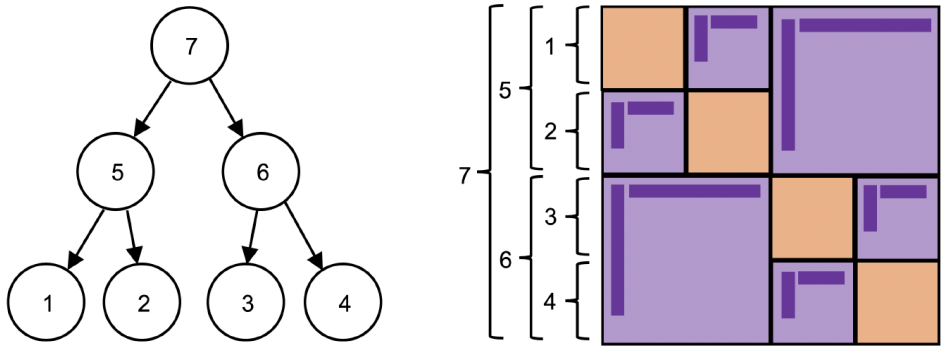

Figure 1: Construction of a HODLR Matrix

Example partitioning tree (left) and corresponding 2-level HODLR matrix (right). Orange blocks are full rank, purple are represented with a low-rank factorization. Figure taken from [10].

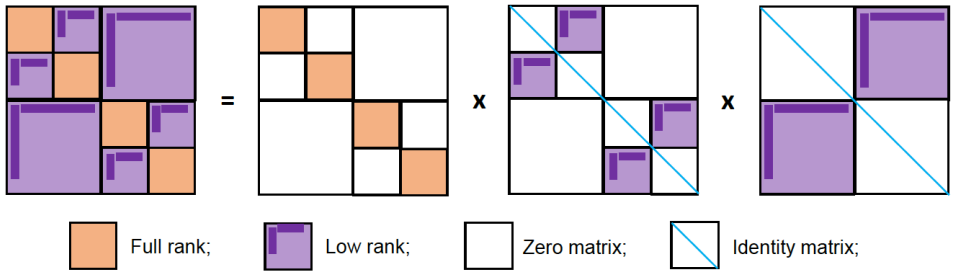

Figure 2: Factoring of a HODLR Matrix

Example factoring of the matrix from Figure 1. Figure taken from [10].

approximations. Other GP methods that focus on local structure, such as nearest neighbor GPs or tapering methods, are unable to capture the global structure, when it exists, with as much accuracy as the FIFA-GP method [10].

Details of the construction of HODLR matrices and operations on them are beyond the scope of [10], and as such we do not document them here. Open-source C++ code for those operations is available from the **HODLRlib** library [3]. The FIFA-GP sampler requires a method to factor symmetric, positive-definite HOLDR matrices, find their determinant, matrix-vector multiplication, and a solver. Those tools are all available from the library; details on the algorithms are found in [1, 2].

5 Implementation

5.1 Algorithms

Using the HODLR matrix construction outlined in the above section, the operations involving covariance matrix K in the Gibbs sampler equations (1)-(4) are replaced with their corresponding HODLR counterparts. Computing the inverse, determinant, and Cholesky decompositions required to sample $\mathbf{f}$ – normally $\mathcal{O}(n^3)$ operations – are replaced with $\mathcal{O}(n \log^2 n)$ counterparts using the **HODLRlib** library. In particular, the discrete prior in (4) allows the inverse and determinant to be computed for each possible value of ρ prior to running the sampler. This notably improves the speed in practice for equations (3) and (4).

The paper notes that the only full conditional requiring additional approximation is $\mathbf{f}|-$. For ease of

notation, let $K = K_{\sigma^2,\rho}$. The posterior covariance $K(K+\tau^{-1})^{-1}$ is not well approximated by $\mathcal{H}$-matrices but the mean $K(K+\tau^{-1})^{-1}\mathbf{y}$ is easily calculated. The sampling is thus computed in the following steps summarizing Algorithm 1 in [10]. The strategy is to approximate K and $M = \tau K + I$ and leverage the fact that $K(\tau K + I)^{-1} = (\tau K + I)^{-1}(\tau KK + K)(\tau K + I)^{-1}$.

Algorithm 1: Sample GP using $\mathcal{H}$-matrices

Result: Produce a sample from $p(\mathbf{f}|\mathbf{y}, X, \Theta)$

Construct: $\tilde{K} \approx K$;	`// H-matrix construction`
Construct: $\tilde{M} \approx \tau K + I$;	`// H-matrix construction`
Construct: W such that $\tilde{K} = WW^T$;	`// H-matrix symmetric factorization`
Sample: $\mathbf{a}, \mathbf{b} \sim N(0, I)$;	
Let: $\mathbf{Z} = \sqrt{\tau}\tilde{K}\mathbf{a} + W\mathbf{b}$;	// $Z \sim N(0, \tau\tilde{K}\tilde{K} + \tilde{K})$
Solve: $\tilde{M}\mathbf{w} = \mathbf{Z}$;	// $w \sim N(0, \tilde{K}[\tau\tilde{K} + I]^{-1})$
Solve: $\tilde{M}\mathbf{r} = \tau\mathbf{y}$;	// $r = (\tau\tilde{K} + I)^{-1}\tau\mathbf{y}$
return $\mathbf{w} + \tilde{K}\mathbf{r}$;	

We implement the full Gibbs sampler, including the above sampler for (1), in C++ with calls to the **HODLRlib** for the relevant computations. The code is wrapped with **pybind11** and imported using **cppimport** to create the final **FIFA-GP** package in Python.

5.2 Installing FIFA-GP

The package **FIFA-GP** can be installed by running the following code in the command line

```
pip install --index-url https://test.pypi.org/simple/ fifa-gp
```

or by downloading the zipped source file from TestPyPi at `https://test.pypi.org/project/fifa-gp/`, navigating to the unzipped folder, and running the command

```
python setup.py install
```

The package, along with code to reproduce the examples and comparisons in the report, are also hosted on Github in the repository `vittorioorlandi/STA663_FIFA_GP`.

5.3 The FIFA_GP Class

We strove to make the **FIFA-GP** package similar to Python industry standards for dealing with Gaussian Processes, such as the Bayesian **GPy** package and the frequentist tools in the **sklearn.gaussian_process** module [9, 12]. To do so, the package defines a class for objects of type `FIFA_GP`. Upon instantiation, the relevant C++ functions for sampling are imported via **cppimport** and thereby made accessible to `FIFA_GP` methods. In line with other packages, **FIFA-GP** features a `fit` method for fitting the Gaussian Process, `predict_` methods for performing prediction on relevant quantities of interest, and `get_` methods for retrieving the fitted parameters of the GP. A complete list of the `FIFA_GP` attributes and methods can be found in Tables 1 and 2.

The `fit` method estimates parameters from a 1-dimensional GP with a squared exponential kernel employing the priors from [10]. Extending the sampler to deal with higher dimensional inputs requires a tensor-product approximation that is substantially more complicated and is the object of future work. Additional details can be found in [10]. Reasonable defaults are provided for both the priors and the HODLR approximations. A basic illustration of usage of the package is shown in Listing 1.

Table 1: Attributes of the `FIFA_GP` class

Attribute	Description
X	Observed covariates
y	Observed responses
regression	Whether `FIFA_GP.y` is taken to be noisy data
tol	The tolerance for the HODLR approximation
M	The maximum sub-matrix size in the HODLR approximation
nsamps	Number of samples
tau	Posterior draws of noise precision τ
sig_f	Posterior draws of function variance σ_f^2
rho	Posterior draws of kernel lengthscale ρ
f	Posterior draws of GP function $\mathbf{f}$

Table 2: Methods of the `FIFA_GP` class

Method	Description
fit	Fits a Gaussian Process
get_params	Retrieve posterior samples of the GP parameters
get_params_mean	Retrieve the posterior means of the GP parameters
predict_f	Posterior predictive for $\mathbf{f}$ at new observations
predict_f_mean	Mean of posterior predictive for $\mathbf{f}$ at new observations
predict_y	Posterior predictive for y at new observations
predict_y_mean	Mean of posterior predictive for y at new observations

```
# Above: create / import covariates X, Xtest and response Y

# Import the FIFA_GP class
from fifa_gp.regression import FIFA_GP

# Set up an object of type FIFA_GP
gp = FIFA_GP()

# Fit FIFA-GP to the data, using default prior and MCMC parameters
gp.fit(X, Y)

# Predict f for observations in Xtest using the posterior predictive mean
gp.predict_f_mean(Xtest)

# Predict outcomes for units in Xtest using the posterior predictive mean
gp.predict_y_mean(Xtest)

# Posterior means of the GP parameters
gp.get_params_mean()
```

Listing 1: Illustration of the core functionality of **FIFA-GP**

6 Results

6.1 Time Comparison

In this section, we test the time performance of our package when applied to randomly generated datasets of varying sizes. As our method is Bayesian, our aim is to quantify the time improvement ensured by the HODLR approximation against the other Bayesian implementations of a Gaussian process available in Python. In particular, we test the **FIFA-GP** package against i) a pure Python exact Gibbs sampler based off equations (1)-(4), ii) the hybrid Monte Carlo sampler in the package **GPy**, and iii) a similar version of the same sampler coded in **PyStan** [13]. While in **Gibbs**, it is easy to replicate the same prior structure we

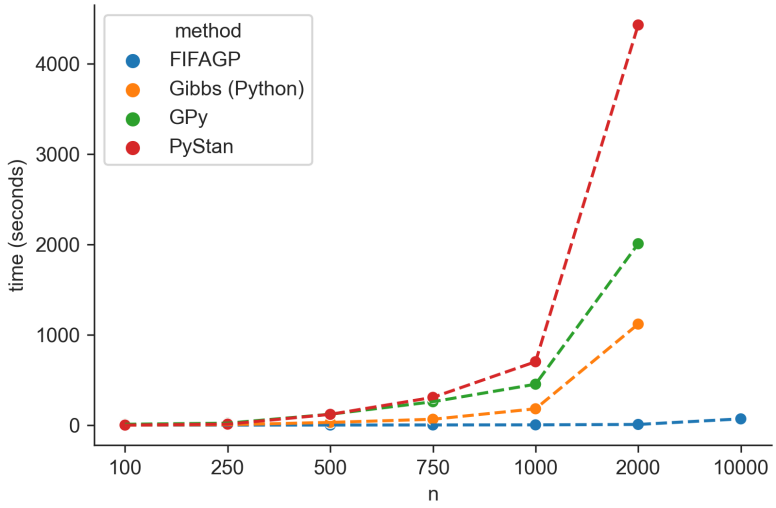

Figure 3: Elapsed time in seconds to draw 100 samples from the posterior for varying sample sizes n.

assume in our package, neither **GPy** nor **PyStan** allows for a discrete prior over the lengthscale parameter ρ. This leads to a different posterior space for the parameters in each setting. However, our main interest here is to to compare the efficiency of each sampler under a default specification. We also note that the bottleneck of the sampling code is not the sampling of ρ. Lastly, as we show below, the structure of the prior is not influential in determining the posterior mean of the process.

Following the same procedure as in Section 5.2 of [10], we simulate data from the following model:

$$y_i = f(x_i) + \epsilon_i, \quad f(x_i) = \sin(2x_i) + \frac{1}{8}e^{x_i}$$

and where $x_i \overset{iid}{\sim} N_{[-2,2]}(0,1)$, i.e. a standard normal truncated to $[-2,2]$, $\epsilon_i \overset{iid}{\sim} N(0,1)$ for $i = 1,\dots,n$. We repeat this procedure for $n \in \{100, 250, 500, 750, 1000, 2000, 10000\}$ and run each method on each simulated dataset. We set the same prior structure for both **FIFA-GP** and **Gibbs**, assuming a uniform discrete prior over 50 equally spaced values between 0.5 and 3 for the lengthscale ρ, and we set the parameters a_1, b_1, a_2, and b_2 all equal to 1. In **GPy**, we adopt the same priors as in the default implementation illustrated in the online tutorial: a $Ga(1/2, 1/2)$ prior over the kernel variance, a $Ga(1/2, 1/2)$ over the likelihood variance and a $Ga(5, 5)$ over the lengthscale. Finally, in **PyStan** we set an $InvGa(5, 5)$ over the lengthscale, an $InvGa(1/2, 1/2)$ over the kernel variance and a $Ga(1/2, 1/2)$ over the likelihood precision. All the differences in these prior structures are due to the different ways in which the exponential kernel is constructed in each package.

Figure 3 plots the elapsed time in seconds required by each sampler to draw 100 samples from the posterior distribution. The advantages of **FIFA-GP** against its alternatives are minor when n is small, but become highly significant when n is moderate to large, at which points other samplers become infeasible. Noticeably, **Gibbs**, **GPy** and **PyStan** failed to yield a result for $n = 10000$, whereas **FIFA-GP** took scarcely over a minute. The exact values of each computational times are reported in Table 3 in the appendix. Finally, Figure 4 shows the average of the posterior samples of $\mathbf{f}$ in each sampler and for each sampled dataset. With the exception of **Gibbs** in the $n = 250$ setting, each method strongly agrees with the others in grasping the trajectory of the data. That **FIFA-GP** is able to perform drastically more efficient sampling without compromising accuracy is its primary strength. To reproduce the results in Figure 3 and 4, follow the tutorial on the jupyter notebook `Gaussian_process_regression.ipynb` in the Github repository.

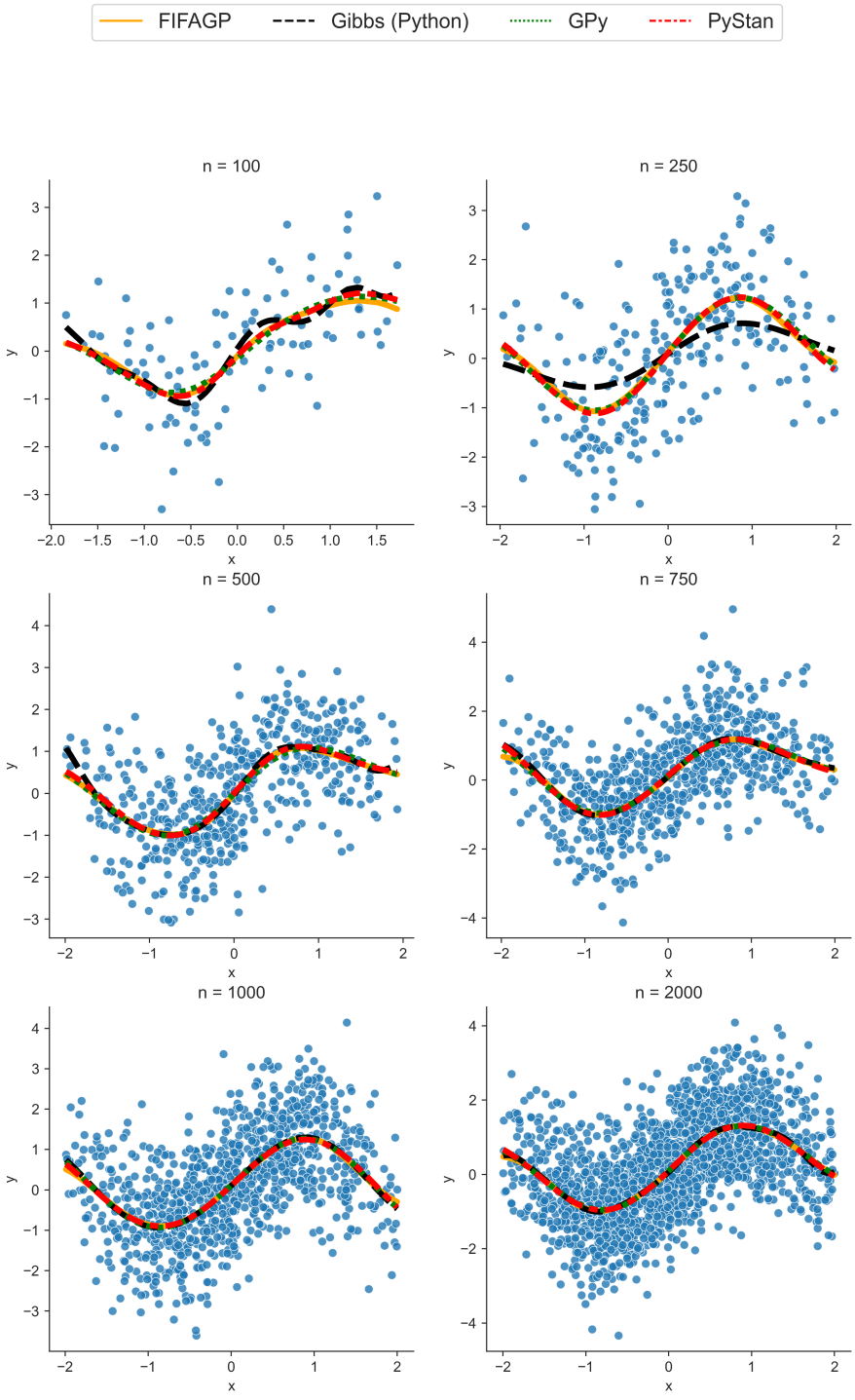

Figure 4: Posterior fit of each method on the simulated datasets.

```
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.507    0.507  128.767  128.767 <ipython-input-12-f0379a032f44>:75(Gibbs_GP)
   100    0.446    0.004   89.922    0.899 <ipython-input-12-f0379a032f44>:71(Gibbs_samplef)
   100    4.863    0.049   25.812    0.258 <ipython-input-12-f0379a032f44>:6(Gibbs_get_Kinvs)
   100    0.061    0.001   12.416    0.124 <ipython-input-12-f0379a032f44>:37(Gibbs_sampleRho_MH)
   184    4.553    0.025    4.901    0.027 <ipython-input-12-f0379a032f44>:1(Gibbs_build_K)
   100    0.067    0.001    0.071    0.001 <ipython-input-12-f0379a032f44>:29(Gibbs_sample_sigma2f)
   100    0.005    0.000    0.014    0.000 <ipython-input-12-f0379a032f44>:22(Gibbs_sample_tau)
     1    0.000    0.000    0.000    0.000 <ipython-input-12-f0379a032f44>:14(Gibbs_Proposalmatrix)
```

Figure 5: A snapshot of the profiler output for a single call to the **Gibbs** GP sampler written in Python.

6.2 Profiling

To further assess the performance of **FIFA-GP** and **Gibbs**, we profile both samplers in order to identify bottlenecks and compare their efficiencies. Profiling for **Gibbs** in Python was done using the **pstats** module, while the MacOS software *Instruments* was used to profile the C++ code called by **FIFA-GP**. Snapshots of the profiler outputs for both samplers are shown in Figures 5 and 6. We first consider the profiling of the Python code by looking at the `cumtime` column, which represents the cumulative time spent in the relevant function and sub-functions.[1] Most notably, we notice the following facts. First, about 70% of the total time is spent sampling from the multivariate normal required to sample $\mathbf{f}$ (`Gibbs_samplef`, taking 89 seconds out of 129 total). Second, sampling the lengthscale ρ via the function `Gibbs_sampleRho_MH` takes about 10% of the total sampling time. Third, the function `Gibbs_get_Kinvs` that constructs the kernel and performs the associated matrix multiplications and inversions takes 0.258 seconds per call. Finally, the sampling times for the other kernel parameters, ie. `Gibbs_sample_sigma2f` and `Gibbs_sample_tau`, are negligible. To contrast, note first from Figure 6 that using HODLR aproximations it only requires 7% of the total time to actually sample from the full conditional of $\mathbf{f}$, the function `samplef_HODLR`. Part of the discrepancy between the implementations is due to the fact that sampling from the multivariate normals required to update $\mathbf{f}$ is much more expensive in Python than it is in C++; implementing this portion of **Gibbs** in C++ would greatly increase its efficiency, even without using HODLR approximations. This slightly muddles the comparison of the two approaches. However, we note that even with this dramatic computational cost, **Gibbs** is still faster than **GPy** and **PyStan**. In **FIFA-GP**, roughly a third of the time is spent constructing and factorizing the HODLR approximation $\tilde{K}$, with the calls to the function function `setup_HODLR_rho` taking 31.7% of total time. However, as the possible values for ρ are fixed a priori, this procedure can be performed only once and does lead to not additional costs when performing further sampling with the same grid of lengthscale parameters. Furthermore, while performing the matrix multiplications and inversions takes 0.258 seconds *per call* in **Gibbs**, doing so takes mere milliseconds *in total* for **FIFA-GP**, even accounting for constructing the HODLR approximations. It is here that the advantage of **FIFA-GP** is most evident. Lastly, as in **Gibbs**, time spent sampling the other kernel parameters, namely `sample_rho_mh`, `sample_prec_f`, and `sample_tau`, is negligible. Thus, since around half the time is spent constructing the HODLR matrix approximations for sampling $\mathbf{f}$, with calls to `HODLR_Tree::factorizeSPD` and `HODLR_Tree::assembleTree` taking 55.9% of total time, the sampling of $\mathbf{f}$ is still by far the bottleneck of the algorithm, even though the HODLR matrix approximations greatly accelerate sampling overall.

7 Conclusion

In this report, we outlined the role of Gaussian processes, an extremely flexible and powerful non-parametric tool, and the computational difficulties of using them in practice, particularly for large data. The FIFA-GP algorithm greatly reduces the complexity of each iteration in a Gibbs sampler, going from $\mathcal{O}(n^3)$ to $\mathcal{O}(n \log^2 n)$ to perform the required kernel matrix operations. These gains are indeed large in practice, as shown in our simulations. In the paper introducing this method, the authors also derive a bound on the approximation to the posterior, which has not been done for other GP approximation methods. Future work will focus on adapting FIFA-GP for classification, supporting multidimensional covariate data, and allowing

[1] There are no issues caused by overlapping sub-function calls in any of the following discussion.

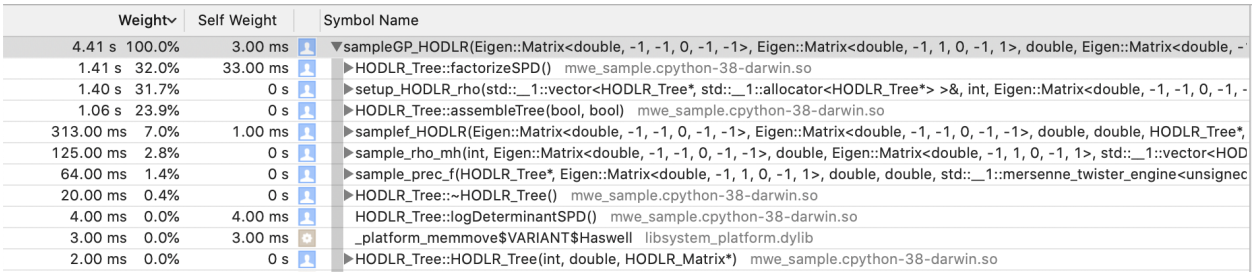

Figure 6: A snapshot of the profiler output for a single call to the **FIFA-GP** sampler written in C++.

for heteroscedastic noise. Overall, we hope that our implementation in Python will favor the use of Gaussian processes under large n scenarios.

References

[1] Sivaram Ambikasaran and Eric Darve. An O(N log N) fast direct solver for partial hierarchically semi-separable matrices. *Journal of Scientific Computing*, 57(3):477–501, 2013.

[2] Sivaram Ambikasaran, Michael O'Neil, and Karan Raj Singh. Fast symmetric factorization of hierarchical matrices with applications, 2016.

[3] Sivaram Ambikasaran, Karan Singh, and Shyam Sankaran. Hodlrlib: A library for hierarchical matrices. *Journal of Open Source Software*, 4:1167, 02 2019.

[4] Anjishnu Banerjee, David Dunson, and Surya T Tokdar. Efficient gaussian process regression for large datasets. *Biometrika*, 100:75–89, 2012.

[5] Abhirup Datta, Sudipto Banerjee, Andrew O Finley, and Alan E Gelfand. Hierarchical nearest-neighbor gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, 111:800–812, 2016.

[6] Abhirup Datta, Sudipto Banerjee, Andrew O. Finley, and Alan E. Gelfand. Hierarchical nearest-neighbor gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, 111(514):800–812, 2016. PMID: 29720777.

[7] Jianhua Z Huan and Huiyan Sang. A full scale approximation of covariance functions for large spatial data sets. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74:111–132, 2012.

[8] Neil Lawrence, Matthias Seeger, and Ralf Herbrich. Fast sparse gaussian process methods: The informative vector machine. In *Proceedings of the 16th annual conference on neural information processing systems*, pages 609–616, 2003.

[9] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017.

[10] Kelly R. Moran and Matthew W. Wheeler. Fast increased fidelity approximate gibbs samplers for bayesian gaussian process regression, 2020.

[11] Duy Nguyen-Tuong, Jan Peters, and Matthias Seeger. Local gaussian process regression for real time online model learning and control. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, pages 1193–1200, 2008.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Allen Riddell, Ari Hartikainen, and Matthew Carter. pystan (3.0.0). PyPI, March 2021.

[14] Huiyan Sang and Jianhua Z. Huang. A full scale approximation of covariance functions for large spatial data sets. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(1):111–132, 2012.

[15] Matthew W. Wheeler. Bayesian additive adaptive basis tensor product models for modeling high dimensional surfaces: an application to high-throughput toxicity testing. *Biometrics*, 75(1):193–201, 2019.

[16] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Proceedings of the 14th annual conference on neural information processing systems*, pages 682–688, 2001.

A Simulation results

Table 3: Time performance (seconds) of each sampler to complete 100 iteration on one simulated dataset of size n

n	FIFAGP	Gibbs (Python)	GPy	PyStan
100	0.59	1.54	10.19	0.77
250	0.60	5.14	22.20	11.35
500	1.51	30.57	121.12	119.70
750	2.47	65.55	259.19	307.64
1000	3.51	181.31	453.12	701.82
2000	8.09	1117.97	2007.18	4426.82
10000	69.79	-	-	-

B Authors contributions

Vittorio Orlandi: Created class structure + documentation for FIFA-GP, refactored code, profiled C++ code, wrote C++ code and helped debug
Graham Tierney: Packaged and wrapped code, dealt with all sorts of nasty import and dependency errors, wrote C++ code and helped debug
Alessandro Zito: Coded base Python GP sampler, did empirical timing analysis / simulations, profiled Python code, wrote C++ code and helped debug