ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Capstone Project

Sparse Gaussian Process Regression

By **Ena Dewan, Gregor Ochsner**

Supervised by **Victor Cohen**

June 26, 2023

Contents

1 Introduction **3**

2 Methods **4**

2.1 Exact GP regression 4

2.2 Sparse approximations for GP regression 5

2.2.1 Inducing point GP 5

2.2.2 Sparse spectrum GP 6

2.3 Gradient boosting regression 8

2.4 Datasets 8

2.5 Experiments 8

2.5.1 Experiment 1: Kernel function comparison 8

2.5.2 Experiment 2: Effect of sparse approximation size . . . 9

2.5.3 Experiment 3: Accuracy and prediction time 10

3 Results **13**

3.1 Experiments 13

3.1.1 Experiment 1: Kernel function comparison 13

3.1.2 Experiment 2: Effect of sparse approximation size . . . 14

3.1.3 Experiment 3: Accuracy and prediction time 14

4 Discussion **19**

5 References **20**

Abstract

Flow field measurements in wind tunnels are very time consuming and expensive and, therefore, it is desirable to speed up the measurement process. Gaussian Process (GP) regression estimates both the mean and variance of a function from the training data. By using the variance information to move a measurement probe in the flow field, one could potentially speed up the measurement process substantially. To apply GP regression to this problem several problems need to be solved: (1) The training data and therefore the regression model need to be updated when a new measurement has been taken, (2) the regression model needs to be real-time capable also with a high number of training data, (3) the model needs to be able to handle a heteroscedastic flow field, where the variance of the measurement data is not constant over the space, (4) a strategy to move the probe in an optimal way needs to be found.
In this project, we focus on problem (2), i.e., we investigate how GP regression can be made fast with a large set of training data. For this purpose, we implement an exact GP and two sparse GP approximations in Python and conduct several experiments. The experiments show that the sparse GP approximations are much faster than the exact GP as soon as the number of training data increases above a few hundred, while achieving a similar accuracy. The results also show that not only the theoretical speed-up is relevant, but the implementation needs to be optimized as well: The models based on the GPyTorch library are substantially faster than the model the authors implemented themselves. From the experiments, it is not obvious which sparse approximation model is best suited for the given problem. To answer this question, one will have to investigate which of the models can be best adapted to problem (1) stated above.

1 Introduction

The SMARTAIR project [1] aims at optimizing the measurement process in a wind tunnel using machine learning techniques. Such measurements of the flow field (e.g. velocity or pressure) in the 3D space are very time consuming and expensive, because - nowadays - they are done by moving a probe along a predefined grid. The goal of the SMARTAIR project is to develop an algorithm that can steer the probe in a smarter way, depending on the measurement data that has already been acquired.

In this student project, we investigate several regression techniques that may be suitable for the SMARTAIR project. We mainly investigate Gaussian process (GP) regression models, because they predict both the mean and the variance of the predicted function, of which the variance could be used to steer the probe in a smart way. In the student project however, we don't study the real-time implementation of the models nor the steering of the probe. We investigate the accuracy and time requirements of several regression models and evaluate them on artificial flow field data in 1D, 2D, and 3D, to evaluate if they are suitable for the described task. Because of the high number of data points, the main focus is on sparse GP approximation models.

2 Methods

Four regression models are used for the experiments in this project. Section 2.1 describes the original GP model, which is referred to as exact GP (**EGP**) in this text. Section 2.2 describes two sparse GP approximations, one based on artificial inducing points and another one based on an approximation of the kernel function with a set of trigonometric basis functions. The former is referred to as inducing point GP (**IPGP**) in this text and is described in Section 2.2.1. The latter is referred to as sparse spectrum GP (**SSGP**) and described in Section 2.2.2. Finally, Section 2.3 describes gradient boosting regression (**GBOOST**) as a non-GP reference model.
Sections 2.4 and 2.5 describe the used datasets and the conducted experiments, respectively.

2.1 Exact GP regression

Formally, a GP is a (potentially infinite) collection of random variables such that the joint distribution of every finite subset of random variables is a multivariate Gaussian. A GP is the extension of the Gaussian distribution to functions. A GP for the function $f : \mathcal{X} \to \mathbb{R}$ is completely defined by a mean function $\mu(x)$ and a covariance (or kernel) function $k(x, x')$

$$f(x) \sim \mathcal{GP}\big(\mu(x), k(x, x')\big).$$

The kernel function $k(x, x')$ determines the smoothness, periodicity, and other properties of the function.
When only looking at n points of the function, $\tilde{x}$, then the function values at these points $\tilde{f} \coloneqq f(\tilde{x})$ are distributed by a multivariate Gaussian

$$\tilde{f} \sim \mathcal{N}(\tilde{\mu}, \tilde{\Sigma}),$$

where $\tilde{\mu} \in \mathbb{R}^{n\times 1}$ is given by $\mu(\tilde{x})$ and $\tilde{\Sigma} \in \mathbb{R}^{n\times n}$ is given by $\big[\tilde{\Sigma}\big]_{i,j} = k(\tilde{x}_i, \tilde{x}_j)$.

EGP regression [12] is one of the applications of GPs. EGP regression is the combination of a GP and conditioning. Given are n observed inputs $X = \big[x_1, \ldots, x_n\big] \in \mathbb{R}^{d\times n}$ and the corresponding outputs $Y = \big[y_1, \ldots, y_n\big]^T \in \mathbb{R}^{n\times 1}$, where the observations are corrupted by additive noise $y_i = f(x_i) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. Given are also k test points $X^* = \big[x_1^*, \ldots, x_k^*\big] \in \mathbb{R}^{d\times k}$ for which we are interested in the function values $f^* \coloneqq f(X^*)$. The joint distribution of training and test data is then given by:

$$\begin{bmatrix} Y \\ f^* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(X) \\ \mu(X^*) \end{bmatrix}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X^*) \\ K(X^*,X) & K(X^*,X^*) \end{bmatrix} \right),$$

where $K(X, X) \in \mathbb{R}^{n \times n}$, $K(X, X^*) = K(X^*, X)^T \in \mathbb{R}^{n \times k}$, and $K(X^*, X^*) \in \mathbb{R}^{k \times k}$ are covariance matrices, and $\mu(X) \in \mathbb{R}^{n \times 1}$ and $\mu(X^*) \in \mathbb{R}^{k \times 1}$ are mean vectors.

We compute posterior distribution by conditioning on the training data, obtaining

$$f^*|Y, X, X^* \sim \mathcal{N}(\mu^*, \Sigma^*),$$

where

$$\mu^* = \mu(X^*) + K(X^*, X)\big[K(X, X) + \sigma_n^2 I\big]^{-1}\big(Y - \mu(X)\big)$$

and

$$\Sigma^* = K(X^*, X^*) - K(X^*, X)^T\big[K(X, X) + \sigma^2 I\big]^{-1}K(X, X^*).$$

This posterior distribution is the regression result with the predictive mean μ^* and predictive covariance matrix Σ^* at test points X^*.

The task of the machine learning engineer when applying EGP regression is the selection of the mean function $\mu(x)$, the kernel function $k(x, x')$ with its parameters as well as finding an apropriate value for data noise parameter σ_n^2. In this project, the mean function is always set to zero. The kernel function is selected based on the experiment described in Section 2.5.1. All parameters of the kernel function and the noise parameter are learned by maximizing the marginal likelihood [12]

$$\log p(Y|X) = -\frac{n}{2}\log 2\pi - \frac{1}{2}\log|K(X, X) + \sigma_n^2 I| - \frac{1}{2}Y^T\big[K(X, X) + \sigma_n^2 I\big]^{-1}Y.$$

Computing the predictive mean, the predictive covariance matrix, and the marginal likelihood requires inverting the matrix $K(X, X) + \sigma_n^2 I$ at a cost of $\mathcal{O}(n^3)$. This property makes EGP regression intractable for a large number of training points. To overcome this limitation, many approximate or sparse methods have been proposed. Section 2.2 describes two such sparse GP regression approximations.

For the current project, we use the GPyTorch [4] library to implement EGP regression.

2.2 Sparse approximations for GP regression

2.2.1 Inducing point GP

As the name indicates, IPGP regression [13], [2] depends on a small number inducing points $Z = \big[z_1, \ldots, z_m\big] \in \mathbb{R}^{d \times m}$ which sufficiently represent the

rest of the training data. With $\mu(x) = 0$, the IPGP posterior distribution is given by

$$f^*|Y, X, X^* \sim \mathcal{N}(\mu^*, \Sigma^*),$$

where

$$\mu^* = K(X^*, Z)K(Z, Z)^{-1}\mu_z$$

and

$$\begin{aligned}\Sigma^* = K(X^*, X^*) &- K(X^*, Z)^T K(Z, Z)^{-1} K(Z, X^*) \\ &+ K(X^*, Z)K(Z, Z)^{-1} A_z K(Z, Z)^{-1} K(Z, X^*)\end{aligned}$$

with the additional variables μ_z and A_z, which are given by

$$\mu_z = \frac{1}{\sigma_n^2} K(Z, Z)\big[K(Z, Z) + \sigma_n^{-2} K(Z, X)K(X, Z)\big]^{-1} K(Z, X)Y$$

and

$$A_z = K(Z, Z)\big[K(Z, Z) + \sigma_n^{-2} K(Z, X)K(X, Z)\big]^{-1} K(Z, Z).$$

In contrast to EGP regression, only matrices of size $m \times m$ need to be inverted, which reduces the time complexity of IPGP to $\mathcal{O}(nm^2)$ [13].

The inducing points Z are additional hyperparameters. They are initialized by randomly selecting m training points and then optimized together with the other hyperparameters. Despite concerns about overfitting [2], we maximize the marginal likelihood to find the optimal set of hyperparameters. For the experiments, we use the same kernel function for IPGP and EGP regression. As for the EGP regression, we use the GPyTorch [4] library to implement IPGP regression for the current project.

2.2.2 Sparse spectrum GP

For SSGP regression [6], we assume that our regression function is composed of m trigonometric basis functions

$$f(x) = \sum_{i=1}^{m} a_i \cos\left(2\pi s_i^T x\right) + b_i \sin\left(2\pi s_i^T x\right),$$

where the frequency vectors s_i are hyperparameters and the amplitudes a_i, and b_i are assumed to be independent random variables with a prior distribution

$$a_i \sim \mathcal{N}\Big(0, \frac{\sigma_0}{m}\Big), \quad b_i \sim \mathcal{N}\Big(0, \frac{\sigma_0}{m}\Big).$$

Under this assumption, we get a GP for $f(x)$ with mean function zero and covariance function

$$k(x, x') = \frac{\sigma_0}{m}\phi(x)^T\phi(x'),$$

where

$$\phi(x) = \Big[\cos\left(2\pi s_1^T x\right)\sin\left(2\pi s_1^T x\right)\ldots\cos\left(2\pi s_m^T x\right)\sin\left(2\pi s_m^T x\right)\Big]^T.$$

The posterior distribution in this case is given by

$$f^*|Y, X, X^* \sim \mathcal{N}(\mu^*, \Sigma^*),$$

where

$$\mu^* = \Phi_*^T A^{-1}\Phi Y$$

and

$$\Sigma^* = \sigma_n^2\Phi_*^T A^{-1}\Phi_*$$

with the additional variables Φ, Φ_*, and A given by

$$\Phi = \big[\phi(x_1)\ldots\phi(x_n)\big], \quad \Phi_* = \big[\phi(x_1^*)\ldots\phi(x_k^*)\big],$$

and

$$A = \Phi\Phi^T + \frac{m\sigma_n^2}{\sigma_0^2}I.$$

Similar to IPGP regression, the time complexity to compute the posterior distribution for SSGP regression is $\mathcal{O}(nm^2)$ [6].

The hyperparameters (s_i, σ_0^2, σ_n^2) are again trained by maximizing the marginal likelihood

$$\log p(Y|X) = -\frac{1}{2\sigma_n^2}\big(Y^TY - Y^T\Phi^TA^{-1}\Phi Y\big) - \frac{1}{2}\log|A| + m\log\frac{m\sigma_n^2}{\sigma_0^2} - \frac{n}{2}\log 2\pi\sigma_n^2.$$

The SSGP regression model is implemented by the authors based on the PyTorch [10] libray. The implementation is not optimized for speed, i.e., the Cholesky decomposition is not used.

2.3 Gradient boosting regression

In addition to GP regression, we also use GBOOST regression [3] in the experiment. In GBOOST regression, multiple weak learners are trained sequentially, each trying to correct the remaining error[1] of the previous learners. As a result, the algorithm will improve slightly with each additional weak learner. The weak learners in gradient boosting are regression trees. The final prediction model is a combination of all the weak learners together. An advantage of boosting algorithms over other machine learning techniques is that they minimize bias while not significantly increasing variance.

GBOOST has several hyperparameters that can be tuned for a given problem. For the experiments in the current project, we optimize the following four hyperparameters: (1) number of regression trees, (2) maximum depth of the regression trees, (3) fraction of data used for the training of each regression tree, and (4) minimum number of samples to split a node. We select the parameters using a grid search and cross validation.

The GBOOST regression mode is implemented based on the scikit-learn [11] library.

2.4 Datasets

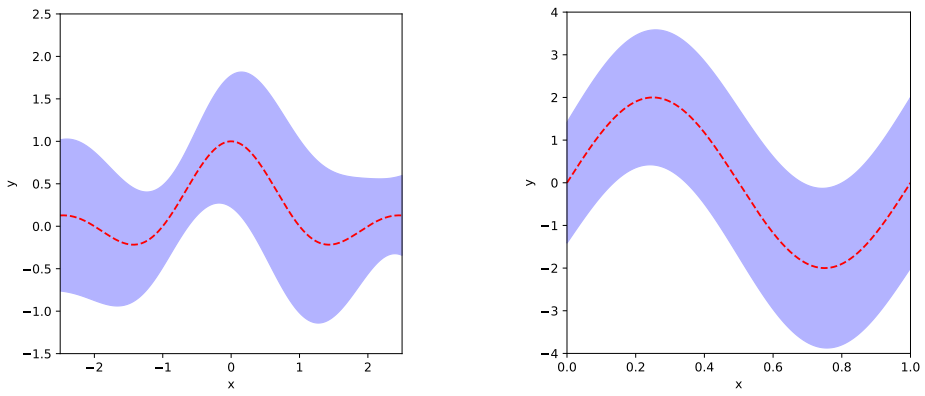
All experiments for this project are conducted with artificial flow field data. The data is sampled from two different fields named Liu [9] and Goldberg [5], which define mean and variance functions over the input space, i.e., the fields are heteroscedastic. Both fields are defined in 1D, 2D, and 3D with a scalar output. Figure 1 shows the mean and variance of both fields in 1D. Figure 2 shows the mean and variance of both fields in 2D. Figures 3 and 4 show the mean of the two fields in 3D.

2.5 Experiments

2.5.1 Experiment 1: Kernel function comparison

In the first experiment, we analyze the effect of the kernel function on the regression performance in 1D. The selection of the kernel function is the most important task of the machine learning engineer when doing GP regression. The kernel function encodes the assumption on how the data is shaped, and it may have a set of hyperparameters that can be trained by maximizing the

[1] We use a squared-error loss for the experiments in the current project.



(a) Liu field

(b) Goldberg fiel

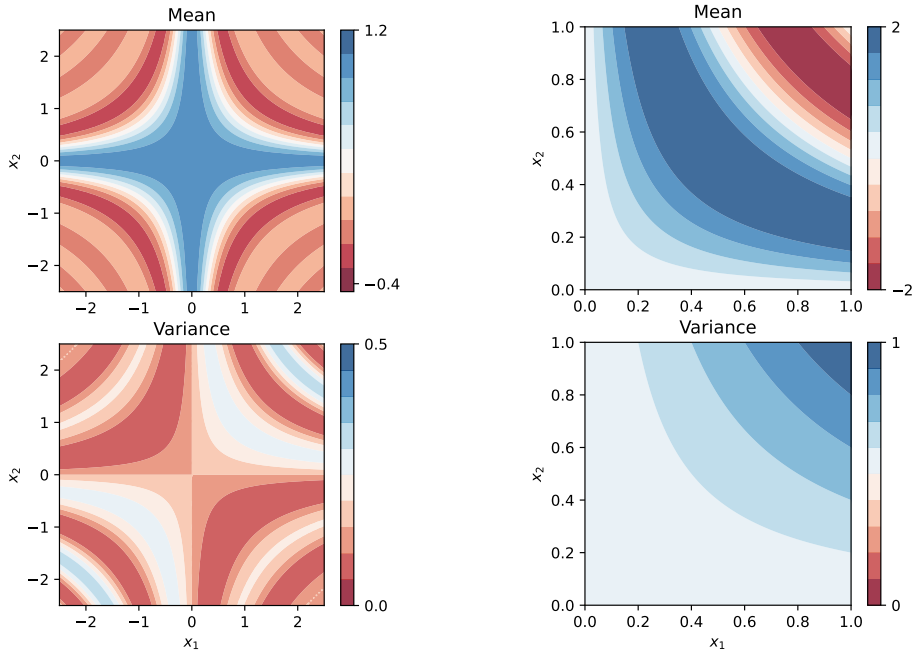
Figure 1: One dimensional data fields used for the experiments. The dashed red lines show the mean of the distributions; the blue shaded areas indicate the variance (as the mean plus/minus two standard deviations).

marginal likelihood.

In this specific experiment, we compare the following kernels: (1) radial basis function (RBF), (2) polynomial, (3) cosine, and (4) Matern. We use the EGP model as described in Section 2.1. For the polynomial kernel, we use a degree of 5. To test the kernels, we use the 1D Liu flow field with 300 training points. We generate the training data 10 times and fit the GP models to each set of data. The hyperparameters are only trained on the first set of data and then reused for the subsequent 9 sets. After each fit of the model, we predict the output on 5000 equally spaced points and compare its mean to the true mean of the flow field. The results of the experiment are described in Section 3.1.1.

2.5.2 Experiment 2: Effect of sparse approximation size

In the second experiment, we investigate how the performance of the sparse GP models changes when changing the sparsity parameter. Both sparse approximation models have such a discrete sparsity parameter, which cannot be optimized using gradient descent and which represents the trade-off between prediction accuracy computational effort. This sparsity parameter is the number of inducing points and the number of trigonometric basis functions for the IPGP and SSGP, respectively. Experiment 2 is conducted to show how the prediction accuracy and the prediction time depend on this sparsity parameter.



(a) Liu field

(b) Goldberg field

Figure 2: Two dimensional data fields for used for the experiments. The top panels show the mean of the distributions; the bottom panels show the variance of the distributions.

The results of the experiment are highly dependent on the data set and cannot be viewed as a general comparison between the models. Preliminary experiments for instance showed that the prediction with the EGP model is faster than the prediction with the sparse models, when the number of training points is low. For this specific experiment, we use the Liu data field in 2D with 1000 training points. We generate the training data 10 times and we retrain the hyperparameters each time. We then predict the output on a grid of 100x101 points and compare the predicted mean to the true mean of the data field. The results of the experiment are described in Section 3.1.2.

2.5.3 Experiment 3: Accuracy and prediction time

In the third experiment, we compare the performance of the EGP, IPGP, SSGP, and GBOOST models in 1D, 2D, and 3D experiments with both artificial flow fields. For the 1D experiments, we use 50 randomly sampled training points and 5'000 equally space test points. For the 2D experiments, we use 500 randomly sampled training points and a grid of 100x101 (=10'100)

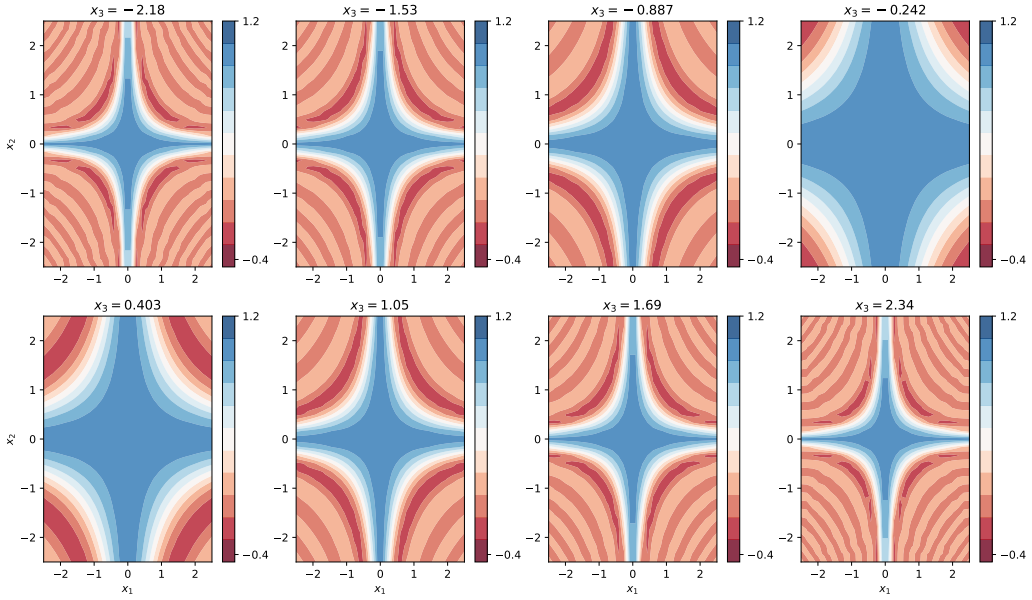


Figure 3: Three dimensional **Liu** data field used for the experiments. Each panel shows the mean over the first two independent dimensions with the third dimension fixed to a specific value. The variance is not shown.

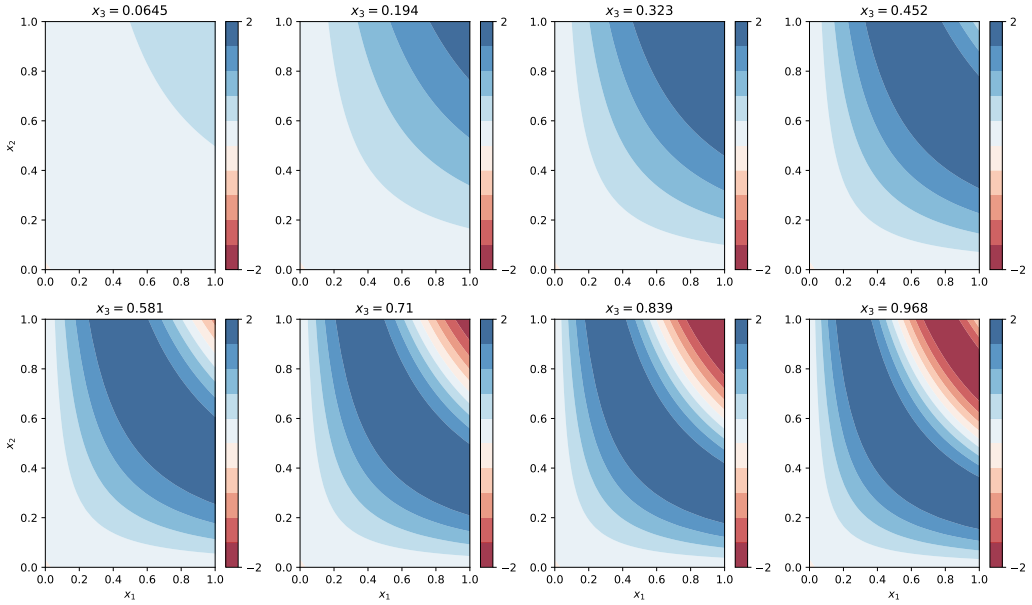


Figure 4: Three dimensional **Goldberg** data field used for the experiments. Each panel shows the mean over the first two independent dimensions with the third dimension fixed to a specific value. The variance is not shown.

test points. For the 3D experiment we use 1000 randomly sampled training points and a grid of 30x31x31 (=29'760) test points. The prediction error is calculated as the mean squared error (MSE) between the predicted mean and the true mean of the distributions on the test points. The prediction time is the measured time to predict the output on all the test points normalized to 1000 test point[2]. All experiments are repeated 10 times with a new set of training data and the results are reported as mean and standard deviation of these 10 repetitions.

For the EGP and IPGP models, we are using the RBF kernel. For the sparse GP models, the number of inducing points or basis functions was manually selected for each experiment. For the IPGP, the number of inducing points is 10, 50, and 100 for the 1D, 2D, and 3D cases, respectively. Similarly, for the SSGP, the number of basis functions is 5, 20, and 60, for the 1D, 2D, and 3D cases. All other hyperparameters of the four regression models are optimized numerically and stored before running the experiment. The results of the experiments are described in Section 3.1.3.

[2]E.g. the measured time to predict 29'760 test points is divided by 29.76 to get the normalized value.

Table 1: Kernel comparison results. The table lists the MSE between the predicted mean and the true mean of the data field for 10 training data sets.

Kernel type	MSE (mean ± std)
RBF	0.0041 ± 0.0031
Polynomial	0.0348 ± 0.0036
Cosine	0.1940 ± 0.0030
Matern	0.0053 ± 0.0032

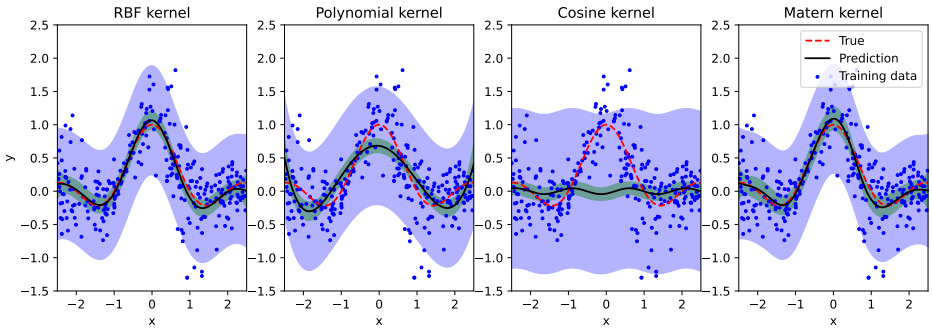


Figure 5: Example results of the kernel comparison experiment for one training data set. Each subplot shows the same true mean and training data, but different prediction mean standard deviation areas.

3 Results

3.1 Experiments

3.1.1 Experiment 1: Kernel function comparison

Table 1 lists the results of the kernel comparison experiment. Clearly, the RBF and the Matern kernels are well suited for the given data, while the polynomial kernel does not work well and the cosine kernel does not work at all. Figure 5 shows the experimental results for one of the training data sets. The figure confirms the superior fit of the RBF and Matern kernels.
The results of the kernel comparison experiment confirm the popular view that the RBF kernel is a good choice for most data, while kernels like the cosine kernel are only suitable for very specific (e.g. periodic) data. All other experiments in this project are therefore conducted with the RBF kernel.

3.1.2 Experiment 2: Effect of sparse approximation size

Figure 6 shows the results of the sparsity experiment. The top row shows that the prediction accuracy improves with both sparse models when the sparsity parameter is increased. Nevertheless, the EGP model consistently provides a better accuracy. Additionally, the SSGP model shows a higher variance in the accuracy compared to the other models at least for a low number of trigonometric basis function.
The bottom row of Figure 6 shows the prediction time. With 1000 training samples, the sparse models are clearly faster than the EGP model. The IPGP model is also substantially faster than the SSGP model. However, this difference can be mostly attributed to the unoptimized implementation of the SSGP model[3]. While the prediciton time for the SSGP model increases with the number of basis functions, no substantial increase is visible with the number of inducing points for the IPGP model.

3.1.3 Experiment 3: Accuracy and prediction time

Table 2 lists the results of the main regression experiment. In all experiments (except for the Liu field in 2D), the EGP and IPGP models yield very similar prediction errors. The GBOOST model's accuracy strongly depends on the flow field. While it's en par with the other models in the Liu field, it performs substantially worse on the Goldberg field. In 3D for the Liu field, GBOOST yields the lowest prediction error of all models. The performance of the SSGP model is similar to that of the other GP models, but not consistently better or worse.

The prediction time of the GP models for 1000 test points varies with the number of training points. For 50 and 500 training points (1D and 2D cases) the EGP model is faster than both sparse GP models. Only for 1000 training points (3D case), the IPGP model is faster than the EGP model. The GBOOST model is extremely fast in comparison, while the SSGP model is substantially slower. As explained in Section 3.1.2, the SSGP model is slow, because it's implementation has not been optimized for speed.

Figures 7, 8, and 9 show example results of the regression experiments in 1D, 2D, and 3D, respectively.

[3]The SSGP model was implemented by the authors and the code was not optimized for speed; the IPGP and EGP models are based on GPyTorch [4].

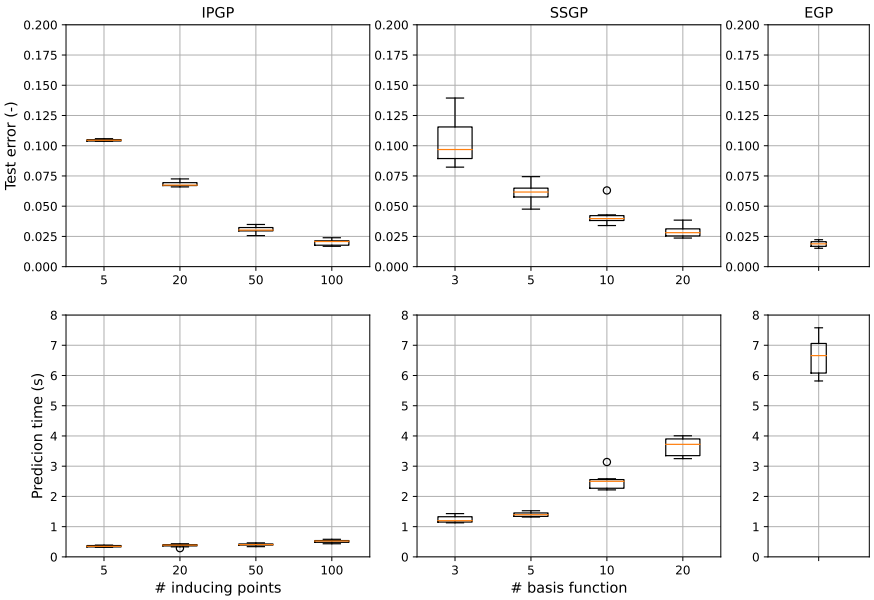


Figure 6: Comparison of the prediction accuracy and prediction time of the sparse GP models with the sparsity parameter varied. The top row shows the MSE of the predicted mean versus the true mean of the field. The bottom row shows the required time for the prediction of all test samples. The left-hand side column shows the dependency of the IPGP model on the number of inducing points; the middle column shows the dependency of the SSGP model on the number of trigonometric basis functions. The right-hand side column shows the EGP model as reference.

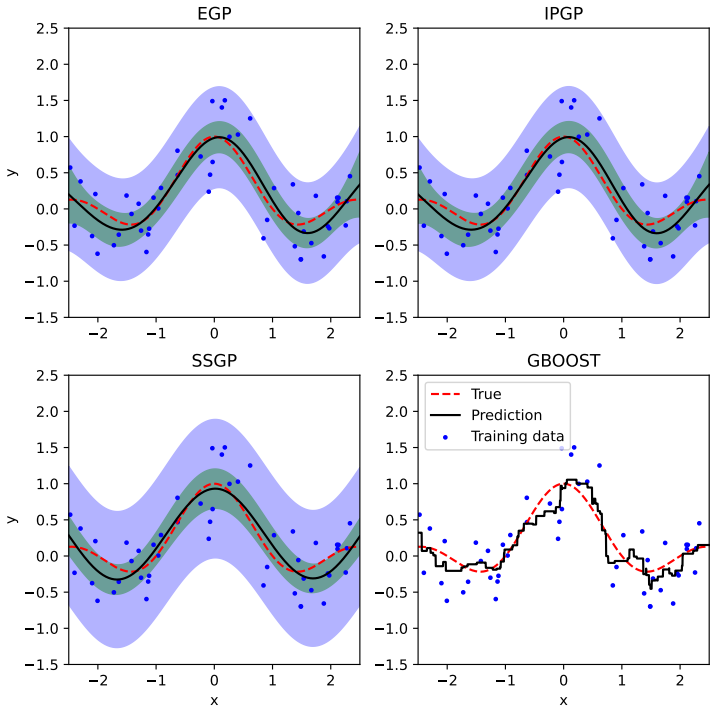


Figure 7: Example results of the 1D regression experiment with the Liu field. The four panels show the true mean, the training data, and the prediction for all four models. For the GP models, the shaded areas additionally indicate the model variance (green) and the data variance (blue) as the mean plus/minus two standard deviations.

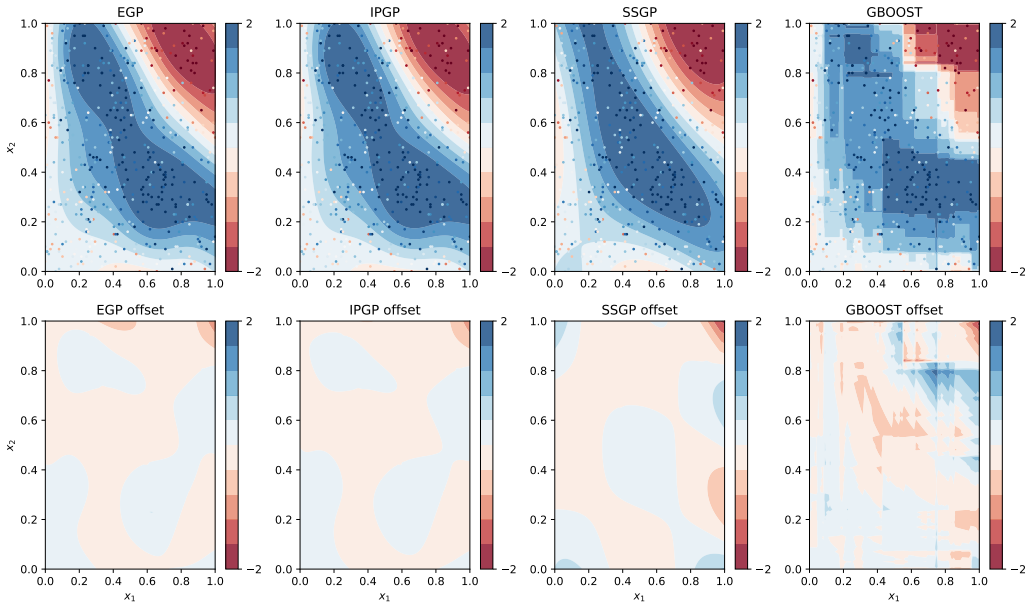


Figure 8: Example results of the 2D regression experiment with the Goldberg field. The top row shows the training data and the predicted mean for all four models. The bottom row shows the deviation of the prediction from the true mean of the field. The true mean is is displayed in the top-right panel in Figure 2.

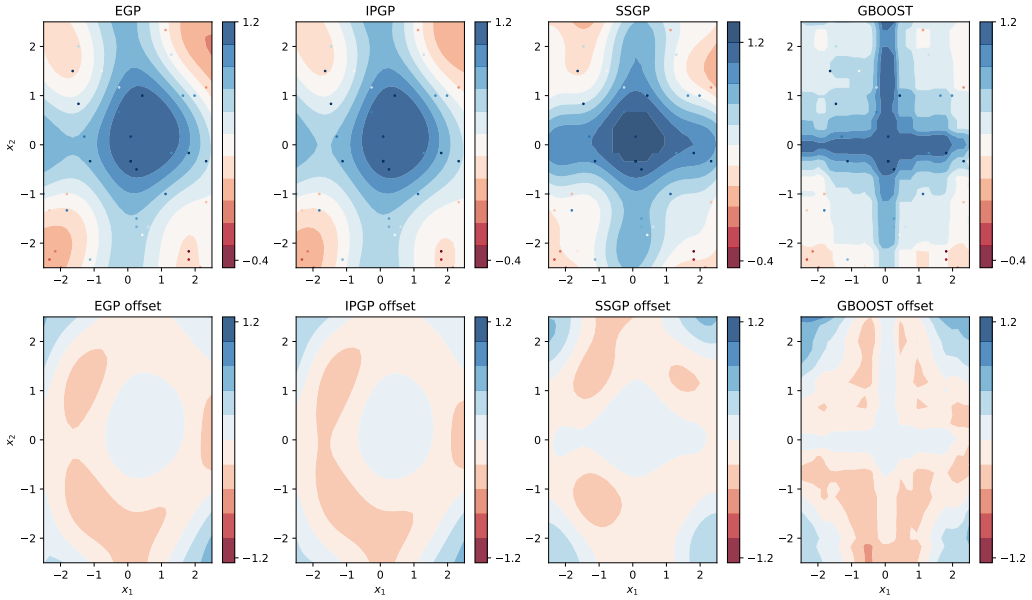


Figure 9: Example results of the 3D regression experiment with the Liu field for the slice at $x_3 = -0.242$. The top row shows the training data and the predicted mean for all four models. The bottom row shows the deviation of the prediction from the true mean of the field. The true mean for this slice is displayed in the top-right panel in Figure 3.

Table 2: Main results of the regression experiments in 1D, 2D, and 3D with both flow fields. The test error columns list the MSE of the prediction versus the true mean of the fields. The prediction time column lists the measured time to predict 1000 test points averaged over both flow fields.

Dim.	Model	MSE - Liu (mean ± std)	MSE - Goldberg (mean ± std)	Pred. time (s) (mean ± std)
1	EGP	0.0224 ± 0.013	0.078 ± 0.039	0.002 ± 0.000
	IPGP	0.0224 ± 0.013	0.078 ± 0.039	0.019 ± 0.001
	SSGP	0.0236 ± 0.011	0.061 ± 0.051	0.137 ± 0.005
	GBOOST	0.0400 ± 0.017	0.186 ± 0.064	0.000 ± 0.000
2	EGP	0.0338 ± 0.006	0.036 ± 0.010	0.010 ± 0.001
	IPGP	0.0461 ± 0.005	0.035 ± 0.010	0.037 ± 0.002
	SSGP	0.0480 ± 0.002	0.055 ± 0.006	0.338 ± 0.012
	GBOOST	0.0433 ± 0.004	0.119 ± 0.016	0.000 ± 0.000
3	EGP	0.0833 ± 0.002	0.036 ± 0.007	0.503 ± 0.037
	IPGP	0.0840 ± 0.002	0.034 ± 0.007	0.126 ± 0.007
	SSGP	0.0689 ± 0.002	0.045 ± 0.007	0.877 ± 0.076
	GBOOST	0.0576 ± 0.003	0.121 ± 0.014	0.000 ± 0.000

4 Discussion

The results from Experiment 3 show that both sparse GP models may be well suiteds for the SMARTAIR project. While the EGP model is quite fast at prediction with a low number of training points (< 1000), it becomes very slow when the number of training points increases. The prediction time of both sparse GP models increases much slower when the number of training points increases. At the same time, the sparse GP models achieve a similar accuracy as the EGP model. The non-GP reference GBOOST achieves a good accuracy on the Liu data field, but does not work well on the Goldberg field. However, it's prediction time is extremely fast in comparison. Nevertheless, the GBOOST model does not output a variance estimate of the predicted function and therefore cannot be used for the steering of the probe without further modifications.

The results also show that the theoretical efficiency of an algorithm is not the only aspect to consider. The SSGP model, which is implemented by the authors, is substantially slower than the GP models that are implemented in GPyTorch. In order to use the SSGP model for the SMARTAIR project, the implementation would have to be optimized. To start such an optimization, the authors would further study the GPyTorch implementation and transfer the used techniques to the SSGP implementation.

The results from Experiment 1 confirm that the RBF kernel is a good choice for the analyzed datasets. However, the heteroscedasticity of the data is not reflected in any of the analyzed GP models. Considering that one important aspect of the project is to get a good estimate of the variance of the predicted function, heteroscedastic GP models [7, 8] should also to be investigated.

Last but not least, the authors did not address the problem of applying the GP models in real-time. To use a GP model in real time, one would have to solve the problem of efficiently adding new training data to the model. Additionally, one would have to figure out how to adapt the hyperparameters (e.g. the location of the inducing points) when new training data is continuously added. These problems will have to be solved in further projects.

5 References

References

[1] Smartair project description. `https://datascience.ch/project/smartair/`. Accessed: 2023-06-02.

[2] Sparse gaussian processes tutorial. `http://krasserm.github.io/2020/12/12/gaussian-processes-sparse/`. Accessed: 2023-06-08.

[3] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[4] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.

[5] Paul Goldberg, Christopher Williams, and Christopher Bishop. Regression with input-dependent noise: A gaussian process treatment. *Advances in neural information processing systems*, 10, 1997.

[6] Miguel Lázaro-Gredilla, Joaquin Quinonero-Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal. Sparse spectrum gaussian process regression. *The Journal of Machine Learning Research*, 11:1865–1881, 2010.

[7] Miguel Lázaro-Gredilla and Michalis K Titsias. Variational heteroscedastic gaussian process regression. In *ICML*, pages 841–848, 2011.

[8] Quoc V Le, Alex J Smola, and Stéphane Canu. Heteroscedastic gaussian process regression. In *Proceedings of the 22nd international conference on Machine learning*, pages 489–496, 2005.

[9] Haitao Liu, Yew-Soon Ong, and Jianfei Cai. Large-scale heteroscedastic regression via gaussian process. *IEEE transactions on neural networks and learning systems*, 32(2):708–721, 2020.

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[11] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu. Brucher, MMatthieu Perrot, and Edouard. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Carl Edward Rasmussen, Christopher KI Williams, et al. *Gaussian processes for machine learning*, volume 1. Springer, 2006.

[13] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009.