CompSci Project 3: Gaussian Process Regression

Kosio Beshkov*
Department of Bioscience, University of Oslo, Norway

Mohamed Safy†
Department of Chemistry, University of Oslo, Norway

Tim Zimmermann‡
Institute of Theoretical Astrophysics, University of Oslo, Norway
(Dated: September 1, 2024)

This project investigates the properties and shortcomings of *gaussian processes* (GP), a parameter-free, probabilistic regression model for the prediction of high-dimensional data generative processes. Our in-depth study may be split into three aspects:

Kernel Design: Guided by properties of a noisy Friedman function realization — our synthetic data set — we tailor a series of correlation functions, implement them and benchmark their predictive performance, thereby surpassing the regression quality of a GP based on the universal squared exponential kernel.

Overconfidence and Extrapolation: GPs have the advantage of providing not just a single estimate for the correct prediction, but rather a distribution over functions. From such a distribution one obtains a degree of confidence (belief) at each point of the predicted function. We explore when overconfidence occurs in GPs in the settings of predicting close to the training data (interpolation) and far away from it (extrapolation).

Large Datasets: The application of full GP regression is usually confined to small datasets due to the intrinsic time complexity of the kernel training. Thus, we explore a *product-of-expert* approximation of the former in conjunction with a *robust bayesian committee machine aggregation* to distribute GPs beyond one execution thread. Our implementation of the scheme outperforms `sklearn`'s undistributed GP method in terms of runtime while maintaining its predictive quality.

I. INTRODUCTION

Our discussion departs with a quick recap of the regression problem for noisy datasets and some motivating remarks on why a fully probabilistic treatment of the inference of a latent function may be a promising alternative route besides canonical, deterministic regression methods. In this context, section I A introduces the idea of *gaussian processes* (GP) — gaussian probability distributions over function space — as the pivotal, practically tractable object with which such a probabilistic treatment becomes feasible. We elaborate on how prediction is carried out in this framework in section I B and discuss how to encode prior knowledge about the sought-after, latent function with so called parametrized *kernels* in section I C. How to train these kernels is explained in section I D. We close by introducing the *robust bayesian committee machine* as an approach for *distributed* gaussian process regression meant to address the computational complexity of standard GPs.

Apart from section I E, all results presented in section I are essentially a paraphrasing of [1] and we refer to this standard reference for more information.

A. Gaussian Processes for Regression

We remind the reader of the regression problem formulation of project 1, see [2]. Suppose we are given a realization of a noisy measurement vector $\boldsymbol{y} \in \mathbb{R}^N$ of a data generative process — a function — $g\colon \mathbb{R}^D \to \mathbb{R}$ at randomly chosen but fixed observation sights $\boldsymbol{X} = \begin{pmatrix}\boldsymbol{x}_1 & \boldsymbol{x}_2 & \dots & \boldsymbol{x}_N\end{pmatrix}^\intercal \in \mathbb{R}^{N\times D}$ such that:

$$y_i = g(\boldsymbol{x}_i) + \epsilon_i \,, \qquad \epsilon_i \mid \boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{x}_i|0,\sigma^2) \,, \tag{1}$$

with ϵ_i denoting the iid random variables — the noise — drawn from a normal distribution of common variance σ^2 so that $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon} \mid \boldsymbol{0}, \sigma^2\mathbb{1})$.

Regression then attempts to model the deterministic function g by means of an estimator (model) $\hat{g}$, constructed from $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$, so that predictions with *unseen* test data $\boldsymbol{X'} \in \mathbb{R}^{N'\times D}$ satisfy $\hat{\boldsymbol{g}}(\boldsymbol{X'}) \approx \boldsymbol{g}(\boldsymbol{X'})$ with shorthand notation $(\boldsymbol{f}(\boldsymbol{X'}))_i = f(\boldsymbol{x}'_i)$.

In the context of *linear* regression this entails *postulating* a parametrized class of model functions:

$$\hat{g}_{\boldsymbol{\Theta}}\colon \mathbb{R}^N \to \mathbb{R} \,, \quad \boldsymbol{x} \mapsto \hat{g}_{\boldsymbol{\Theta}}(\boldsymbol{x}) \equiv \boldsymbol{\Phi}(\boldsymbol{x})^\intercal \boldsymbol{\Theta} \,, \tag{2}$$

* constb@ibv.uio.no
† m.e.a.safy@kjemi.uio.no
‡ tim.zimmermann@astro.uio.no

with $\boldsymbol{\Theta}$ as adjustable parameter vector and $\boldsymbol{\Phi}$ a basis function transformation, e.g. $\Phi_j(\boldsymbol{x}_i) = \boldsymbol{x}_i^j$ for polynomial models.

Although simple in theory, this approach comes with a clear caveat: One needs to specify the class of permissible regression models $\hat{g}_{\boldsymbol{\Theta}}$, alongside its complexity, e.g. the polynomial degree, *a priori*. Depending on the unknown process $g(\boldsymbol{x})$, this may be a formidable task in its own and we remind the reader of the elaborate model selection analysis carried out in [2].

An alternative course of action that circumvents the model specification may be summarized as follows: (i) Instead of investing 100% of our prior belief in one class of functions, in the example above polynomials of a certain degree, we assign a *prior probability* to all functions $f(\boldsymbol{x})$ consistent with our assumptions on the properties of the true, latent function, e.g. smoothness, periodicity, symmetry, etc. (ii) Upon looking at training data $\mathcal{D}$, we reject all such functions that are not supported by the training data, i.e. don't reproduce it. (iii) Averaging all functions that are left based on their updated *posterior probabilities* then yields an estimator $\hat{g}(\boldsymbol{x}_i') = \mathbb{E}[f(\boldsymbol{x}_i')] \approx g(\boldsymbol{x}_i')\ \forall \boldsymbol{x}_i'$.

Each of the steps translates into a well-defined mathematical problem. First, we specify a *stochastic process* — a (prior) probability distribution on function space. Second, we *condition* the stochastic process on the training data, thereby rejecting trial functions which are no longer consistent with the information at hand. At last, we compute the expectation value for $f(\boldsymbol{x})$ under the conditioned (posterior) distribution.

We refer to section I B for the latter two steps. Here we give additional information on (gaussian) stochastic processes as probability distributions over function space.

Loosely speaking, a stochastic process on $\mathbb{R}^D$ may be characterized as a set $\{F(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathbb{R}^D\}$ of random variables $F(\boldsymbol{x})$ associated with each element of the, here uncountable, index set $\mathbb{R}^D$. A *realization* of such a process may then be understood as a function $f(\boldsymbol{x})$ that follows from jointly fixing all random variables F at all $\boldsymbol{x}$ — a procedure consistent with the notion of drawing from the process' probability distribution $\mathcal{P}_{F(\boldsymbol{x})}(f(\boldsymbol{x}))$ defined over *function space*.

Clearly, generic stochastic processes are rather cumbersome objects to work with. We thus limit our scope to *gaussian processes* which may be defined on a practical level as follows:

Definition I.1 (Gaussian Process, [1])**.** A Gaussian process (GP) is a collection of random variables any finite number of which have a joint Gaussian distribution.

As for finite multivariate gaussian distributions, GPs are fully determined by their first and second moment, i.e. a mean *function* $m(\boldsymbol{x})$ and covariance *function* — or kernel — $k(\boldsymbol{x}, \boldsymbol{x}')$. Thus, a GP my also written as:

$$f(\boldsymbol{x}) \sim \mathcal{GP}\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')\right) \ . \tag{3}$$

A couple of remarks are in order:

Firstly, practice shows that setting $m(\boldsymbol{x}) = 0$ and centering the training data $\boldsymbol{y} \to \boldsymbol{y} - \bar{y}$ is sufficient in most cases. Consequently, the only degree of freedom left is the kernel which will encode all sought-after properties of our regression model, e.g. smoothness, periodicity or symmetry, see section I C.

Secondly, note that Definition I.1 actually contains a claim: Although a GP is defined on an infinite dimensional function space, as long as we ask for finite objects like the function value vector $\boldsymbol{f}(\boldsymbol{X})$ of a GP-sample function, we are allowed to restrict ourselves to finite dimensional joint probability distributions. For instance, the aforementioned discrete realization $\boldsymbol{f}$ at sights $\boldsymbol{X}$ of a GP sample $f(\boldsymbol{x})$ follows as random draw from:

$$\boldsymbol{f}|\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{0}, k(\boldsymbol{X}, \boldsymbol{X}))\ , \quad k(\boldsymbol{X}, \boldsymbol{X}) \in \mathbb{R}^{N \times N}\ . \tag{4}$$

An explanation for why this holds true, i.e. why finite dimensional observables may be deduced from finite dimensional probability distributions is beyond the scope of this work and we will tacitly accept this claim as true.

Thirdly, besides the property of GPs to combine the infinite dimensional sophistication with a tractable finite dimensional formalism, gaussians provide additional, highly convenient properties. Most notably, one finds *the set of all multivariate gaussians to be closed under multiplication, division, conditioning and marginalization*, i.e. if the input to these operations is gaussian so is their output. We refer to the Appendix for a summary of the most important identities and already anticipate that the gaussian process regression procedure outlined above will result in a gaussian posterior probability distribution since the prior distribution will be a discretized GP-prior.

Lastly, the cosmologist of the group feels obliged to mention that GPs are not just a probabilistic tool for regression or classification but also find application in other fields. For instance, currently favored inflationary theories predict the initial conditions of the universe to be a realization of a GP with an isotropic, power-law power spectrum (the Fourier transform of the kernel).

B. Gaussian Process Prediction

Back to the problem of function regression. To construct a GP-based estimator, we first construct a joint, prior probability distribution for the noisy data vector $\boldsymbol{y}$ at training points X and the sought-after, noise-free function values $\boldsymbol{f}' \equiv \boldsymbol{f}(\boldsymbol{X}')$ at test sights $\boldsymbol{X}' = \begin{pmatrix}\boldsymbol{x}_1' & \dots & \boldsymbol{x}_{N'}'\end{pmatrix}^\intercal \in \mathbb{R}^{N' \times D}$. Since we assign probabilities to functions based on a gaussian process and only attempt to predict discretized versions of such functions, this yields:

$$\boldsymbol{f}', \boldsymbol{y} \mid \boldsymbol{X}', \boldsymbol{X} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} k(\boldsymbol{X}', \boldsymbol{X}') & k(\boldsymbol{X}', \boldsymbol{X}) \\ k(\boldsymbol{X}, \boldsymbol{X}') & k(\boldsymbol{X}, \boldsymbol{X}) + \sigma \mathbb{1} \end{bmatrix}\right)\ , \tag{5}$$

where the elements of the covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{(N+N') \times (N+N')}$ are set by evaluating the chosen kernel function at all points in $\{\boldsymbol{X}, \boldsymbol{X}'\}$, i.e $\Sigma_{ij} =$

$k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{x}_i, \boldsymbol{x}_j \in \{\boldsymbol{X}, \boldsymbol{X}'\}$. Notice the increased variances for $\boldsymbol{y}$ due to the isotropic noise contribution $\sigma^2\mathbb{1}$ and the lack thereof for the variances of $\boldsymbol{f}'$ — after all our goal is to model the noise-free, latent function g.

Realize the known training targets $\boldsymbol{y}$ are still unspecified in eq. (5). We change this by *conditioning* eq. (5) on $\boldsymbol{y}$, thereby asking the question: Given the noisy measurement $\boldsymbol{y}$, its evaluation sights $\boldsymbol{X}$ and the prediction locations $\boldsymbol{X}'$, what is the degree of belief we assign to an arbitrary function consistent with this data?

Using the conditioning property of multivariate gaussians in eq. (A.4), we arrive at the posterior distribution:

$$\boldsymbol{f}' \mid \boldsymbol{y}, \boldsymbol{X}', \boldsymbol{X} \sim \mathcal{N}\left(\bar{\boldsymbol{f}}, \Sigma_{\boldsymbol{f}}\right) , \tag{6}$$

with:

$$\bar{\boldsymbol{f}}(\boldsymbol{X}') = K(\boldsymbol{X}', \boldsymbol{X})[K(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\mathbb{1}]^{-1}\boldsymbol{y} , \tag{7}$$

$$\begin{aligned}\Sigma_{\boldsymbol{f}}(\boldsymbol{X}') = {} & K(\boldsymbol{X}', \boldsymbol{X}') \\ & - K(\boldsymbol{X}', \boldsymbol{X})[K(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\mathbb{1}]^{-1}K(\boldsymbol{X}, \boldsymbol{X}') .\end{aligned} \tag{8}$$

Now, since the predictive distribution (6) is still gaussian, executing the last step, i.e. computing the expectation value over all function with respect to eq. (6) is trivial and results in:

$$\hat{g}(\boldsymbol{X}') = \bar{\boldsymbol{f}}(\boldsymbol{X}') . \tag{9}$$

A couple of remarks are in order:

Firstly, note that we choose to construct all elements of $\bar{\boldsymbol{f}}$ *simultaneously* by considering the mean of the *joint* distribution of eq. (6). An alternative, in practice often favoured, approach is to consider each test sight $\boldsymbol{x}_i'^{\intercal} = \boldsymbol{e}_i^{\intercal}\boldsymbol{X}'$ independently and then iterate through all test sights sequentially. On a practical level, this means using the univariate gaussian:

$$f_i' \mid \boldsymbol{y}, \boldsymbol{x}_i', \boldsymbol{X} \sim \mathcal{N}\left(\bar{f}_i', \sigma^2_{f_i'}\right) , \tag{10}$$

with:

$$\bar{f}_i' = K(\boldsymbol{x}_i', \boldsymbol{X})[K(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\mathbb{1}]^{-1}\boldsymbol{y} , \tag{11}$$

$$\begin{aligned}\sigma^2_{f_i'} = {} & K(\boldsymbol{x}_i', \boldsymbol{x}_i') \\ & - K(\boldsymbol{X}', \boldsymbol{X})[K(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\mathbb{1}]^{-1}K(\boldsymbol{X}, \boldsymbol{X}')\end{aligned} \tag{12}$$

instead of eq. (6).

The added benefit of this approach is access to a simple confidence interval, $\sigma_{f_i'}$, for each predicted function value. It also allows for a more intuitive interpretation of how GP regression operates:

The predicted function value $\bar{f}_i'$ is a linear combination of all observed, noisy values y_j and weights that depend on the correlation between $\boldsymbol{x}_i'$ and $\boldsymbol{x}_j$. Positive correlation results in positive weights and uptake of the function value, anti-correlation, by contrast, implies a function value decreasing contribution to the linear combination.

The posterior variance, on the other hand, corresponds to the prior variance $k(\boldsymbol{x}_i', \boldsymbol{x}_i')$ reduced by any non-zero correlation between $\boldsymbol{x}_i'$ and the training sights $\{\boldsymbol{x}_j\}_{j=1\ldots N}$. For test points far away from the training set $\boldsymbol{k}(\boldsymbol{x}_i', \boldsymbol{X}) \to 0$, implying we recover our prior uncertainty $k(\boldsymbol{x}_i', \boldsymbol{x}_i')$ as error measure.

We close by mentioning that eq. (6) and eq. (10) require the inversion of a real, symmetric $N \times N$ matrix, or equivalently a method for solving $(K(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\mathbb{1})\boldsymbol{\alpha} = \{\boldsymbol{y}, K(\boldsymbol{X}, \boldsymbol{x}_i')\}$. Luckily, if $\sigma > 0$ its existence is assured, see [2] for the argument leading to this conclusion. A numerically stable way to approach the situation is a Cholesky decomposition of the form:

$$K(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\mathbb{1} = \boldsymbol{L}\boldsymbol{L}^{\intercal} , \tag{13}$$

with L as lower triangular matrix. Solving the system of equations then follows from two back-substitution steps, expressed as \-operation:

$$\boldsymbol{\alpha} = L^{\intercal}\backslash(L\backslash\{\boldsymbol{y}, \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{x}_i')\}) . \tag{14}$$

The time complexity of this computation is $\mathcal{O}(N^3)$ and is dominated by the Cholesky decomposition step. However, note eq. (13) is independent of the test data $\boldsymbol{X}'$. This implies we can compute the decomposition once (as part of the kernel training, see section I D), cache the result and reuse it for all predictions. This reduces the time complexity of the prediction stage to $\mathcal{O}(N^2)$.

C. Kernels

Clearly, the selection of an appropriate kernel $k(\boldsymbol{x}, \boldsymbol{x}')$ has critical importance for gaussian process regression. As mentioned before, it is the kernel function in which we encode all our prior knowledge about the data generative process $g(\boldsymbol{x})$ and express it in terms of how correlated two function values at points $\boldsymbol{x}$ and $\boldsymbol{x}'$ should be.

To assure well-posedness, the covariance matrix of a multivariate gaussian must by positive-semidefinite and invertible. Both can be guaranteed if we construct the former by covariance functions that satisfy:

$$\int k(\boldsymbol{x}, \boldsymbol{x}')f(\boldsymbol{x})f(\boldsymbol{x}')\mathrm{d}^D x \mathrm{d}^D x' > 0 \, , \forall f \in L_2(\mathbb{R}^D) . \tag{15}$$

Beyond this, no formal requirements exist which gives rise to an entire zoo of permissible correlation functions.

1. *The Squared Exponential Kernel*

To keep the discussion compact, we refer to [3] for a in-depth discussion on commonly used kernels, and only focus on variations of the canonical *squared exponential kernel* or *radial basis function kernel* (RBF):

$$k_{\mathrm{RBF}}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2}\frac{|\boldsymbol{x} - \boldsymbol{x}'|^2}{l^2}\right) . \tag{16}$$

with l the tunable *characteristic length scale* of the process. Noteworthy properties of the RBF kernel are (i) its infinite differentiability translating into smooth, more precisely infinite mean square differentiable, GP-realizations, [1], (ii) its universality, i.e. the ability of a RBF-GP to approximate *any* function to arbitrary precision given enough data $\mathcal{D}$, [4] and (iii) its isotropic nature, $k(\boldsymbol{x}, \boldsymbol{x}') = f(|\boldsymbol{x} - \boldsymbol{x}'|)$.

An common extension of the RBF kernel, known as *automatic relevance detection* (ARD) RBF, allows for different length scales $\boldsymbol{l} = \left(l_1 \dots l_D\right)^\intercal$ so that:

$$k_{\text{ARD}}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2}|\left(\boldsymbol{x} - \boldsymbol{x}'\right)/\boldsymbol{l}|^2\right) \ . \tag{17}$$

where the /-operation denotes element-wise division.

2. Constructing New Kernels from Old Ones

a. Compositions A particular convenient property of kernel functions is their closedness under addition, multiplication and linear transformation, i.e. if $k_1(\boldsymbol{x}, \boldsymbol{x}')$ and $k_2(\boldsymbol{x}, \boldsymbol{x}')$ are valid correlation functions so are:

$$k_+(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}') \tag{18}$$
$$k_\times(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') \cdot k_2(\boldsymbol{x}, \boldsymbol{x}') \tag{19}$$
$$k_{\alpha,\beta}(\boldsymbol{x}, \boldsymbol{x}') = \alpha k_1(\boldsymbol{x}, \boldsymbol{x}') + \beta \ , \quad \alpha, \beta \in \mathbb{R} \tag{20}$$

Note that adding two kernels corresponds to a logical `OR` while multiplication implements an `AND` operation.

b. Warping New kernels can also be constructed by transforming the input vectors via a (non)linear transformation $\boldsymbol{u}\colon \mathbb{R}^D \to \mathbb{R}^M$ before plugging it into the kernel:

$$k_{\boldsymbol{u}}(\boldsymbol{x}, \boldsymbol{x}') = k\left(\boldsymbol{u}(\boldsymbol{x}), \boldsymbol{u}(\boldsymbol{x}')\right) \tag{21}$$

This allows us to implement the idea of a *projection kernel*: If $\boldsymbol{A}$ denotes the projection onto a lower dimensional subspace of $\mathbb{R}^D$, say the x_1-direction with $\boldsymbol{A} = \boldsymbol{e}_1\boldsymbol{e}_1^\intercal$, then:

$$k_{\boldsymbol{A}}(\boldsymbol{x}, \boldsymbol{x}') = k\left(\boldsymbol{A}\boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}'\right) \tag{22}$$

produces a kernel which only measures correlations in this subspace. Combining the `AND` operation of section I C 2 a with subspace-projections then allows for correlating selected dimensions at will — a construction we exploit in section III B.

c. Symmetrization Let $D = 1$. It is then also possible to restrict GP sample functions to be *symmetric* around at $x = x_S$. The corresponding kernel reads:

$$k_S(x, x') = \frac{1}{2}\left[k\left(x_S - x, x' - x_S\right) + k\left(x - x_S, x' - x_S\right)\right] \ . \tag{23}$$

D. Gaussian Process Training

The prediction procedure outlined in section I B assumed a fully specified correlation function. However, the discussion of section I C made clear that common kernels depend on various hyperparameters $\boldsymbol{\Theta}$ such as length scales or amplitudes. Clearly, the quality of the regression result will be sensitive to the suitedness of these hyperparameters for the data at hand. To tune them a pre-prediction training/fitting stage is required. Here, we sketch the canonical approach to GP-fitting, as described in [1].

Since GP-regression may be understood as a bayesian method, it is only plausible to approach the hyperparameter optimization in the same vein and to do so we turn to the posterior distribution over $\boldsymbol{\Theta}$ given training data $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$. According to Bayes' theorem, we have:

$$p(\boldsymbol{\Theta}|\boldsymbol{y}, \boldsymbol{X}) \propto p(\boldsymbol{y} \mid \boldsymbol{\Theta}, \boldsymbol{X}) p(\boldsymbol{\Theta}) \ . \tag{24}$$

The likelihood, $p(\boldsymbol{y} \mid \boldsymbol{\Theta}, \boldsymbol{X})$, is easily obtained by revisting eq. (1). As the sum of two gaussians, i.e. the noise distribution and the GP-prior, is still gaussian, one finds:

$$p(\boldsymbol{y} \mid \boldsymbol{\Theta}, \boldsymbol{X}) = \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{0}, k_\Theta(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \mathbb{1}\right) \ . \tag{25}$$

We mention in passing that eq. (25) is also referred to as *marginal likelihood* — terminology we will adopt but not explain any further.

In a fully consistent setting, the next step would be a specification of a hyperparameter prior $p(\boldsymbol{\Theta})$ followed by extracting summary statistics alongside confidence regions of the posterior distribution $p(\boldsymbol{\Theta} \mid \boldsymbol{y}, \boldsymbol{X})$. For reasons that remain partially elusive to the authors, this is not done in practice. Instead, the standard approach is a maximum likelihood estimation (MLE) of $\boldsymbol{\Theta}$:

$$\boldsymbol{\Theta} = \arg\max_\Theta \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{0}, k_\Theta(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \mathbb{1}\right) \ , \tag{26}$$

which is identical to the *maximum a posterior estimation* (MAP) assuming a flat prior — an assumption probably not to unwieldy.

The MLE is carried out by using the log-marginal likelihood as a cost function in conjunction with a gradient-based optimization technique such as LBFGS, see [2] for more details. We refer to [1] for the gradient equation for $\log p(\boldsymbol{y} \mid \boldsymbol{\Theta}, \boldsymbol{X})$ and only note that its expression again demands for the evaluation of a $N \times N$ matrix inverse. GP-fitting is thus an $\mathcal{O}(N^3)$ process, making its application to training sets with $N > 10^4$ points intractable. Section I E explores potential approximative techniques to mitigate this caveat.

E. Distributed Gaussian Processes

As mentioned in section I D, an inherent caveat of GP regression is its computational complexity: Training and naive prediction require $\mathcal{O}(N^3)$ steps. While the latter

can be pushed down to $\mathcal{O}(N^2)$ by caching the training set dependent matrix inverses during training, the former cannot be improved without additional approximations.

Multiple approaches exist. Here, we limit ourselves to the *product-of-expert*-type (PoE) method presented in [5] called *robust Bayesian Committee Machine* (rBCM), but note that more reliable versions of this scheme already exist, see [6].

a. Distributed Training All PoE methods approximate the full GP setting by assuming that the prior correlation matrix $k(\boldsymbol{X}, \boldsymbol{X})$ can be reasonably well approximated as *block diagonal matrix* with M blocks. Each block can then be associated with a subset of the entire training set $\mathcal{D}^{(k)} = \{\boldsymbol{X}^{(k)}, \boldsymbol{y}^{(k)}\}$ and the log marginal likelihood — our cost function— decomposes approximately into:

$$\log p(\boldsymbol{y} \mid \boldsymbol{\Theta}, \boldsymbol{X}) \approx \sum_{k=1}^{M} \log p_k(\boldsymbol{y}^{(k)} \mid \boldsymbol{\Theta}, \boldsymbol{X}^k) \; . \tag{27}$$

Evidently, the computation of each term is independent of all other terms and can therefore be evaluated by M processes, so-called experts, *simultaneously*.

Let $N_k \ll N$ denote the approximately equal number of data points per expert. Assuming sufficient resources, the time complexity of the distributed gaussian process training step then requires $\mathcal{O}(N_k^3) \ll \mathcal{O}(N^3)$ steps.

We stress although each expert operates on a private dataset, all experts *share the same hyperparameters*. A consequence of this is an intrinsic hyperparameter regularization compared to the full GP approach.

b. rBCM Aggregation While all PoE schemes share the same training stage, differences arise in the prediction step depending on how all expert results are aggregated into one overall prediction.

For simplicity, set $M = 2$ and only consider a single test point $\boldsymbol{x}'$. rBCM rewrites the posterior distribution over $f' \equiv f(\boldsymbol{x}')$ by application of Bayes theorem into:

$$p(f' \mid \boldsymbol{x}', \mathcal{D}^{(1)}, \mathcal{D}^{(2)}) \propto p(\mathcal{D}^{(1)}, \mathcal{D}^{(2)} \mid \boldsymbol{x}', f') p(f' \mid \boldsymbol{x}') \; , \tag{28}$$

and assumes *conditional independence* of both training sets, i.e.:

$$\begin{aligned} &p(f' \mid \boldsymbol{x}', \mathcal{D}^{(1)}, \mathcal{D}^{(2)}) \\ &\qquad \propto p(\mathcal{D}^{(1)} \mid \boldsymbol{x}', f') p(\mathcal{D}^{(2)} \mid \boldsymbol{x}', f') p(f' \mid \boldsymbol{x}') \; . \end{aligned} \tag{29}$$

Again applying Bayes' theorem to each likelihood yields a relation that associates the posterior distribution of the experts to the overall posterior distribution for f':

$$p(f' \mid \boldsymbol{x}', \mathcal{D}^{(1)}, \mathcal{D}^{(2)}) \propto \frac{p(f' \mid \boldsymbol{x}', \mathcal{D}^{(1)}) p(f' \mid \boldsymbol{x}', \mathcal{D}^{(2)})}{p(f' \mid \boldsymbol{x}')} \; . \tag{30}$$

For general M, one finds:

$$p\left(f' \mid \boldsymbol{x}', \mathcal{D} = \cup_k \mathcal{D}^{(k)}\right) \propto \frac{\prod_{k=1}^{M} p(f' \mid \boldsymbol{x}', \mathcal{D}^{(k)})}{p(f' \mid \boldsymbol{x}')^{M-1}} \; . \tag{31}$$

Since every expert is a GP, $p(f' \mid \boldsymbol{x}', \mathcal{D})$ remains gaussian and the multiplication identity, see eq. A.2, yields both the mean and variance of the univariate distribution:

$$\bar{f}(\boldsymbol{x}') = \sigma_f^2(\boldsymbol{x}') \sum_{k=1}^{M} \sigma_k^{-2}(\boldsymbol{x}') \bar{f}_k(\boldsymbol{x}') \; , \tag{32}$$

$$\sigma_f^{-2}(\boldsymbol{x}') = \sum_{k=1}^{M} \sigma_k^{-2}(\boldsymbol{x}') + (1 - M) k(\boldsymbol{x}', \boldsymbol{x}')^{-1} \; , \tag{33}$$

with $\bar{f}_k(\boldsymbol{x}')$ and $\sigma_k(\boldsymbol{x}')$ taken from eq. (10).

At last, rBCM suggests that the importance of each expert, denoted β_k, should not be uniform across all experts, i.e. $\beta_k \neq 1$, but should instead depend on how informative the posterior distribution of each expert is compared to its prior. A metric capable of quantifying this is the *difference in differential entropy* between both distributions:

$$\beta_k = \frac{1}{2} \left(\log k(\boldsymbol{x}', \boldsymbol{x}') - \log \sigma_k^2(\boldsymbol{x}') \right) \; . \tag{34}$$

The moments of the rBCM posterior distributions then read:

$$\bar{f}(\boldsymbol{x}') = \sigma_f^2(\boldsymbol{x}') \sum_{k=1}^{M} \beta_k(\boldsymbol{x}') \sigma_k^{-2}(\boldsymbol{x}') \bar{f}_k(\boldsymbol{x}') \; , \tag{35}$$

$$\sigma_f^{-2}(\boldsymbol{x}') = \sum_{k=1}^{M} \beta_k(\boldsymbol{x}') \sigma_k^{-2}(\boldsymbol{x}') + \left(1 - \sum_{k}^{M} \beta_k(\boldsymbol{x}')\right) k(\boldsymbol{x}', \boldsymbol{x}')^{-1} \tag{36}$$

We refer to section II for implementational details and section III D for a benchmark between vanilla GP and rBCM-based, distributed GP in terms of regression quality and runtime complexity.

II. IMPLEMENTATIONAL DETAILS

The code developed for this work can be found at the following -repo. This section is meant to make certain aspects of the implementation explicit. We start in section II A by summarizing the internals of `NoiseFittedGP`, a minor modification to `sklearn`'s `GaussianProcessRegressor` class. Section II B focuses on the details of the `DistributedGaussianProcessRegressor` class — an implementation of the rBCM GP outlined in section I E, while section II C explains our modifications to `sklearn`'s kernel collection used for the numerical study in section III.

A. `NoiseFittedGP`

The analysis of section III B is carried out with `NoiseFittedGP` — a derivative of `sklearn`'s GP implementation.

The difference between `NoiseFittedGP` and `GaussianProcessRegressor` lies in how we treat noise during the training and prediction stage.

While `GaussianProcessRegressor` treats noise as user provided, untunable parameter — the `alpha` parameter — `NoiseFittedGP` adds an isotropic `WhiteKernel` implicitly to whatever correlation function the user provides. This allows us to fit the noise contamination of the target vector $\boldsymbol{y}$ without postulating it. Once known, we precompute prediction relevant quantities with the noise augmented kernel but set its contribution to zero for the final prediction step so that the latent function and not future, noisy prediction are approximated. Note, this is consistent with eq. (11) and eq. (12) in which the noise only enters as a contribution to the training set dependent matrix inverse (a prediction pre-computation quantity).

We refer to `guassian_process_regression.py` for more details.

B. `DistributedGaussianProcessRegressor`

The `DistributedGaussianProcessRegressor` class implements the (r)BCM-aggregated PoE-distributed gaussian process presented in section IE. Its current form is a derivative of `sklearn`'s GP implementation `GaussianProcessRegressor`. The pythionic pseudo-code below showcases its coarse-grained structure. We refer to the implementation file `distributed_gaussian_process.py` for more information.

```
class dGP:
    def __init__(self, M=1, kernel, ...):
        # SKLEARN INIT

        self.kernel = self.kernel + WhiteKernel(
            noise_level=0.1)

    def fit(self, X, y):
        # Noise fitted by WhiteKernel
        self.alpha = 0
        # Reinit the internal kernel
        self.kernel_ = clone(self.kernel)

        self.set_training_data(X, y)

        # SKLEARN CODE FOR OPTIMIZATION

        # Precompution of per-expert quantities
        self.precomp_k = [
            precompute_fixed_quantities_expert_.
                remote(
                self.X_train_[k], self.y_train_[
                    k], self.kernel_
            )
            for k in range(self.M)
        ]

        # Always predict latent function
        self.alpha = self.kernel_.k2.noise_level
        self.kernel_.k2.noise_level = 0

        # Put optimized kernel into shared mem
        self.kernel_id = ray.put(self.kernel_)

    def log_marginal_likelihood(self, theta, **
        kwargs):
        # Put kernel into shared mem (no copies)
        kernel_id = ray.put(self.kernel_)

        # Concurrent cost function eval
        llm_k_values = ray.get([
                log_marginal_likelihood_expert_.
                    remote(
                    theta,
                    self.X_train_[k],
                    self.y_train_[k],
                    kernel_id,
                    self.alpha, # 0 (during fit)
                    **kwargs,
                )
                for k in range(self.M)
            ]
        )

        return sum(llm_k_values)

    def predict(self, X, **kwargs):

        if self.M == 1:
            # Standard GP Regression
            return ray.get(
                predict_expert_.remote(
                    X,
                    self.X_train_[0],
                    self.y_train_[0],
                    self.precomp_k[0],
                    self.kernel_id,
                    **kwargs
                )
            )

        # Concurrent per-expert prediction
        running = [
            predict_expert_.remote(
                X,
                self.X_train_[k],
                self.y_train_[k],
                self.precomp_k,
                self.kernel_id,
                return_std=True,
                return_cov=False,
            )
            for k in range(self.M)
        ]

        mu = np.zeros(X.shape[0])
        sigma2 = np.zeros(X.shape[0])
        beta = 0

        # On-line aggregation when
        # expert results become available
        while len(running):
            done, running = ray.wait(running)
            # aggregate mu, sigma, beta (eq. (35))
        if kwargs["return_std"]:
            return mu, np.sqrt(sigma2)
        else:
            return mu

    def set_training_data(self, X, y):
```

```
partition = np.array_split(np.arange(X.
    shape[0]), self.M)

self.X_train_ = []
self.y_train_ = []
for partition_k in partition:
    self.X_train_.append(ray.put(X[
        partition_k, :]))
    self.y_train_.append(ray.put(y[
        partition_k]))
```

Two points are worth highlighting:

Firstly, `DistributedGaussianProcessRegressor` adopts the same noise fit philosophy as `NoiseFittedGP`.

Secondly, closer inspection of the pseudo-code shows that the actual per-expert computations for the cost function evaluation, pre-computations and fitting are deferred to `log_marginal_likelihood_expert_`, `precompute_fixed_quantities_expert` and `predict_expert_` — all of which represent code chunks of the original GP implementation shipped with `sklearn`. Thus, the sole purpose of `DistributedGaussianProcessRegressor`, besides the noise fit explained above, is to (i) manage data usage between each expert via a shared memory, (ii) a concurrent dispatch of the expert-level functions that operate on a chunk of the shared memory and (iii) aggregation according to eq. (35). The former two points are achieved via `ray` — a parallel python framework, [7].

C. `shifty_kernels`

`shifty_kernels.py` contains a collection of kernel modifications shipped with `sklearn`. The name is deliberate as (i) minimal testing was done beyond what the numerical study of section III required and (ii) we break the API contract of `sklearn`'s kernel suite to accommodate for correlation functions that enforce symmetry as outlined in section I C 2 c.

More specifically, we ship variants of `sklearn`'s `ConstantKernel`, `WhiteKernel`, `RBF` and add eq. (22) and eq. (23) in form of a `ProjectionKernel` and `SymmetricKernel1D`.

The reason for the modification of existing correlation functions is that `sklearn` forbids the computation of kernel gradients wrt. hyperparameters $\boldsymbol{\Theta}$ for differing kernel arguments. It is only allowed to compute $\nabla_{\boldsymbol{\Theta}} k(\boldsymbol{X}, \boldsymbol{X})$ and not $\nabla_{\boldsymbol{\Theta}} k(\boldsymbol{X}, \boldsymbol{Y})$. Unfortunately, it is the latter case with $\boldsymbol{Y} = -\boldsymbol{X}$ that we need to implement eq. (23) and since in principle any one dimensional correlation function k is permissible in eq. (23) all kernel implementations that will be used in conjunction `SymmetricKernel1D` have to be adapted to allow for non-identical matrix arguments. The authors are not aware of any side effects that such an API change entails.

We also emphasize that neither the projection matrix $\boldsymbol{A}$ in eq. (22) nor the center of symmetry x_S in eq. (23) are tunable hyperparameters. While the former restriction is conceptionally sound, the latter is again a limitation of `sklearn`'s implementation. To understand this, revisit eq. (23). Taking the gradient wrt. x_S — a necessary step in computing the loss function gradient $\partial_{x_S} \log\left(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\Theta} = \left(\dots \; x_S \; \dots\right)\right)$ requires computing the derivative of a generic kernel k *wrt. its arguments and not the hyperparameters* — a consequence of the chain rule. This functionality has to be provided for *every* kernel that will be used in conjunction with `SymmetricKernel1D`, but is provided by *none* of the correlation functions included in `sklearn`. It is for this reason that we refrained from promoting x_S to be a trainable hyperparameter.

An interesting modification to this project would be a from-scratch implementation of the kernel suite in the autodifferentiation framework `jax`, see [8]. This would make any form of gradient computation — the reason for all our modifications and synthetic limitation — trivial.

III. NUMERICAL STUDY

We proceed by showcasing the capabilities and limitations of gaussian process regression, also called kriging, on a high dimensional, $D = 5$, synthetic and noisy dataset. Section III A introduces the dataset and infers qualitative properties from it. We use these insights to construct an optimized correlation function and benchmark it against plausible alternatives in section III B. Section III C focuses on the robustness of the GP-prediction and in particular how trustworthy the results are inside and far away from the training data. Finally, we showcase the capabilities of the rBCM-aggregated, distributed gaussian process in section III D.

A. Friedman Function

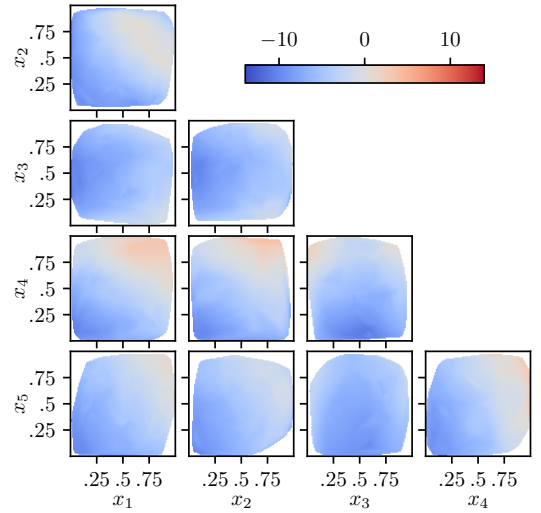
The latent function we attempt to model will be given by Friedman's function,

$$g(\boldsymbol{x}) \colon [0,1]^5 \to \mathbb{R} \tag{37}$$

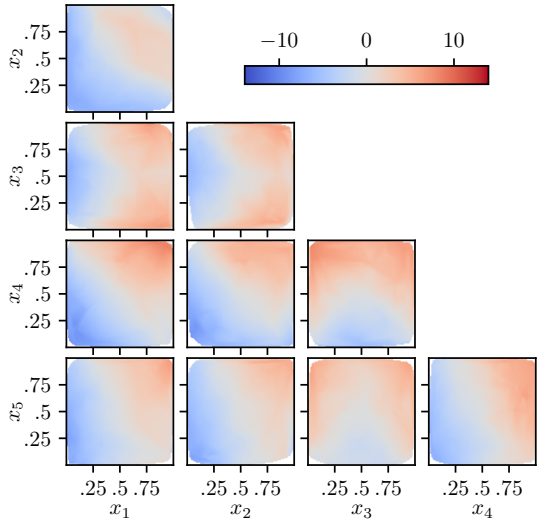
[9], a scalar function on a $D = 5$ dimensional unit hypercube domain. It is deliberate that we omit additional information of the known function. Instead, the task is to deduce its characteristics from a noisy training dataset and use only this information to construct an appropriate kernel for the GP-prior distribution.

To do so, and consistent with eq. (1), we add iid gaussian noise with arbitrary but fixed standard deviation $\sigma = 0.7$ to the latent function and sample the data generative process at $N = 700$ uniformly distributed points in the unit hypercube, so that $\boldsymbol{X} \in \mathbb{R}^{N \times D} \sim U([0,1]^5)$

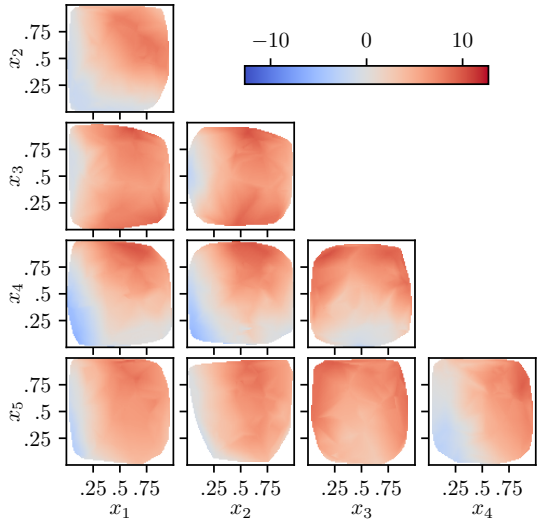
To get an insight into the properties of the dataset, we linearly interpolate between the noisy function values $\boldsymbol{y}(\boldsymbol{X}) \in \mathbb{R}^N$ and take $\binom{5}{2} = 10$ two-dimensional $x_i x_j$-cross sections through its convex hull. All remaining coordinates are fixed to $x_{\{1\dots5\}-\{i,j\}} = 0.25, 0.5, 075$. The result of this procedure is shown in Fig. 1.



(a) $x_{\{1\ldots5\}-\{i,j\}} = 0.25$



(b) $x_{\{1\ldots5\}-\{i,j\}} = 0.5$



(c) $x_{\{1\ldots5\}-\{i,j\}} = 0.75$

FIG. 1: Two dimensional cross sections through the noisy, linearly interpolated training data generated from Friedman's function.

Closer inspection allows for the following conclusions: (i) The function seems relatively smooth and not noise dominated, (ii) the x_4 and x_5 direction appear to be uncorrelated, i.e. if x_4 *or* x_5 is large so is y, (iii) function values in the x_1 and x_2 direction seem to be correlated, i.e. only if both x_1 *and* x_2 is large so is y and (iv) all cross section that involve the x_3 coordinate are symmetric around $x_3 = 0.5$.

As we will see in section III B this is sufficient information to construct a nearly ideal correlation function.

B. Kernel Comparison

The observations made in section III A translate naturally into a series of plausible correlation function candidates:

The smoothness and noise observation, suggest that all candidate correlation functions should be based on a RBF kernel, cf. eq. (16), and a constant, white noise, kernel through which we infer the noise level directly from the data. Since the presence of the latter is always implied during training by the internal workings of `DistributedGaussianProcessRegressor`, see II B, our first candidate kernel is:

$$\texttt{sq\_exp}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_A^2 \exp\left(-\frac{1}{2}\frac{|\boldsymbol{x} - \boldsymbol{x}'|^2}{l^2}\right) , \tag{38}$$

with l and σ_A^2 denoting the trainable length scale and amplitude respectively.

Notice eq. (38) assumes one common length scale l in radial direction and not a characteristic scale per-dimensional, $\boldsymbol{l}$. This might be overly restrictive. Thus we use the ARD RBF kernel of eq. (17) as second candidate correlation function:

$$\texttt{sq\_exp\_ard}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_A^2 \exp\left(-\frac{1}{2}|\left(\boldsymbol{x} - \boldsymbol{x}'\right)/\boldsymbol{l}|^2\right) . \tag{39}$$

To allow for independent length scales *and* and independent amplitude $\sigma_{A,i}^2$ for each dimension x_i, the sub-space projection kernel of eq. (22) can be employed. One finds:

$$\texttt{sq\_exp\_i12345}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{5} \texttt{sq\_exp}_i(\boldsymbol{e}_i\boldsymbol{e}_i^\intercal\boldsymbol{x}, \boldsymbol{e}_i\boldsymbol{e}_i^\intercal\boldsymbol{x}') , \tag{40}$$

with $(\boldsymbol{e}_i)_j = \delta_{ij}$ and each $\texttt{sq\_exp}_i$ being parametrized by its own length scale and amplitude $\{l_i, \sigma_{A,i}^2\}$.

Eq. (40) assumes independence of all dimensions — a consequence of the summation. To implement the observed coupling between the x_1 and x_2 direction we replace their addition with a multiplication which acts as a logical

AND operation:

$$\texttt{sq\_exp\_c12\_i345}(\boldsymbol{x}, \boldsymbol{x}') = \prod_{i=1}^{2} \texttt{sq\_exp}_i(\boldsymbol{e}_i\boldsymbol{e}_i^\intercal\boldsymbol{x}, \boldsymbol{e}_i\boldsymbol{e}_i^\intercal\boldsymbol{x}') + \sum_{i=3}^{5} \texttt{sq\_exp}_i(\boldsymbol{e}_i\boldsymbol{e}_i^\intercal\boldsymbol{x}, \boldsymbol{e}_i\boldsymbol{e}_i^\intercal\boldsymbol{x}') \,. \quad (41)$$

At last, we add symmetry for the x_3 direction around $x_3 = 0.5$, cf. eq. (23) to arrive at:

$$\texttt{sq\_exp\_c12\_s3\_i45}(\boldsymbol{x}, \boldsymbol{x}') = \prod_{i=1}^{2} \texttt{sq\_exp}_i(\boldsymbol{e}_i\boldsymbol{e}_i^\intercal\boldsymbol{x}, \boldsymbol{e}_i\boldsymbol{e}_i^\intercal\boldsymbol{x}') + k_S(\boldsymbol{e}_3\boldsymbol{e}_3^\intercal\boldsymbol{x}, \boldsymbol{e}_3\boldsymbol{e}_3^\intercal\boldsymbol{x}') + \sum_{i=4}^{5} \texttt{sq\_exp}_i(\boldsymbol{e}_i\boldsymbol{e}_i^\intercal\boldsymbol{x}, \boldsymbol{e}_i\boldsymbol{e}_i^\intercal\boldsymbol{x}') \,. \quad (42)$$

Eq. (38) - (42) comprise the collection of candidate kernels used to fit Friedman's function. Table I summarizes the number of hyperparameters per kernel and the observations in section III A that led to them. Note that the additional hyperparameter is given through the white noise kernel used at the training stage but absent during prediction.

Kernel	# Hyperparameters	Observation
`sq_exp`	2+1	(i)
`sq_exp_ard`	6+1	(ii)
`sq_exp_i12345`	10+1	(ii)
`sq_exp_c12_i45`	10+1	(iii)
`sq_exp_c12_s3_i45`	10+1	(iv)

TABLE I: Candidate Correlation functions. Additional hyperparameter through implicit white noise kernel.

We proceed by training five different GPs — one for each kernel — on the training data shown in Fig. 1 estimate the *expected generalization error* $\text{MSE} = \mathbb{E}[(\boldsymbol{y}' - \hat{g}(\boldsymbol{X}'))^2]$ through a ten-fold cross validation, see [2] for more information. The result of this procedure is an estimate for the mean square error $\hat{\text{MSE}} \approx \mathbb{E}[(\boldsymbol{y}' - \hat{g}(\boldsymbol{X}'))^2]$ for each candidate kernel which we show in Fig. 2.

Multiple observations can be made:

We first note that all estimated expected prediction errors are bounded from below by the irreducible noise $\sigma^2 = 0.7^2 = 0.49$ intrinsic to the training set. This is consistent with the *bias-variance decomposition* analyzed in [2].

Evidently, the most adapted kernel `sq_exp_c12_s3_i45` performs best, closely followed by the symmetry free but partially coupled `sq_exp_c12_i345` kernel and the simplistic RBF correlation function. By contrast, the most flexible kernels `sq_exp_ard` and `sq_exp_i12345` yield significantly deteriorated predictive performance.

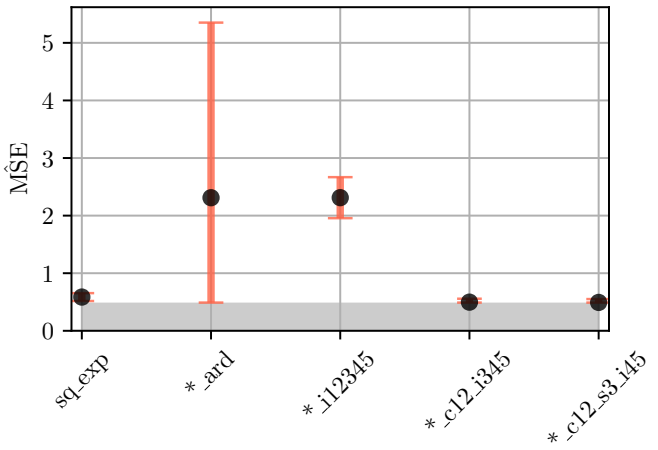


FIG. 2: Cross-validated estimate of the expected prediction error for all candidate kernels. The gray shaded error represents the irreducible noise $\sigma^2 = 0.7^2 = 0.49$ — according to the bias-variance decomposition the lower bound for the MSE, [2].

These results are plausible: Recall that the RBF correlation function is *universal* and thus capable of approximating any function given enough data. Since the Friedman function is rather well behaved without any erratic features it is reasonable to assume that $N = 700$ data points are already sufficient to arrive in this regime.

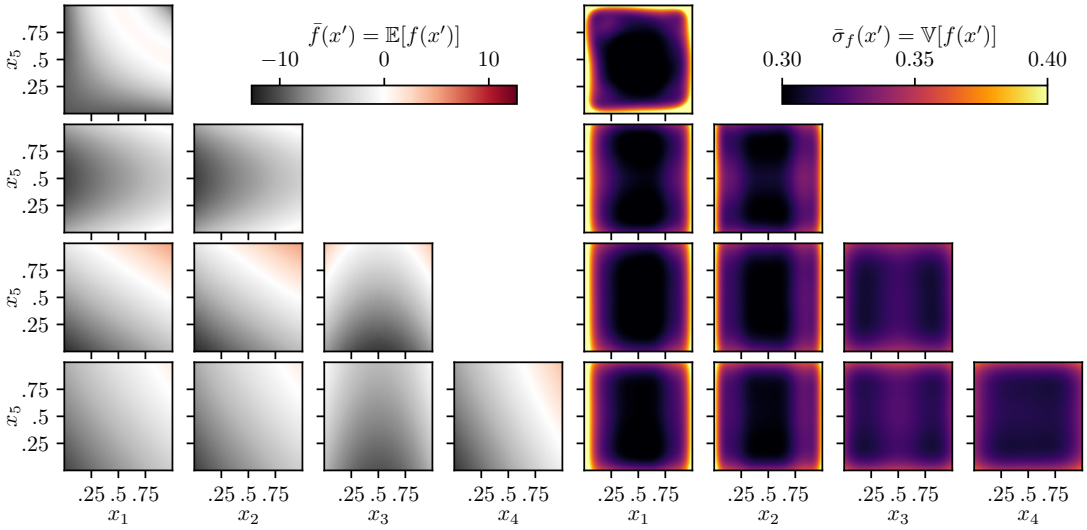
The high number of hyperparameters for `sq_exp_ard` and `sq_exp_i12345`, on the other hand, has the strong potential for parameter degeneracies and a highly complicated cost function surface including many local minima. The relatively large error for both kernels supports this hypothesis: Depending on the random initial conditions training got stuck in various local minima, but did not arrive at the global minimizer, even after restarting each training ten times in this experiment.

Only the addition of structural constraints such as dimension coupling and symmetrization seems to alleviate this problem, yielding kernels that outperform the vanilla RBF correlation function.

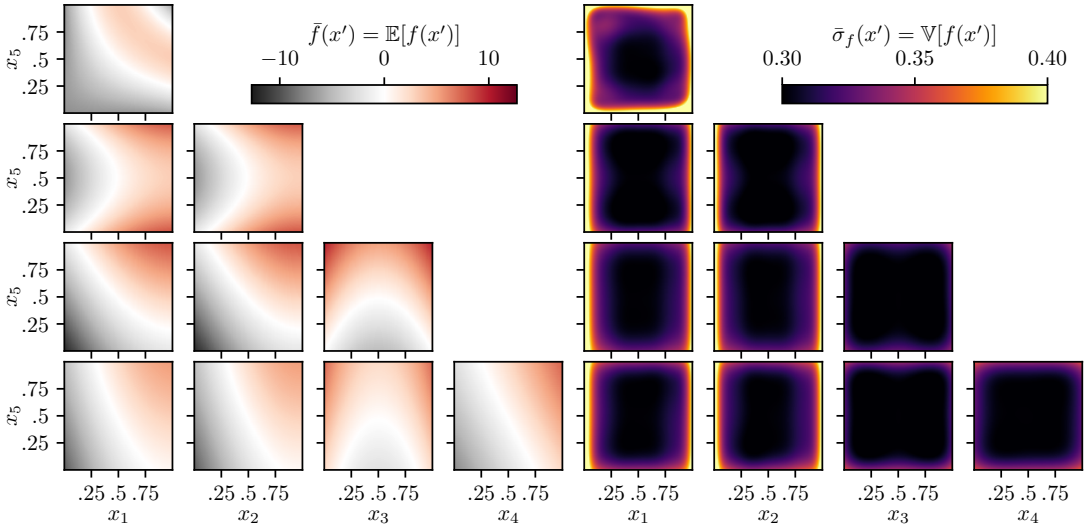
We close this section by reproducing the cross sections of Fig. 1 obtained from the best performing `sq_exp_c12_s3_i45`-GP for both the mean prediction, eq. (11), and variance, eq. (12), in Fig. 3.

C. Overconfindence, Interpolation & Extrapolation

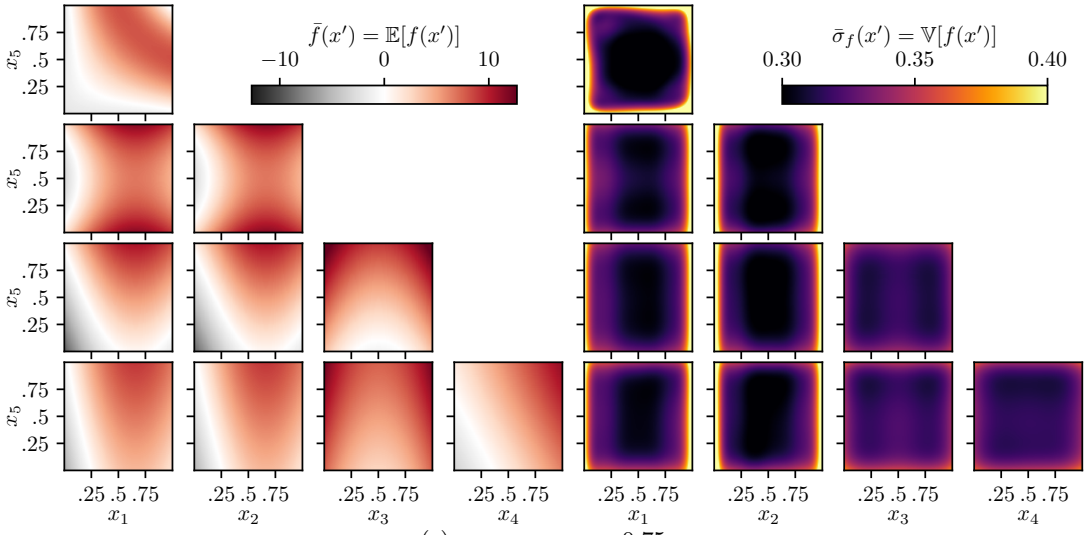
As elaborated in the introduction the fact that GPs are formulated in the context of Bayesian probability gives us access to not just a single function, but rather a Gaussian distribution over functions. This can help us to further understand whether the confidence estimated by our model is appropriate.



(a) $x_{\{1\ldots5\}-\{i,j\}} = 0.25$



(b) $x_{\{1\ldots5\}-\{i,j\}} = 0.5$



(c) $x_{\{1\ldots5\}-\{i,j\}} = 0.75$

FIG. 3: Two dimensional cross sections of the mean function $\bar{f}(\boldsymbol{x}')$ and variance function $\sigma(\boldsymbol{x}')$ deduced from the trained `sq_exp_c12_s3_i45`-GP.

In this section, we explore the question of how GPs generalize to unseen data. There are two ways to understand this question, the first being whether the functions that are drawn from the GP accurately predict the target function in the vicinity of the training points. We will refer to this type of generalization as being in an interpolative regime. The other way to interpret this question is to ask whether the GP is able to predict the function far away from the training points. We will refer to this type of generalization as being in an extrapolative regime. More formally, we use the following definition:

Definition III.1. Given a set of training points $\boldsymbol{X}$, a set of test points $\boldsymbol{X}'$ and the convex hull $\tau(\boldsymbol{X})$ of the points of $\boldsymbol{X}$:

- For all $\boldsymbol{x}'_i \in \tau(\boldsymbol{X})$, we say that $f(\boldsymbol{x}'_i)$ is in the interpolative regime.
- For all $\boldsymbol{x}'_i \notin \tau(\boldsymbol{X})$, we say that $f(\boldsymbol{x}'_i)$ is in the extrapolative regime.

Given this distinction, we can start to study the overconfidence of our GP in these two settings. To do that one more ingredient is necessary, namely a function which measures the notion of overconfidence.
We propose that one can say that a GP is overconfident at a point $\boldsymbol{x}'$ if the variance $k(\boldsymbol{x}', \boldsymbol{x}')$ at that point is smaller than the residual error $R(\boldsymbol{x}') = ||f(\boldsymbol{x}') - y(\boldsymbol{x}')||_2$ (in our case the l_2 norm is equivalent to taking the absolute value, since the image of the Friedman function is 1 dimensional).
In order to observe the performance of the GP in these two regimes, we had to restrict our training set to a subset of the full 5-dimensional hypercube. To do this, we used Latin Hypercube sampling restricted to the hypercube $[0.25, 0.75]^5$ and sampled the Friedman function Eq. 37 with an added noise of $\sigma = 0.3$. With this approach points in the corners of the larger hypercube do not appear in the training set. As a result testing on samples from these corners and evaluating the aforementioned measure tells us what the extrapolative overconfidence of the GP is.
The results shown in Fig. 4 paint a clear picture, the variance of the model quickly grows away from the training points. This outcome is expected as predictions far away from the training data are dominated by the prior and there is no apriori reason for this prior to match the predicted function. Furthermore in those regions, the error is also larger than the variance, so we can safely say that the model is overconfident in the extrapolative regime. The more surprising fact is that while the variance is lower within the convex hull of the training set, there are still many points on which the GP is overconfident.

To make this point more clear in Fig. 5, we show histograms of the overconfidence in both the interpolation and extrapolation regimes for the same cross-sections as in Fig. 4. From this plot, we confirm that while overconfidence is on average higher in the extrapolation regime, the GP also seems to be overconfident in the interpolation regime. We propose that this could be due to insufficient sampling of the smaller hypercube as it is a 5 dimensional function which is very hard to sample well and such high sampling regimes are not efficiently computable within the scope of this work. However we also note that this result is still surprising, especially given the seemingly well behaved nature of the Friedman function.

To illustrate that given inappropriate sampling, high overconfidence can occur in the interpolation regime, we consider a much simpler one dimensional function, namely a Legendre polynomial of order 35. The outcome (shown in Fig. 6) illustrates that for insufficient sampling overconfidence occurs in both the interpolative and extrapolative regimes. This is not the case when the domain of the function is sufficiently sampled, in which case overconfidence is a problem only in the extrapolation regime.

From this, we conclude that, without knowing a very data specific prior (which would defeat the purpose of fitting a GP in the first place), it is impossible to generalize outside of the convex hull of the training set. Furthermore, we saw that even doing interpolation can be quite difficult for high dimensional functions as it can require very dense sampling of a functions' domain.

D. Distributed Gaussian Processes

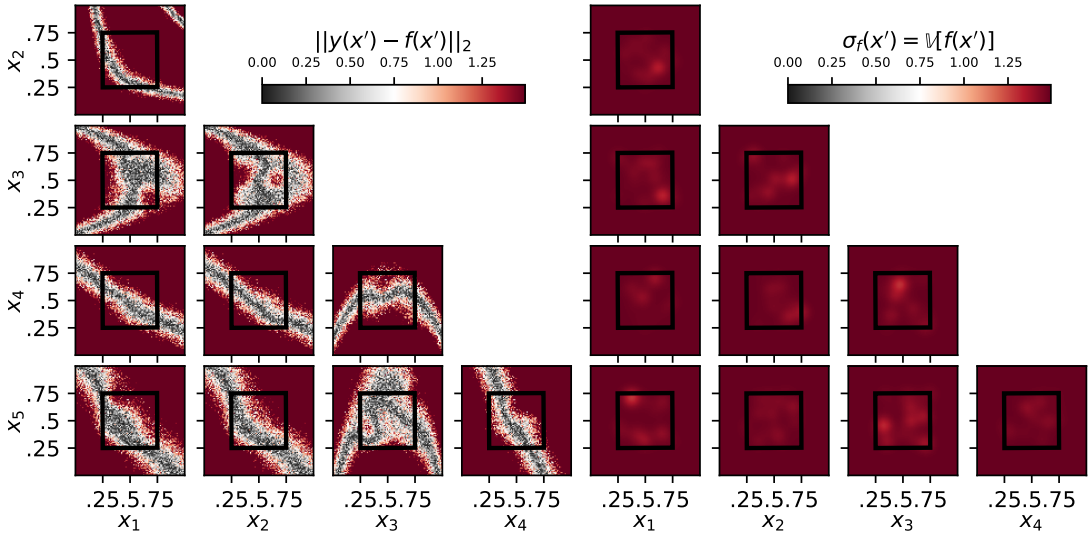
At last, we turn to the distributed GP approach and consider whether the PoE-style training and rBCM-like prediction alleviate the untractable cubic time complexity of standard GP.

Consider Fig. 7 which illustrates the wallclock time for the evaluation of the log marginal likelihood — the most expensive function in the training stage — as a function of training size N for both full GP (`NoiseFittedGP`) and the distributed scheme (`dGP`) and $M = 10$ experts.

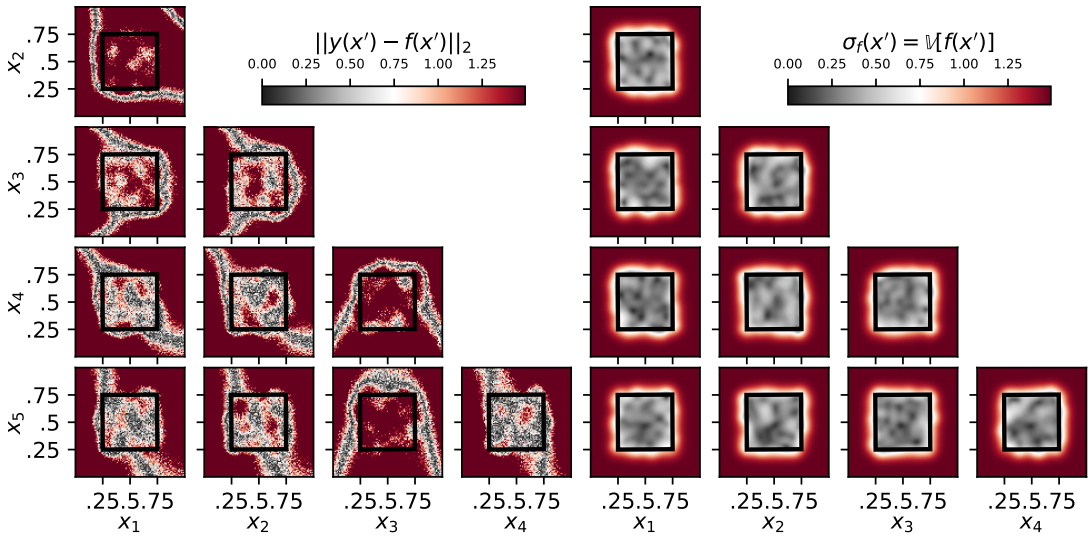
One finds a considerable speed up for `dGP` past $N > 200$ compared to full GP. For smaller problem sizes, runtime overhead dominates. Furthermore, we observe, as expected, the same time scaling between `dGP` and `NoiseFittedGP`, but different from $\mathcal{O}(N^3)$. A simplistic explanation for this might be that the range of considered problem sizes is not large enough to probe the asymptotic scaling regime. Going the larger N, however, is out of the scope of this work, due to runtime constraints for `NoiseFittedGP` and memory constraints in case of `dGP`.

We close by retraining our adapted kernel `sq_exp_c12_s3_i45` with the same training set as used in section III B to check the consistency with the full GP regression results.

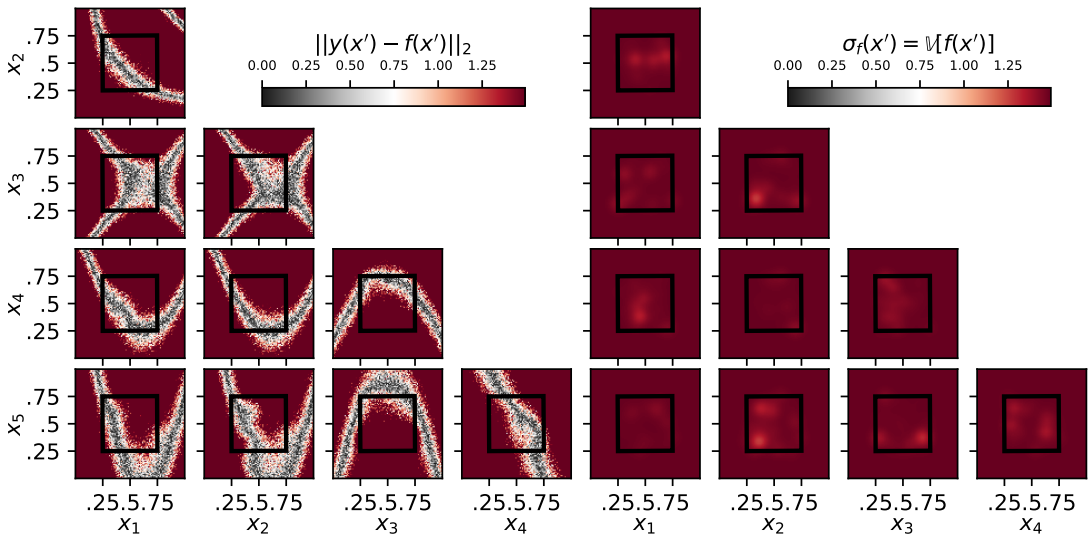
A cross-validation analysis yields an estimated expected prediction error of $\hat{\text{MSE}} = 0.54^{+0.09}_{-0.05}$ — again close to the irreducible noise in the training set, and only insignificantly worse than the result obtained with `NoiseFittedGP`. We deem this as a sufficient to conclude that (i) the implementation is competitive with full GP in terms of accuracy while alleviating its time complexity limitations to some extent.



(a) $x_{\{1\ldots5\}-\{i,j\}} = 0.2$



(b) $x_{\{1\ldots5\}-\{i,j\}} = 0.5$



(c) $x_{\{1\ldots5\}-\{i,j\}} = 0.8$

FIG. 4: Two dimensional cross sections of the error function $||y(x') - f(x')||_2$ and variance function $\sigma(\boldsymbol{x}')$ deduced from a trained `sq_exp`-GP.

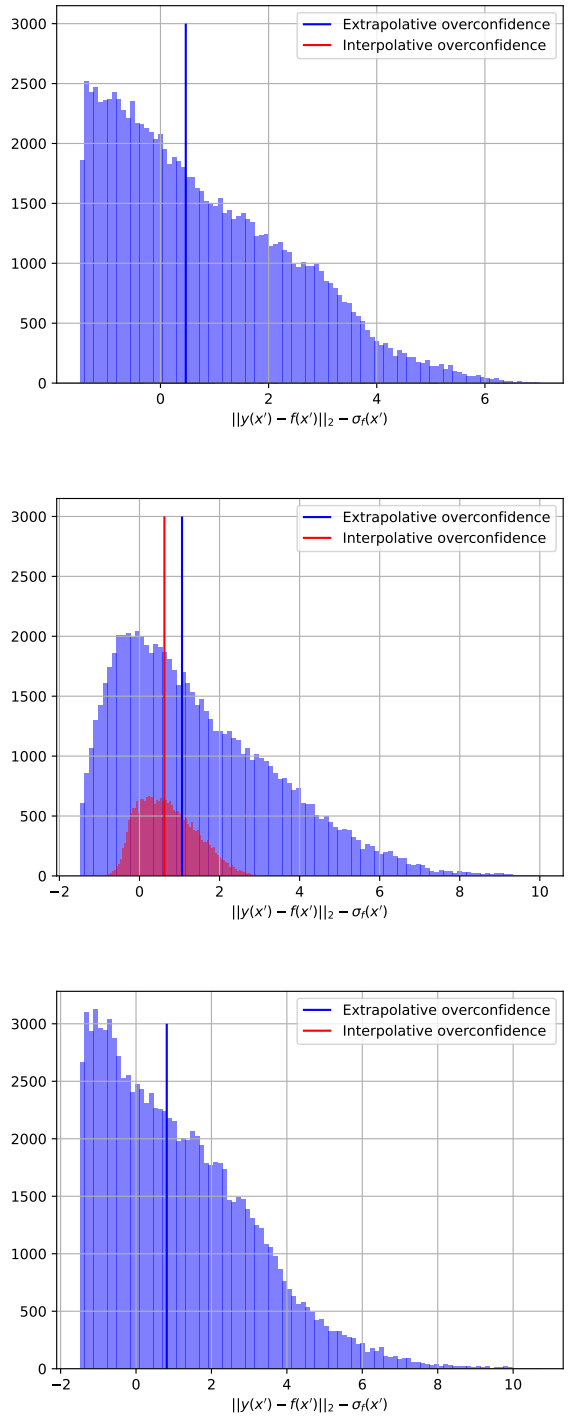


FIG. 5: Histograms + medians of the overconfidence for all three cross-sections. Positive values indicate overconfidence. Keep in mind that cross-sections outside of $[0.25, 0.75]$, do not contain any points that would fall in the interpolation regime.

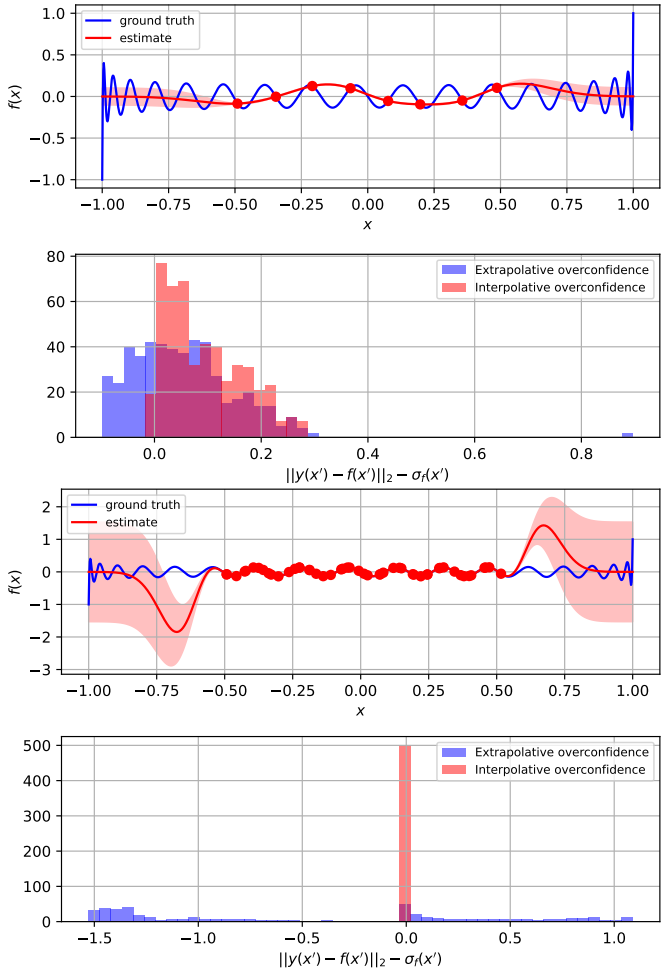


FIG. 6: Gaussian process prediction for Legendre polynomials. 8 training points (top) and 50 training points (bottom).

The reader is referred to Fig. 8 for a visualization of the regression result analogous to Fig. 3.

IV. CONCLUSION

To summarize we have analyzed the performance of Gaussian Process regression on the high-dimensional Friedman function, by introducing custom kernels that slightly outperform the widely used RBF kernel. However, we concluded that, potentially due to its universality properties, the RBF kernel is still appropriate to use and has the significant advantage of leading to noticeably faster computations.

Additionally, we analyzed the overconfidence and generalization of GPs, by restricting the training set to a subspace of the full domain of the predicted function. We noticed that overconfidence has to occur when the GP is the regime of extrapolation. On the other hand, the overconfidence in interpolation is heavily dependent on the density with which the domain of a function is

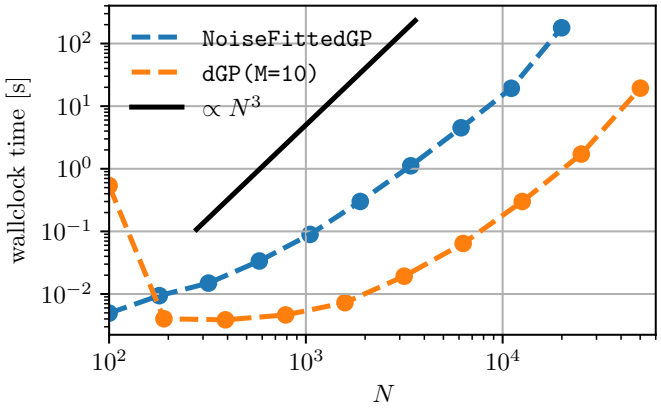


FIG. 7: Comparison of the wallclock runtime of the cost function evaluation — the log marginal likelihood —between standard GP and `DistributedGaussianProcessRegressor` with $M = 10$ experts. While for the standard method prevails at very small datasets where process switching and memory management overhead dominate, `DistributedGaussianProcessRegressor` outperforms the vanilla GP implementation past $N = 200$.

sampled.
Finally, we implemented the significantly faster Distributed Gaussian Process approach. We further showed that our implementation matched the performance of the widely used Sklear `GaussianProcessRegressor`, while providing a significant speed up.

ACKNOWLEDGMENTS

We thank Anders Kvellestad for the exceptional conceptional clarity of the third-cycle CompSci lecture series.

Appendix: Gaussian Distribution Identities

Let

$$\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-D/2}|\boldsymbol{\Sigma}|^{-1/2}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\intercal\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{A.1}$$

denote the multivariate gaussian distribution over $\boldsymbol{x} \in \mathbb{R}^D$ with mean $\boldsymbol{\mu} \in \mathbb{R}^D$ and positive-semidefinite, invertible covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times D}$. Then the following properties hold:

a. Multiplication :

$$\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}', \boldsymbol{\Sigma}') \propto \mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{\Sigma}''(\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{\Sigma}'^{-1}\boldsymbol{\mu}'), \underbrace{\left(\boldsymbol{\Sigma}^{-1} + \boldsymbol{\Sigma}'^{-1}\right)^{-1}}_{\boldsymbol{\Sigma}''}\right) \tag{A.2}$$

b. Conditioning/Marginalization : If:

$$\boldsymbol{x}, \boldsymbol{y} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_{\boldsymbol{x}} \\ \boldsymbol{\mu}_{\boldsymbol{y}}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{\boldsymbol{xx}} & \boldsymbol{\Sigma}_{\boldsymbol{xy}} \\ \boldsymbol{\Sigma}_{\boldsymbol{xy}}^\intercal & \boldsymbol{\Sigma}_{\boldsymbol{yy}}\end{bmatrix}\right) \tag{A.3}$$

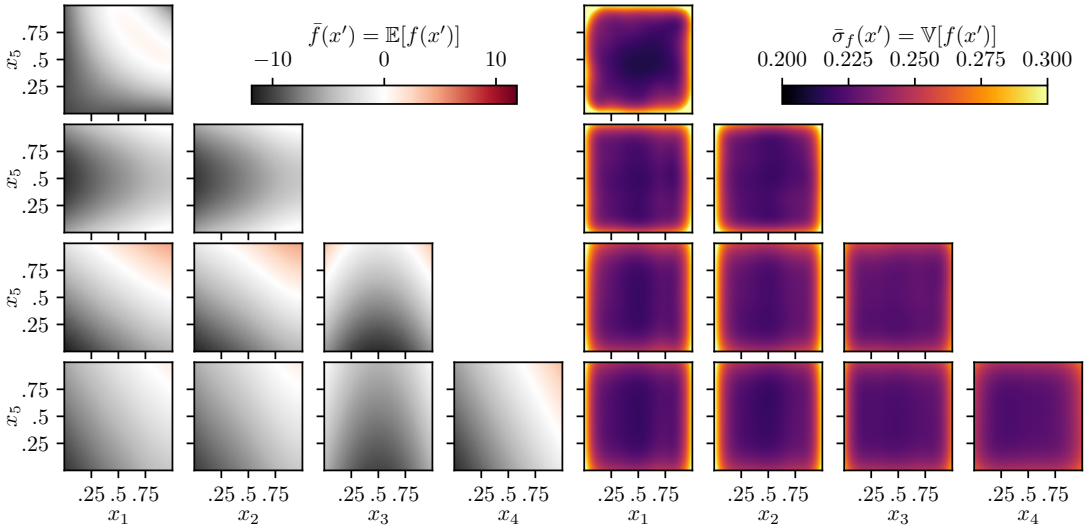
then the *conditioned* distribution is:

$$\boldsymbol{x} \mid \boldsymbol{y} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{x}} + \boldsymbol{\Sigma}_{\boldsymbol{xy}}\boldsymbol{\Sigma}_{\boldsymbol{yy}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}}), \boldsymbol{\Sigma}_{\boldsymbol{xx}} - \boldsymbol{\Sigma}_{\boldsymbol{xy}}\boldsymbol{\Sigma}_{\boldsymbol{yy}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{xy}}^\intercal\right) \tag{A.4}$$

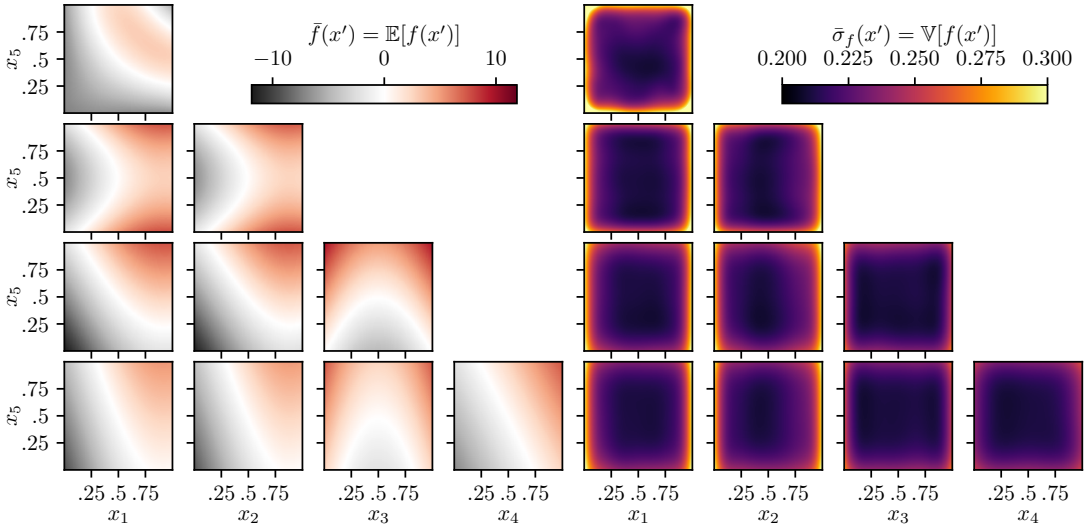
and the *marginalized* distribution reads:

$$\boldsymbol{x} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{x}}, \boldsymbol{\Sigma}_{\boldsymbol{xx}}\right) . \tag{A.5}$$

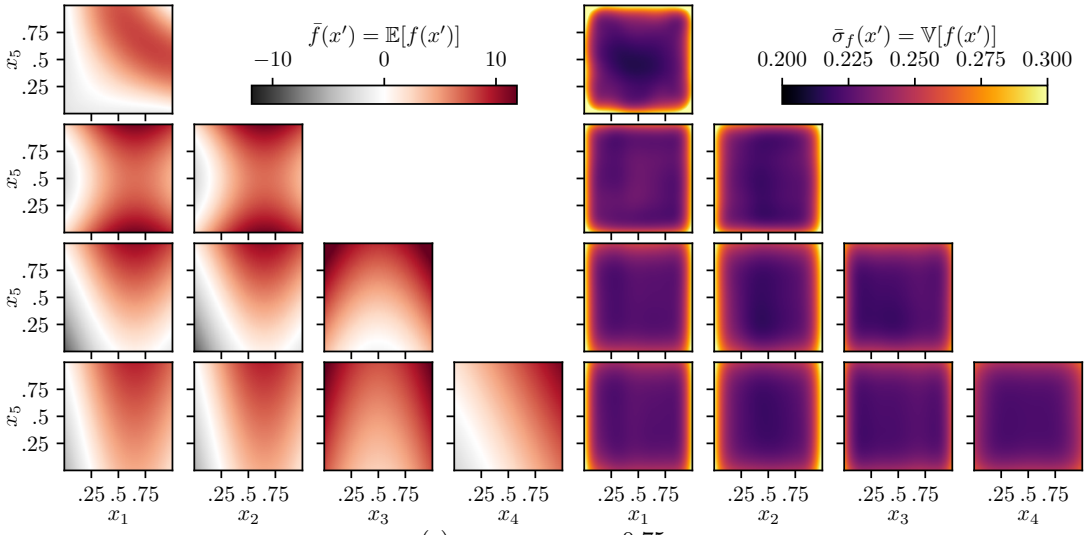
[1] C. E. Rasmussen, *Gaussian processes for machine learning* (MIT Press, 2006).
[2] K. Beshkov, M. Safy, and T. Zimmermann, "CompSci Project 1:Regression Analysis and Resampling Methods," (2022), availiable on request.
[3] D. Duvenaud, *Automatic Model Construction with Gaussian Processes*, Ph.D. thesis, University of Cambridge (2014).
[4] C. A. Micchelli, Y. Xu, and H. Zhang, J. Mach. Learn. Res. **7**, 2651–2667 (2006).
[5] M. P. Deisenroth and J. W. Ng, JMLR W&CP, vol 37, 2015 (2015), arXiv:1502.02843 [stat.ML].
[6] H. Liu, J. Cai, Y. Wang, and Y.-S. Ong, (2018), arXiv:1806.00720 [stat.ML].
[7] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, (2017), arXiv:1712.05889 [cs.DC].
[8] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," (2018).
[9] S. Surjanovic and D. Bingham, "Virtual library of simulation experiments: Test functions and datasets," Retrieved May 29, 2022, from `http://www.sfu.ca/~ssurjano` (2013).



(a) $x_{\{1\ldots5\}-\{i,j\}} = 0.25$



(b) $x_{\{1\ldots5\}-\{i,j\}} = 0.5$



(c) $x_{\{1\ldots5\}-\{i,j\}} = 0.75$

FIG. 8: Two dimensional cross sections of the mean function $\bar{f}(\boldsymbol{x}')$ and variance function $\sigma(\boldsymbol{x}')$ deduced from the trained `sq_exp_c12_s3_i45`-distributed GP using rBCM aggregation a.