Empirical evaluation of Gaussian Process approximation algorithms

Krzysztof Chalupka

Master of Science
School of Informatics
University of Edinburgh

2011

Abstract

Many large datasets are becoming available as technology advances. The fast development of the Internet makes creating huge databases of interesting data more feasible than ever; similarly, scientific simulations on modern computers can produce large amounts of information that needs to be analyzed. Machine Learning uses methods developed in Computer Science and Mathematics to deal with challenges posed in the context of such large scale data analysis. As the Bayesian framework became more popular, many flexible and theoretically elegant methods have been developed in the field. One such Bayesian framework uses Gaussian Processes to perform the two basic Machine Learning tasks, *regression* and *classification*. As it turns out, regression with Gaussian Processes is particularly elegant and analytically tractable. However, it scales badly with the size of the dataset which makes it infeasible for use in most interesting situations. Several *approximation algorithms* were developed to deal with this issue. While there were attempts to analyze and compare these approximations theoretically, not much has been done to present an unbiased and useful empirical evaluation of the algorithms. In this dissertation we create a solid framework for such comparison and perform experiments that allow us to analyze the practical usefulness of Gaussian Process approximation algorithms.

Acknowledgements

I am very grateful to Chris Williams for being the most helpful and patient mentor– him guiding me through this project has been a truly educative experience. I thank Iain Murray for many useful ideas and critical comments without which my understanding of the issues discussed here would be much narrower. I am also indebted to Andreas Krause who first sparked my interest in Machine Learning through his enthusiastic attitude towards research. Finally, none of this work would happen without the loving support of my mother- dziekuje za wszystko, mamo.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Krzysztof Chalupka*)

Table of Contents

Chapter 1

Introduction

This short introductory chapter serves to motivate our work (Section 1.1) and present related research that we are aware of (Section 1.2). We also outline the rest of this dissertation in Section 1.3.

1.1 Motivation

The field of Machine Learning (ML) is concerned with creating methods that extract useful information from data. The often encountered need to deal with with large datasets suggests using statistical methods, and such *statistical inference* has been perhaps the predominant paradigm in the field in recent years[1] . In particular, *Bayesian methods* have grown in popularity due to their great flexibility and theoretical optimality (see Bishop [2006], a popular recent book investigating Bayesian methods in some depth and significant breadth). In this thesis we look at one Bayesian framework, based on the *Gaussian Process* (GP) formalism, in more detail. Gaussian Process algorithms can be useful in many different settings. The most obvious application is in the offline *regression* problem, which we describe in much detail in 2; but the flexible nature of Bayesian algorithms in general and Gaussian Processes in particular makes them appear in such contexts as sensor placement (Krause [2005]) and experimental design (Srinivas et al. [2010]), in approximating stochastic differential equations (Archambeau et al. [2007]) or incorporated into Dirichlet Processes (Jackson et al. [2007]).

We will first explain how and why Gaussian Process algorithms can be useful and then

[1] Vapnik [1999] provides an overview of the theory behind statistical learning.

proceed to investigate the main point of concern: their space and time complexity. It turns out that Gaussian Processes suggest a useful but slow class of algorithms. The Machine Learning community has developed numerous approximation algorithms that try to lighten these computational requirements. While many publications investigate interesting theoretical properties of GP approximations , a fair *practical* comparison of their usefulness has not been published to our knowledge. This is not a satisfactory state of matters. After all, the goal of introducing an approximation algorithm is to make another method practically useful; hence it would be very useful to explicitly state how the new approximation's performance relates to that of other, already existing methods. In this dissertation we develop a clear and practical way of evaluating the approximations' usefulness and demonstrate experimentally which method, and why, the potential user should choose for their particular task.

1.2 Related Work

Throughout this thesis we will refer to many publications introducing new GP approximation algorithms. Very often these papers evaluate the new methods' performances in ways that are hard to compare explicitly. More research has been done to look at the theoretical properties of the different GP approximations. We refer to some of these publications extensively throughout our work. The list below gives short overview of some of these papers.

- Quinonero-Candela and Rasmussen [2005] provides a unifying framework that shows how several seemingly different approximations can be represented using very similar formalisms.
- Snelson [2001] is a PhD thesis introducing several related methods (FI(T)C, PI(T)C and Warped GP - we will encounter each of them in this dissertation). It also provides interesting insights into the workings of several other algorithms and presents a limited practical comparison of the different flavors of FITC that Snelson [2001] introduces.
- Quinonero-Candela et al. [2007] is similar to Quinonero-Candela and Rasmussen [2005] but wider in scope, as it mentions more methods and looks both at regression and classification settings.

1.3 Outline of the dissertation

Our ultimate goal is to understand which GP approximation, and why, is best used in a particular setting. Even though our evaluation focuses on empirical performance, some theoretical background is necessary to develop and understanding of the issues involved both in creating the evaluative framework and analyzing the results.

- *Chapter 2* develops the basic Machine Learning background and relates one of the simplest ML algorithms, linear regression, to Gaussian Process regression. It also looks at GPs as algorithms working in the *function space*. Understanding this material is important, as the mathematical formulation of GP inference shows explicitly how the algorithm scales badly with growing amounts of data.
- In *Chapter 3* we describe four Gaussian Process approximation algorithms that we feel would best represent the different kinds of approximations currently available.
- *Chapter 4* elaborates on the implementation phase of this project, in which we gathered already-existing code and modified it for our purposes, as well as put significant amounts of work into implementing some algorithms ourselves. Importantly, we also elaborate on the *practical complexity* of the algorithms in some detail; as it turns out, the practical differences in different algorithms' runtime and space requirements can be significant, which is not visible during theoretical, asymptotic analysis.
- We then decide on the framework for empirical algorithm performance comparison in *Chapter 5*. Several alternative approaches to the problem are discussed in this chapter. The choices that needed to be made include what datasets to test on and how to quantify the performance of the algorithms in a way general enough to enable us to compare the different approximations fairly.
- *Chapter 6* presents the practical results of our work: we show, from several different points of view, how the algorithms perform in practice. Discussing these practical results encourages tackling some interesting questions about the nature of the algorithms. Finally, in this chapter we explain which approximations can work best given a particular task.
- *Chapter 7* concludes this thesis and presents possible future directions for related research. It also mentions several GP approximations we do not analyze in this

dissertation and relates them to our results.

In addition, we provide two Appendices that should make our work more useful for any interested party. *Appendix A* lists all the *hyperparameter tables*, one important output of our tests (the concept of hyperparameters in the context of GP algorithms is explained in Chapter 2). *Appendix B* presents a guide to our codebase, which should make reproducing our results easy for anyone interested in using the code we have written.

Chapter 2

Theoretical Background

This chapter presents the basic theory needed to understand the remaining parts of the dissertation. We do not assume the reader is familiar with machine learning theory, but some knowledge of probability theory and statistics, basic linear algebra, and calculus is necessary to follow the discussion below.

Sections 2.1 and 2.2 introduce the problem of regression and perhaps the most basic regression model, *linear regression*. This model is then expanded using the *kernel trick* to introduce Gaussian Process Regression (GPR) in Section 2.3. The latter section also derives the GPR model from a different point of view, providing a wider perspective for understanding the material. We complete Section 2.3 and the chapter by looking in some detail at different phases of working with the model in the Bayesian framework: training the parameters and hyperparameters and producing posterior predictions.

2.1 Regression

In Machine Learning, the *regression problem* is to predict values of a function given only a limited amount of information about it. More formally, given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ of d-dimensional *input points* $\mathbf{x}_i$ and scalar *output values* $y_i = f(\mathbf{x}_i) + \varepsilon_i$ (where ε_i is a noise term), the goal is to predict the value of $f(\mathbf{x}_*)$ for some *test point* $\mathbf{x}_*$ not in the input set.

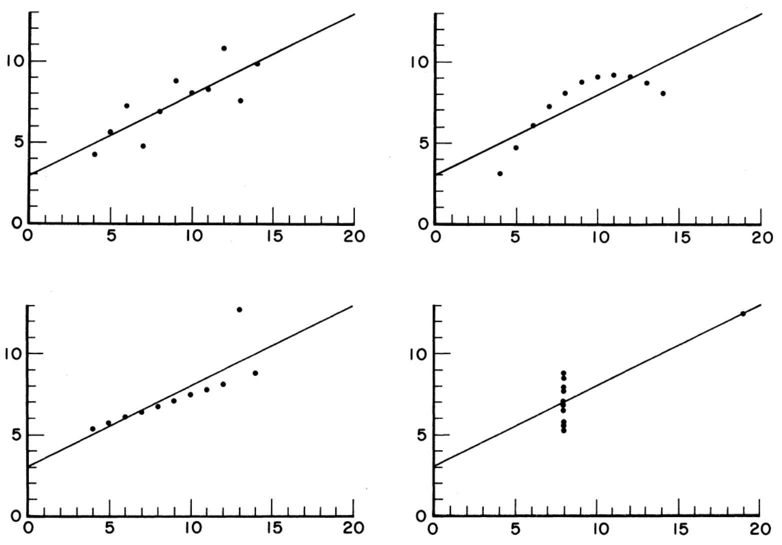


Figure 2.1: Even though the four simple datasets shown above have very distinct characteristics (linear outputs with Gaussian noise, smooth nonlinear outputs, linear outputs with little noise and an outlier, strongly clustered outputs with an outlier), all of them have the same linear best squared error fit. Linear regression seems to work best in the top left example, where the data follows the assumption of a linear relationship between the inputs and the outputs, with added Gaussian noise. Figure adapted from Anscombe [1973].

2.2 Linear Regression

Regression algorithms work by making assumptions about the function f and searching a *hypothesis space* - the space of all functions fulfilling these assumptions - for the best fit to the data. In linear regression only functions of form $f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + w_0$ ($\mathbf{w} \in \mathbb{R}^d$ and w_0 is a *bias term*) are considered. The task reduces to finding the best value of the parameters $\mathbf{w}, w_0$, often by minimizing some loss function such as Mean Squared Error $MSE = \left\langle (y_i - \mathbf{x}_i^T\mathbf{w})^2 \right\rangle$ (triangular brackets indicate the average in this dissertation. Usually it should be clear which vales we average over, like the $(y_i, \mathbf{x}_i)$ tuples in this case). Linear regression is very simple and easily interpretable, but has little expressive power. Figure 2.1 shows four very different datasets that all have the same best linear fit. As evident, much information about the structure of the data can be lost when using simple regression techniques. The first step we will take to define more interesting regression algorithms is to look at linear regression from a probabilistic standpoint.

If we assume that the noise term is Gaussian and the same for every i, so that $y_i = f(\mathbf{x}_i) + \varepsilon, \varepsilon \sim \mathcal{N}(0, \sigma_n^2)$, we get a *likelihood* model of the data. Defining a *prior* over the parameters $\mathbf{w}$, for example $p(\mathbf{w}) \sim \mathcal{N}(0, \Sigma_{par})$ where Σ_{par} is the prior covariance matrix, completes a probabilistic Bayesian model for linear regression. Rasmussen and Williams [2006] shows that under this model, the posterior over the parameters is again Gaussian,

$$p(\mathbf{w}|X,\mathbf{y}) \sim \mathcal{N}(\frac{1}{\sigma_n^2}A^{-1}X\mathbf{y}, A^{-1}), \tag{2.1}$$

where $A = \sigma_n^{-2}XX^T + \Sigma_{par}^{-1}$. We see that the Bayesian model yields a probabilistic distribution over the output hypotheses. In particular, for a test input $\mathbf{x}_*$ the estimated output value f_* is shown to be

$$p(f_*|\mathbf{x}_*, X, \mathbf{y}) \sim \mathcal{N}(\frac{1}{\sigma_n^2}\mathbf{x}_*^T A^{-1}X\mathbf{y}, \mathbf{x}_*^T A^{-1}\mathbf{x}_*). \tag{2.2}$$

Such predictive distribution can be very useful and makes Bayesian algorithms particularly attractive. Gaussian Process Regression is a fully Bayesian algorithm which can be seen as Bayesian linear regression with nonlinear basis functions. The next sections discuss this idea in more detail. For more on Bayesian linear models in general, see Bishop [2006].

2.3 Gaussian Process Regression

2.3.1 Extending linear regression

We can add nonlinearity to the Bayesian linear model by redefining $f(\mathbf{x}) = \phi(\mathbf{x})^T\mathbf{w}$ where $\phi : \mathbb{R}^d \to \mathbb{R}^D$ is a nonlinear *basis function* mapping the inputs into a *feature space* $\mathbb{R}^D$. This model is otherwise the same as Bayesian linear regression (if we make Gaussian assumptions on noise and the parameters) and as shown in Rasmussen and Williams [2006], Equation 2.2 now becomes

$$\begin{aligned} p(f_*|\mathbf{x}_*, X, \mathbf{y}) \quad \sim \quad & \mathcal{N}(\phi_*^T \Sigma_{par}\Phi(K+\sigma_n^2 I)^{-1}\mathbf{y}, \\ & \phi_*^T\Sigma_{par}\phi_* - \phi_*^T\Sigma_{par}\Phi(K+\sigma_n^2 I)^{-1}\Phi^T\Sigma_{par}\phi_*). \end{aligned} \tag{2.3}$$

Above, we use the notation $\phi_* = \phi(x_*), \Phi = \Phi(X), A = \sigma_n^2\Phi\Phi^T + \Sigma_{par}^{-1}, K = \Phi^T\Sigma_{par}\Phi$. This is same as equation 2.2 with every $\mathbf{x}$ substituted with $\phi(\mathbf{x})$, rewritten in a way that suggests the possibility of using the *kernel trick* (Scholkopf and Smola [2002]).

Because Σ_{par}, as a covariance matrix, is positive definite, we can always find its square root $\Sigma_{par}^{1/2}$ (so that $\Sigma_{par}^{1/2}\Sigma_{par}^{1/2} = \Sigma_{par}$) and define $\psi(\mathbf{x}) = \Sigma_{par}^{1/2}\phi(\mathbf{x})$ so that $\phi(\mathbf{x}_i)^T\Sigma_{par}\phi(\mathbf{x}_j) = dot(\psi(\mathbf{x}_i), \psi(\mathbf{x}_j))$ where *dot* denotes the usual vector dot product. Looking back at Equation 2.3, we see that it can now be formulated in terms of a *kernel function* that is defined by this dot product, $k(\mathbf{x}_i, \mathbf{x}_j) = dot(\psi(\mathbf{x}_i), \psi(\mathbf{x}_j))$:

$$p(f_*|\mathbf{x}_*, X, \mathbf{y}) \sim \mathcal{N}(K_{*,\mathbf{f}}(K_{\mathbf{f},\mathbf{f}} + \sigma_{noise}^2\mathbf{I})^{-1}\mathbf{y}, K_{*,*} - K_{*,\mathbf{f}}(K_{\mathbf{f},\mathbf{f}} + \sigma_{noise}^2\mathbf{I})^{-1}K_{\mathbf{f},*}). \quad (2.4)$$

In this equation, $K_{*,\mathbf{f}}$ signifies the vector with entries $k(\mathbf{x}_*, \mathbf{x}_1), k(\mathbf{x}_*, \mathbf{x}_2), ..., k(\mathbf{x}_*, \mathbf{x}_N)$ and $K_{\mathbf{f},\mathbf{f}}$ is a matrix with ij^{th} entry equal to $k(\mathbf{x}_i, \mathbf{x}_j)$. It is possible and indeed often useful to implicitly define the dot products in terms of a kernel function; in this case, the algorithm can work with dot products in high (or infinite) dimensional spaces without explicitly computing the dot products feature-by-feature.

We have now formulated kernelized Bayesian linear regression, which shifts the data into a possibly infinite dimensional feature space and efficiently finds a linear fit to the data in that space, enabling nonlinear predictions in the original input space. The feature space is fully defined by the kernel k. This function is called a *covariance function* in Gaussian Process (GP) literature. It turns out that kernelized Bayesian linear regression is equivalent to GP regression, which we now describe from a more direct point of view.

2.3.2 Basic GPR: Training and Testing

In this short introduction to GP regression (GPR) from the function space perspective we mostly follow the concise overview in Quinonero-Candela and Rasmussen [2005]; for a detailed treatment of Gaussian Processes see Rasmussen and Williams [2006].

A Gaussian Process is a generalization of the Gaussian distribution to stochastic processes. Instead of finite dimensional vectors, a GP is defined over functions; this extension becomes intuitive if functions are seen as vectors of infinite (often uncountable) dimension. If we have a collection of random variables $\{f(\mathbf{x})\}_i$ such that any finite number of them has a joint Gaussian distribution, we write

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

where m is a mean function and k is a covariance function which specifies how closely correlated or similar any two random variables are[1]. Intuitively, if we draw functions from this distribution we are most likely to get functions "similar" to m on average, with any additional assumptions like smoothness, stationarity, local independence etc. encoded by k. The covariance function specifies correlations between random variables which in case of regression will be the latent function values $f(\mathbf{x}_i)$ on inputs $\mathbf{x}_i$. However, these covariances will often be functions of distances between the input points, so that $k(f(\mathbf{x}_i), f(\mathbf{x}_j)) = h(\mathbf{x}_i, \mathbf{x}_j)$ where h is some function from the input space into the real line. As a consequence, we will often write simply $k(\mathbf{x}_i, \mathbf{x}_j)$; it should be remembered, however, that we not try to model the input distribution here.

The above defines a GP distribution over functions. This distribution can be used in Bayesian regression as a prior. If the likelihood (that models the observation noise) takes the common form of Gaussian noise, many of the usual Bayesian integrals can be solved analytically. This fortunate behavior makes GPR a simple and useful method. Formally, assume again that we are given a data set $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ of inputs $\mathbf{x}_i$ and outputs y_i and make the standard independent Gaussian noise assumption,

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \text{ where } \varepsilon_i \sim \mathcal{N}(0, \sigma^2_{noise}).$$

Crucially, we set the prior to a Gaussian Process (with zero mean):

$$p(\mathbf{f}|\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N) = \mathcal{N}(\mathbf{0}, K_{\mathbf{f},\mathbf{f}})$$

where $\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \cdots, f(\mathbf{x}_n)]$ is a vector of function values and $K_{\mathbf{f},\mathbf{f}}$ is a covariance matrix defined as in Section 2.3.1.. To predict the values $\mathbf{f}_*$ of the generative function on a set of test points we calculate the joint posterior

$$p(\mathbf{f}, \mathbf{f}_*|\mathbf{y}) = \frac{p(\mathbf{f}, \mathbf{f}_*)p(\mathbf{y}|\mathbf{f})}{p(\mathbf{y})} \tag{2.5}$$

and marginalize the latent training functions values:

$$p(\mathbf{f}_*|\mathbf{y}) = \int p(\mathbf{f}, \mathbf{f}_*|\mathbf{y}) d\mathbf{f} = \frac{1}{p(\mathbf{y})} \int p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}, \mathbf{f}_*) d\mathbf{f}. \tag{2.6}$$

We defined both the likelihood and the prior to be Gaussians so the solution is a Gaussian too. After going through some linear algebra (again, see Rasmussen and Williams [2006] for the details) we get the posterior

$$p(\mathbf{f}_*|\mathbf{y}) = \mathcal{N}(K_{*,\mathbf{f}}(K_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}\mathbf{y}, K_{*,*} - K_{*,\mathbf{f}}(K_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}K_{\mathbf{f},*}). \tag{2.7}$$

[1]The covariance function must be positive semi-definite, as any covariance matrix is. Section 2.3.1 shows that this is necessary for GPR to be equivalent to kernelized linear regression.

This exactly repeats Equation 2.4. Note the form of the mean prediction vector. The predicted value on each test point is a weighted sum of the training output values. The weights are fully determined by the covariance function, which specifies the influence of each training point on the test variable, and the (noisy) precision matrix. The prior predictive uncertainty on any test point ($K_{*,*}$) is decreased by the amount of information the training points should give us about the test case $K_{*,\mathbf{f}}(K_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}K_{\mathbf{f},*}$. Note that this term is independent of any output values in a GP, but it does not have to be for stochastic processes in general. Figure 2.2 shows a toy regression problem and its GPR predictive distribution. Comparing with Figure 2.1, we see that GPR with an appropriate kernel allows for highly nonlinear predictions; furthermore, the predictive distribution estimates the certainty of prediction, suggesting regions in input space that the algorithm has little reliable information about. This makes the algorithm very flexible and usable in all kinds of non-standard settings some of which we already mentioned in Section 1.1.

2.3.3 Hyperparameter Estimation

An important aspect of Bayesian inference is the availability of a general process for choosing any hyperparameters present in the model. In the case of GPR, the hyperparameters are any parameters used by the covariance function, and the noise variance of the data. A commonly used covariance, which we also use in the experiments presented in this dissertation, is the Squared Exponential function

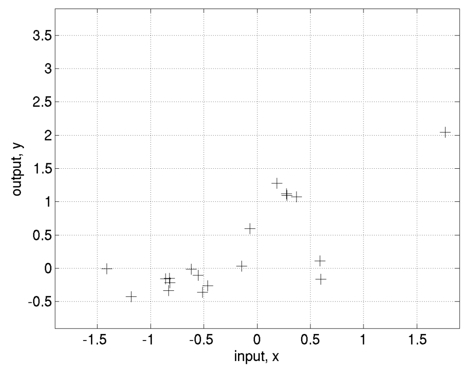
$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}\Sigma_{d=1}^{D} \frac{(x_{d,i} - x_{d,j})^2}{l_d^2}\right) \tag{2.8}$$

where l_d specifies how far the influence of a point reaches in the d^{th} dimension of input space and σ_f^2 is the squared amplitude of the signal. In the Bayesian framework the hyperparameters can be chosen to maximize the likelihood of the data. This has the fortunate effect of favoring models of just-right complexity, the so called Bayesian Occam's Razor (see for example MacKay [2003], Chapter IV). The log marginal likelihood to be minimized is

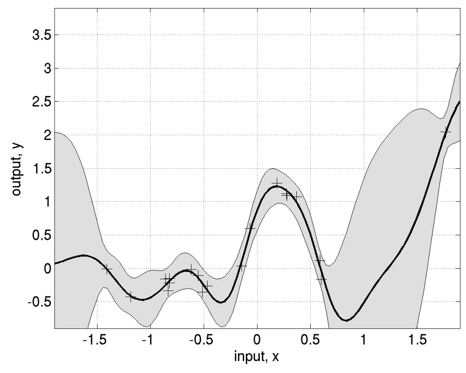
$$\log p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^T(K + \sigma_n^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log|K + \sigma_n^2 I| - \frac{N}{2}\log 2\pi, \tag{2.9}$$

and its partial derivatives with respect to the hyperparameter

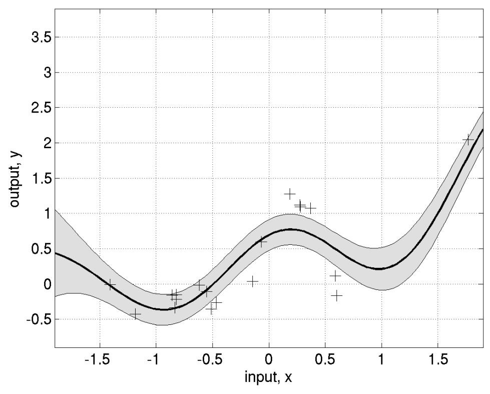
$$\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|X,\theta) = \frac{1}{2}tr\left((\alpha\alpha^T - K^{-1})\frac{\partial K}{\partial\theta_j}\right), \tag{2.10}$$



(a)



(b)



(c)

Figure 2.2: Example data drawn from a Gaussian Process with a Squared Exponential covariance with lengthscale $l = 0.25$ and noise standard deviation $\sigma_n = 0.1$ (a) and resulting predictive distributions (b), (c). In (b) we make GPR predictions using the correct hyperparameters; the mean prediction is shown as a continuous curve. Also shown is one standard deviation of the Gaussian distributed expected error on each test point. As expected, the confidence of the predictions falls away from the datapoints. In (c) we set the hyperparameter lengthscale to be 1 (Section 2.3.3 discusses GP hyperparameters), hence the prediction is much smoother. However, we did not adjust the noise level (it stays at $\sigma_n = 0.1$). This particular hyperparameter misestimation makes some of the training point values to be outside of one predictive error standard deviation. This illustrates the need for good hyperparameter optimization. The plots are generated using modified code from Rasmussen and Nickisch [2010].

with $\alpha = K^{-1}\mathbf{y}$ and tr calculates the trace of a matrix (see Rasmussen and Williams [2006], Chapters 2 and 5). Gradient descent techniques can thus be used to minimize $-\log p(\mathbf{y}|X,\theta)$, in hope any bad local minima will be skipped and hyperparameters fitting the data well are found.

Chapter 3

GPR Approximations

Chapter 2 showed that Gaussian Processes can be used as Bayesian priors over functions in the regression task. Equation 2.7 formulates the predictive distribution calculated in such model (with stationary Gaussian noise). Calculation of the mean and variance of the predictive distribution on any test point requires using the inverted "noisy covariance matrix" $(K_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}$. Similarly, each iteration of gradient descend hyperparameter optimization will require inversions of K. Since K is of size $N \times N$, to run the exact Gaussian Process Regression algorithm we need $O(N^3)$ time and $O(N^2)$ space, where N is the number of training points. This is prohibitive for datasets larger than several thousand points, which are not difficult to obtain. Many approximation algorithms were created to deal with this issue; the Gaussian Processes web site (Rasmussen [2011]) picks thirty-one publications on approximation algorithms as of February 2011. Our goal is to clarify which approximations are useful in practice, why, and in what situations. We have chosen four algorithms to focus on, as they are representative of four different approaches to the approximation process. Sections 3.1 - 3.4 discuss the basic ideas behind these algorithms, and are complemented by Sections 3.5 and 3.6 which elaborate on two algorithmic choices that need to be made before some of the methods are used. Chapter 4 elaborates on our implementations of the approximation algorithms, including a more detailed (not asymptotic) time and space complexity analysis.

3.1 Subset of Data

The Subset of Data (SoD) method is a simple way of speeding up Gaussian Process Regression. Given N training datapoints, the method chooses a subset of m points (which we will call *inducing points*) and performs inference as usual, using this subset only. The SoD predictive distribution (compare with Equation 2.7) is then

$$p_{SoD}(\mathbf{f}_*|\mathbf{y}) = \mathcal{N}(K_{*,\mathbf{f}_{SoD}}(K_{\mathbf{f}_{SoD},\mathbf{f}_{SoD}} + \sigma^2_{noise}\mathbf{I})^{-1}\mathbf{y}_{SoD}, K_{*,*} - K_{*,\mathbf{f}_{SoD}}(K_{\mathbf{f}_{SoD},\mathbf{f}_{SoD}} + \sigma^2_{noise}\mathbf{I})^{-1}K_{\mathbf{f}_{SoD},*}),$$

where $K_{*,\mathbf{f}_{SoD}}$ is the covariance vector between the test value $f(\mathbf{x}_*)$ and the m latent function values $\{f(\mathbf{x}_i)|\mathbf{x}_i \in SoD\}$ and $K_{\mathbf{f}_{SoD},\mathbf{f}_{SoD}}$ is the $m \times m$ covariance matrix on the points in the chosen subset of data SoD. $\mathbf{y}_{SoD}$ are the training outputs on the points in the chosen subset of data.

The running time of the method is $O(m^3)$ for training and $O(m^2)$ per test point for computing the posterior distribution (means and variances). The space complexity is $O(m^2)$ for storing the covariance matrix over the chosen data subset. Figure 3.1(b) shows SoD solving a toy dataset.

An additional choice one has to make to use SoD is how to choose the inducing points. This turns out to be a more general issue valid for other approximations as well. We discuss it in some detail in Section 3.6.

SoD is the simplest approximation we consider. It might seem wasteful to simply throw away data without retaining any information about it. However, in the limit of having infinite data at our disposal, throwing some of it away might be a necessity. In fact, one might be forced to connect another approximation with SoD to be able to use it effectively. In particular, we discuss such connection with later in this dissertation.

3.2 Local GP

The local approximation divides the training set (with N training points) into clusters of size at most m each. Each cluster is then treated as a separate inference problem; in case of Gaussian Process Regression, the full GPR algorithm is performed on each cluster separately, with no information transfer between the clusters. If the hyperparameters are shared between the clusters, the full GP predictive distribution from Equation 2.7

is now approximated by

$$p_{Local}(\mathbf{f}_*|\mathbf{y}) = N(K^{Local}_{*,\mathbf{f}}(K^{Local}_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}\mathbf{y}, K^{Local}_{*,*} - K^{Local}_{*,\mathbf{f}}(K^{Local}_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}K^{Local}_{\mathbf{f},*}),$$

where $K^{Local}_{a,b}$ is a block diagonal covariance matrix with blocks of size at most $m \times m$ (see Equation 3.2 below).

The time cost of training Local GPR is $O(m^3 \times \frac{N}{m}) = O(Nm^2)$. The test time is $O(m^2)$ per test case- with precomputed m-dimensional Cholesky factors one triangulated linear system in m variables has to be solved per test case to get predictive variances. The space required is $O(m^2)$– one cluster covariance matrix needs to be stored at any one time.

Local GP is not a stand-alone approximation. One needs to choose the underlying clustering method– an important choice elaborated on in Section 3.5 where we describe Recursive Projection Clustering (RPC) and Recursive Random Clustering (RRC). Additionally, two obvious hyperparameter training variants are possible. The hyperparameters can be chosen separately for each cluster, or a joint optimization can be performed (so that the hyperparameter gradient is taken to be the sum of the gradients of cluster hyperparameter variables). In case joint hyperparameter optimization is performed, we can model Local GP as full GP Regression using a modified kernel

$$k^{Local}(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} k(\mathbf{x}_i, \mathbf{x}_j) & \text{if } \mathbf{x}_i, \mathbf{x}_j \text{ in the same cluster} \\ 0 & \text{otherwise} \end{cases}$$

In this case the covariance matrix is block diagonal, as in Equation 3.2. For example, if $m = 2$ then the “joint-Local” covariance is:

$$K^{Local} = \begin{pmatrix} k_{1,1} & k_{1,2} & 0 & . & . & . & . & . & 0 \\ k_{2,1} & k_{2,2} & 0 & . & . & . & . & . & 0 \\ 0 & 0 & k_{3,3} & k_{3,4} & & & & & \\ . & . & k_{4,3} & k_{4,4} & & & & & \\ . & . & & & . & & & & \\ . & . & & & & . & & & \\ . & . & & & & & . & & \\ 0 & 0 & 0 & 0 & . & . & . & k_{N-1,N-1} & k_{N-1,N} \\ 0 & 0 & 0 & 0 & . & . & . & k_{N,N-1} & K_{N,N} \end{pmatrix}$$

where $k_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$ for brevity. If the hyperparameters differ between the clusters, each block will in practice use a different covariance function and the algorithm can not be seen as performing GP regression. However, we can expect such model to have more expressive power. Figure 3.1(c) shows the results of running Local GP with joint hyperparameters on a toy dataset. Note the discontinuities in the overall good prediction.

3.3 Fully Independent (Training) Conditional

FITC is a version of Ed Snelson's Sparse Pseudo-Input Gaussian Process (SPGP) method described in detail in Snelson [2001]. It was integrated into a general GP approximation framework and renamed in Quinonero-Candela and Rasmussen [2005]. Let $\mathbf{u}$ be a set of m *inducing points*, ideally chosen so that every test point is close to this set[1]. Let $k(\mathbf{x}_i, \mathbf{u})$ denote the m element row vector of covariances between $\mathbf{x}_i$ and the inducing points; and let $K_{\mathbf{u},\mathbf{u}}$ denote the $m \times m$ matrix of covariances between the inducing points. As Quinonero-Candela and Rasmussen [2005] shows, FITC is performs GP regression using a modified covariance function

$$k_{FITC}(\mathbf{x}_i, \mathbf{x}_j) = k_{SoR}(\mathbf{x}_i, \mathbf{x}_j) + \delta_{i,j}[k(\mathbf{x}_i, \mathbf{x}_j) - k_{SoR}(\mathbf{x}_i, \mathbf{x}_j)]$$

where k is the original covariance function used in the GP to be approximated and

$$k_{SoR}(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{u}) K_{\mathbf{u},\mathbf{u}}^{-1} k(\mathbf{u}, \mathbf{x}_j)$$

is the Subset of Regressors (SoR) covariance function. Intuitively, k_{SoR} allows any two points in the input space to communicate only through the m inducing points. k_{FITC} is similar but keeps the self-covariances of the data intact. Quinonero-Candela and Rasmussen [2005] gives a detailed discussion on the two functions. Using k_{FITC} instead of k reduces the runtime from $O(n^3)$ to $O(m^2 n)$ in the training phase and $O(m^2)$ in the test phase, thanks to a clever use of the matrix inversion lemma (see Snelson [2001], Sections 1 and 2). Figure 3.1(d) shows FITC regression used on a simple toy problem. In the figure, FITC is closer to the full GP prediction than SoD, but fails to capture information about the one datapoint far away from all inducing points.

[1] Originally, Snelson proposed to optimize the locations of the inducing points during hyperparameter optimization. This extension makes the algorithm scale worse with dimensionality, but improve its performance. In this dissertation we consider only the apparently more popular version of the algorithm where the inducing points are chosen by simpler methods as described below.

3.4 Improved Fast Gauss Transform MVM

IFGT (Raykar et al. [2005]) is a fast matrix-vector multiplication (MVM) method developed specifically for Gaussian potentials. It uses power expansion truncation and subdivision of input space to evaluate the potentials approximately. It was suggested by Morariu et al. [2008] that it can speed up Gaussian Process Regression if the Squared Exponential kernel is used. As already noted in Gibbs and MacKay [1997], the covariance matrix inversion performed in GPR, $(K_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}\mathbf{y}$ (see Section 2.3.2), can be done using Conjugate Gradients methods. The covariance matrix is positive definite so CG is readily applicable, and one could stop the CG algorithm after $m < N$ iterations. More importantly, during CG iterations, as well as when computing the final output values on the test points $\mu_* = K_{*,\mathbf{f}}\alpha$ where $\alpha = (K_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}\mathbf{y}$, almost all computational cost comes from matrix-vector multiplications (MVMs).

The algorithm itself is complicated. Below we roughly sketch the procedure, basing on Yang et al. [2004]. Note that the authors were unable to compress a full description of the algorithm into the 8-page NIPS limit, and hence their description does not fully define all the terms, referring the reader to technical reports. We do not hope to be able to condense the algorithm's description better than the experts, but we hope the reader can get an idea behind the basic procedure used in IFGT, and how it can help in working with Gaussian Processes, from the short description below.

Assume $\mathbf{x}_*$ is a test point and α_i is the ith element of the weight vector $\alpha = (K_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}\mathbf{y}$ from Equation 2.7. IFGT will then compute the mean GP prediction on $\mathbf{x}_*$, assuming the Squared Exponential covariance with lengthscale l is used, with a bounded error. The error guarantee is based on a complicated error function; the reader needs only to assume that it is possible to bound the error of the IFGT approximation E based on the radii of the created data partitions and accuracy of power expansions.

1. **Required**: compute the approximate value of the summation $\mu_* = \sum_{i=1}^{N} \alpha_i \exp\left(-|\mathbf{x}_i - \mathbf{x}_*|^2/l^2\right)$, with absolute error at most ε.
2. **Procedure**: Perform Farthest Point Clustering (Section 3.6.2) to partition the datapoints $(\mathbf{x})_i$ into k clusters. Choose k so that each cluster's radius is smaller than $|l|\rho$, where ρ is a term that influences the error bound in the next step.
3. 2. Choose p - which determines the order of the power expansion of the Gaussian in the next step - large enough for the error bound to guarantee $E(\mu_*) \leq \varepsilon$.

4. For each cluster, compute a truncated (above the pth degree) multivariate Taylor series around the cluster's center, using only the points in the cluster.

5. Sum the results of computations for each cluster.

Of course, the procedure presented above can be used for any MVM involving- in our case- a covariance matrix K. It turns out that the IFGT MVMs in CG can be performed in a progressively more inexact manner to speed up the process and the algorithm still converges, as shown in Raykar and Duraiswami [2007]– that is, the allowed MVM error ε can grow with CG iterations in a well-defined manner. Figures 3.1(e) and 3.1(f) show IFGT solving a simple toy regression problem. The prediction can be very bad if the relative allowed MVM error is too large, but it is hard to predict *how* the results will be wrong; in comparison, with SoD, FITC, and Local GP we understand fairly well how the predictive inaccuracies relate to the choice of the inducing points or clusters.

3.5 Clustering

We use two simple clustering methods with Local GP. We deliberately do not use the Farthest Point Clustering method (which we *do* use to choose inducing points for SoD and FITC, see Section 3.6) with Local GP. This is because the method's time complexity scales as $O(m^2N)$ where m is the number of points in the largest cluster. FPC provides no guarantees on the maximum cluster size. The methods presented below are less complex and will likely not exploit the structure of the data as well as FPC, but they are guaranteed to produce clusters smaller than a given constant, and choose the cluster sizes to be roughly uniform under the maximum size constraint.

3.5.1 Recursive Projection Clustering (RPC)

This method, suggested by Iain Murray (personal communication), performs the following recursive procedure.

Initiation Given dataset $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and constant m, define $A = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$.

If $|A| > m$: recursive step. Choose two points uniformly at random from A.

Draw a line through these points and calculate the orthogonal projection of all points from A on the line.

Split A into two equal size subsets: A_L, the points to the left of the median projection and A_R, the points to the right of the median.

Repeat the recursive step, with $A \leftarrow A_L$ first and $A \leftarrow A_R$ second.

If $|A| \leq m$: termination. Add A to the clusters set. Terminate this branch of recursion.

3.5.2 Recursive Random Clustering (RRC)

This method proceeds semi-recursively - the top recursion level is similar but not identical to lower levels. Starting from the full dataset of size N and given m - the maximum cluster size:

```
1. Choose $\lfloor \frac{N}{m} \rfloor$ *cluster centers* uniformly at random from the data.
2. Assign each point in the dataset to the closest center, producing $\lfloor \frac{N}{m} \rfloor$ clusters $C_i$.
   Add these to the clusters list $\mathcal{C}$.
3. For each $C_i \in \mathcal{C}$ with $|C_i| > m$:
4.         Remove $C_i$'s from $\mathcal{C}$.
5.         Choose two points in $C_i$– new cluster centers– uniformly at random.
6.         Assign each point in $C_i$ to the closer of the two points, creating two clusters
   $C_k, C_l$. Add $C_k$ and $C_l$ to $\mathcal{C}$.
7.         If any of the two new clusters has size $> m$, repeat steps 4-7 on this cluster.
```

By construction, RPC produces clusters of almost uniform size, equal $\min\limits_{c:n2^{-c} \leq m} n2^{-c}$ where c is a natural number and the actual cluster sizes are rounded to closest natural numbers. RRC's clusters are less uniform in size. Figure 3.2 compares the two clustering methods graphically . The figure shows that in general, RRC's clusters are smaller than RPC's, as expected. This can have a slightly detrimental effect on Local GP's performance if all other factors are ignored.

3.6 Choosing the Inducing Points

The SoD and FITC approximations require that a set of *inducing points* is defined in the input space. In both cases, these points are taken to be somehow representative of the whole training input set. We saw in Figure 3.1 that both algorithms give better predictions close to the inducing points set. Snelson [2001] proposes that the locations of inducing points be optimized in the likelihood as hyperparameters. We decided to take simpler approaches that do not depend on the dimensionality of the problems; otherwise, a general comparison of FITC with the remaining methods would be more difficult. Instead, we used two less computationally demanding approaches to inducing points' selection. Assume we need to obtain a set of m inducing points.

3.6.1 Random subset

The first approach is choosing a random subset of data and was also used in Rasmussen and Williams [2006]. Note that this might not be as bad an option as it seems at first: random choice will respect the structure of the data in the sense that the denser areas (which therefore contribute more to the errors and are perhaps less likely to contain outliers) will have more inducing points placed in them.

3.6.2 Farthest Point Clustering

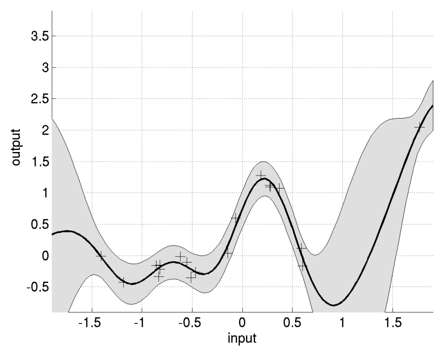
The second method we consider is to use the set returned by the Farthest Point Clustering (FPC) algorithm as the centers of m clusters. Gonzalez [1985] proves that this algorithm is the best possible approximation to the *NP-hard* problem of finding a clustering with the smallest maximum intracluster distance. Thus we can hope that the set of inducing points placed at the centers of such chosen clusters is in a way as close to all other data as possible. This simple algorithm chooses k data clusters as follows:

```
1. Choose a point, $c_1$, at random. This will be the center of the first cluster.
2. For $i = 2, \cdots, k$
3.         $c_i \leftarrow$ the point farthest from the set $\{c_1, \cdots, c_{i-1}\}$.
4. Move each datapoint $x_k$ to cluster $l \in \{1, \cdots, k\}$, whose center $c_l$ is closest to $x_k$.
```

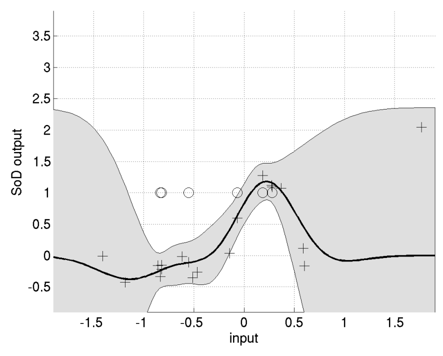
We also experimented with the IVM method which greedily chooses the subset of data that decreases uncertainty on training data most (Lawrence et al. [2003]), but the results seemed very similar to when FPC was used. This makes intuitive sense: for stationary kernels (where covariance is a function of inter-point distances only) and small data subsets, a point farthest from a subset will also decrease the overall uncertainty the most. We did not run all the experiments using IVM because of this and do not report on the method further.

3.6.3 Differences between Random and FPC

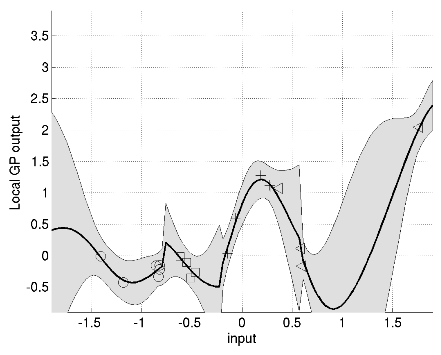
Figure 3.3 illustrates some differences between these two methods when used on the SYNTH2 dataset (Section 5.1 describes this and other datasets we use in our experiments). The figure shows that in low dimensionalities FPC is able to choose inducing points regularly across the input space, while Random is more sensitive to irregularities in data structure. However, in higher dimensions many more points are needed to fill up space, and FPC fails to produce a regular coverage of the space. Random, on the other hand, will still focus on the more important data regions; in high dimensions Random will ignore less dense regions which can reduce accuracy of the inference methods we use (which will then ignore those regions), but this seems the only feasible option in high dimensionalities when the data has little structure. The bottom plots in the Figure plot the inducing point's standard deviation when they are chosen by FPC and Random respectively, as a function of growing subset size. For large subsets both methods converge to the full dataset variance. For small subsets, however, the methods behave very differently in 2 and 8 dimensions, as explained above.



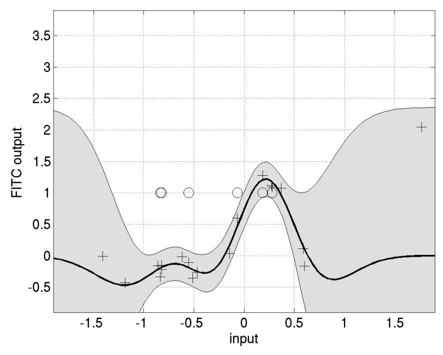
(a)



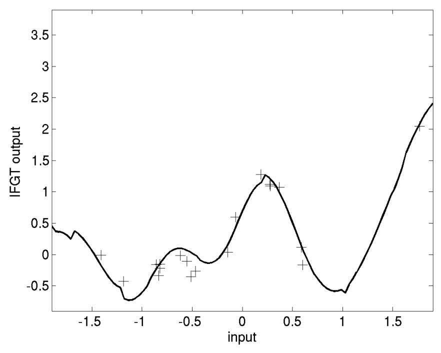
(b)



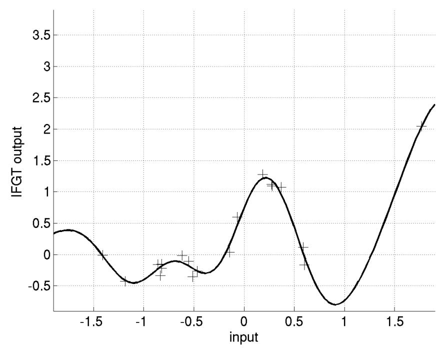
(c)



(d)



(e)



(f)

Figure 3.1: Toy data drawn from a Gaussian Process (same as in Figure 2.2) and resulting predictive distributions calculated using approximate GPR algorithms and the correct hyperparameters. Figure (a) shows the full GP prediction. (b) The Subset of Data approximation predicts badly away from the inducing points (shown as circles), but the predictive variances are adjusted accordingly. (c) Local GP introduces discontinuities on the edges between clusters (each cluster marked by different symbols) but doesn't ignore any data regions. (d) FITC can do better than SoD in some regions. (e) IFGT, with a high maximum allowed MVM error; the prediction seems to follow the overall trend in the data. (f) In this case, the allowed MVM error is very small and the IFGT prediction is almost identical to the full GP. IFGT does not yield predictive variances efficiently, hence they are not shown here (Chapter 4 discusses this and other issues with IFGT).

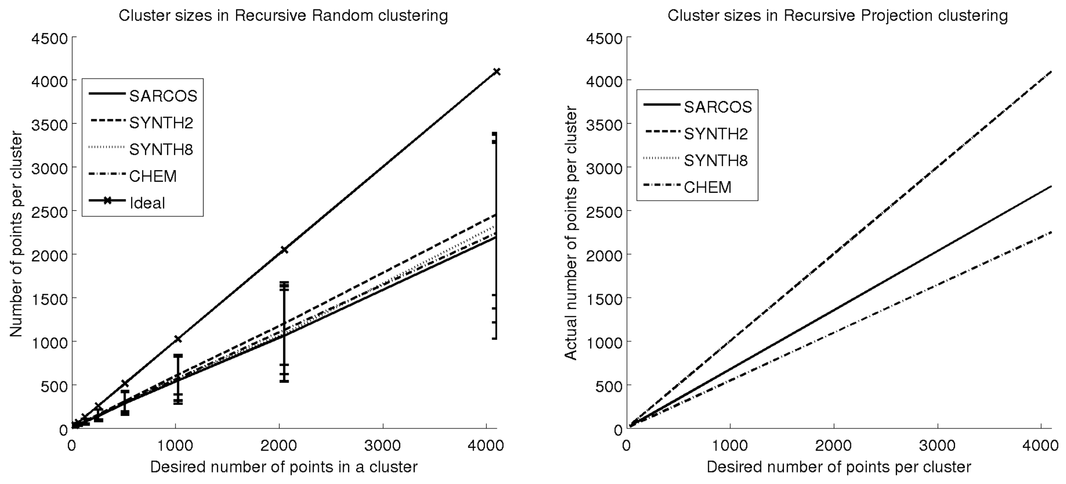


Figure 3.2: Cluster sizes for RPC and RRC. Note almost perfectly linear cluster size growth. The solid black crossed line shows the ideal behavior - clusters of size exactly m as chosen by the user. The values are means taken over 10 trials for each m. The error bars show, for RRC, one standard deviation. RPC's cluster size variance is negligible. RPC's cluster sizes are a deterministic function of the dataset's size only. For the SYNTH datasets, where the training set size is a power of 2, RPC produces clusters of exactly the desired size– as expected. RRC produces clusters of variable size; the sizes are nondeterministic but influenced by the structure of the data.

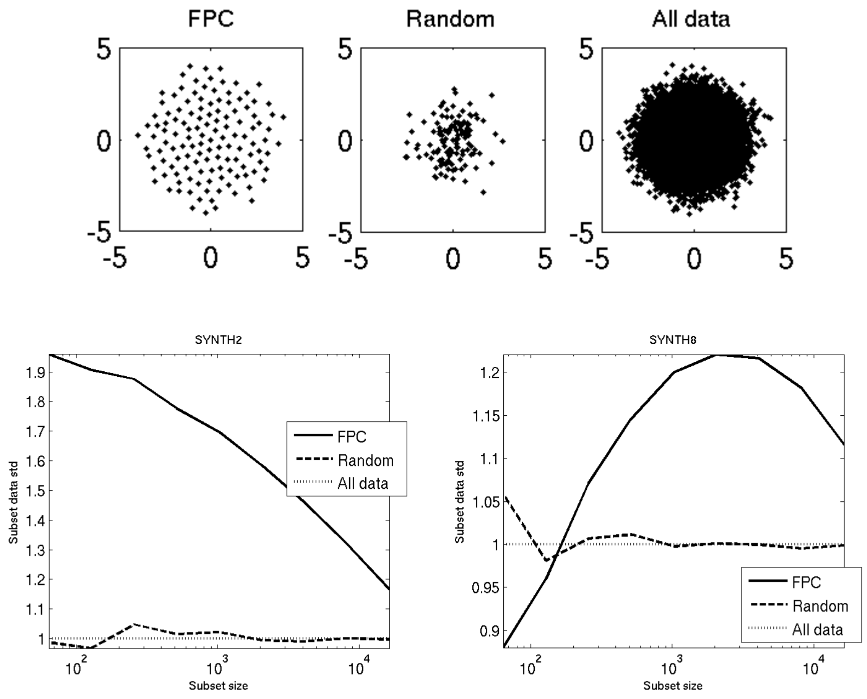


Figure 3.3: FPC and Random subset choice. The average of standard deviation over the data dimensions is shown. **Top** on low dimensional datasets (SYNTH2 here) FPC places the cluster centers on a regular pattern which covers the whole space well. In this case, random subset choice largely ignores the exterior of the data distribution. The SYNTH dataset inputs are sampled from a Gaussian distribution, hence Random has much more points to choose from in the center. In the plots 128 points were chosen by each algorithm from among 32000 training cases shown on the right. **Bottom** the situation is different in high dimensions, where it is more difficult to populate the space densely. The **left** plot shows that for 2 dimensional SYNTH2, Random consistently chooses subsets with variance around one (the mean of standard deviations across all dimensions is shown). FPC for small subsets chooses points roughly on a regular pattern, also on the outskirts of the sample distribution (top left plot), but for larger subset sizes it must start converging to the true data distribution- hence the variance is falling towards value one. The plot on the **right**, on the contrary, shows that in eight dimensions it takes more time for FPS to yield wide subsets. In eight dimensions there is much more volume on the exterior of the distribution. By its nature, FPC will first place points on the exterior, largely ignoring the interior of the sample distribution. This is prone to create thin subspaces with small average variance for small subset sizes (an 8 dimensional hypercube has $2^8 = 256$ faces), and only for large subsets FPC is starting to cover a decent share of the space. The variance starts decreasing when, as in the two dimensional case, FPC is placing more points in accordance with the real data's normal distribution.

Chapter 4

Implementation

Our goal is to compare the GP approximations described in Chapter 3 empirically. To do this, we needed code implementing the approximations, as well as any of the additional algorithms used for clustering etc. Part of our work had already been done by other researchers, and we use their implementations whenever possible. This chapter describes in some detail the programming work we have done, and attributes outside code that we use to its creators. Section 4.1 describes the implementation of the core approximations. We note that using external code was not trivial: we had to put much work into integrating some of the code into our framework, and we dealt with some more and less important bugs present in outside source code. Some of our fixes have been incorporated into the original projects.

This chapter also looks at the actual complexity of the algorithms as we have them implemented, in Section 4.3. These practical (as opposed to asymptotic) complexities will be important in practice when in later chapters we compare the results of running the algorithms on actual complex datasets. The time/space analysis here is still approximate, skipping some important details such as calculating kernel function values, and some less important details like vector-vector operations (matrix-vector operations constitute most of complexity of these algorithms). We hope that such approximate analysis can give the reader an appreciation that asymptotic complexities can ignore significant differences in actual runtimes.

Finally, Sections 4.4 and 4.5 look at some additional code we wrote to be able to work with the GP approximation algorithms.

Appendix B describes the structure of our codebase in more detail, so the code can be reused easier should anyone desire to do so.

4.1 GPML

The most important codebase we use is the Gaussian Processes GPML package (Rasmussen and Nickisch [2010]). The GPML project went through a major revision halfway through our work and we updated our code accordingly, which was a major change especially for the SoD and Local approximations. The current GPML was created by Carl Edward Rasmussen and Hannes Nickisch, the earlier version by C. E. Rasmussen and Chris Williams. Most GPML code is written in Matlab.

4.1.1 SoD and Local GP

The most recent GPML toolbox – version 3.1 – can perform full Gaussian Process Regression and Classification, using a variety of covariance functions as well as several likelihood models. In our experiments, we use the Squared Exponential covariance function and Gaussian likelihood, both of which are implemented in the GPML package. Thus to perform SoD (Section 3.1), once the inducing points are chosen (see below), we simply use GPML's full GP regression on this subset of data.

In case of Local GP (Section 3.2) with separate hyperparameters, once the training data is clustered (see below), each cluster uses GPML to perform GP inference. Local GP with joint hyperparameters is slightly more tricky if it is to be done efficiently. We extended the GPML package to handle this situation, so that Maximum Likelihood hyperparameter estimation can use the sum of the gradients of all clusters on each iteration.

While implementing Local GP, we have spotted an efficiency issue in the GPML codebase. Fixing it speeds regression up, in some cases by over 90% (Section 4.3.2 talks about other issues appearing in the situation where this bug was spotted, that is Local GP with small clusters). Our fix was incorporated in the newest GPML version[1].

[1]See `http://www.gaussianprocess.org/gpml/code/matlab/doc/changelog`.

4.1.2 FITC

Originally we used Ed Snelson's implementation of his SPGP method (Snelson [2006]). This required modifications as Snelson assumes the user wants to optimize the pseudoinput locations in the likelihood, which we don't (Section 3.3). However, halfway through the project a new version of GPML was released which implemented FITC as we need it. Thus we modified our code accordingly, and currently we use GPML's FITC routine. As it turned out, this fresh implementation contained a simple but serious bug which we fixed; our fix again is incorporated in the newest version of GPML code. Since we don't optimize the inducing point locations, we choose these locations using other methods which we implemented ourselves, described below.

4.2 Figtree

The Improved Fast Gauss Transform is implemented in the C++ Figtree package by Vlad Morariu (Morariu [2010] – the package includes Matlab wrappers which we use). This, however, is just a fast Matrix-Vector Multiplication method. We have implemented a progressively more inexact Conjugate Gradient algorithm that uses Figtree for MVM (the idea described in Raykar and Duraiswami [2007]). We then implemented Gaussian Process Regression that uses our CG routine to invert the necessary matrices.

4.2.1 IFGT predictive variances

It is relatively easy to implement mean GPR prediction using CG with IFGT. However, a problem so far ignored by the proponents of IFGT is that it is not clear how to use fast MVMs to estimate the hyperparameters and compute the solution predictive variances efficiently. The predictive distribution variance of full GPR on test point $\mathbf{x}_*$ is computed as

$$K_{*,*} - K_{*,\mathbf{f}}(K_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}K_{\mathbf{f},*},$$

where $\mathbf{f}$ symbolizes all the N training inputs. Our implementations of SoD, Local and FITC store Cholesky factors of relevant covariance matrices. The factors are precomputed during the training phase; then, during test time, the covariance matrix needs not be inverted again. Instead, a triangular linear system is solved, one for each test point.

IFGT on the other hand does not precompute partial solutions to matrix inversion. The obvious approach to computing the predictive variances requires re-solving the linear system $(K_{\mathbf{f},\mathbf{f}} + \sigma^2_{noise}\mathbf{I})^{-1}K_{\mathbf{f},*}$ for each test point $\mathbf{x}_*$ with respective training-test covariance vector $K_{\mathbf{f},*}$. This seems impractical even for small datasets, in our experience. In Raykar and Duraiswami [2007] and Murray [2009] the authors focus on IFGT's MSE performance only and skip the variance problem. Gibbs and MacKay explicitly note that the cost of such approximate variance prediction is larger than of full variance prediction (Gibbs and MacKay [1997]). We suggest that for the algorithm to be considered useful, a way of efficiently computing the variances should be found. This of course applies to any iterative method for solving GPR. We have implemented the obvious method of computing the variances, but don't report the IFGT variance results in the following chapters as the time costs, which are larger than for the full GP, are prohibitive for nontrivial problems.

4.2.2 IFGT hyperparameter optimization

Hyperparameter computation is also problematic with IFGT. In full GP, to estimate the hyperparameters using the MLE method, likelihood derivatives respective to each hyperparameter need to be computed:

$$\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|X,\theta) = \frac{1}{2}tr\left((\alpha\alpha^T - K^{-1})\frac{\partial K}{\partial\theta_j}\right) \tag{4.1}$$

where $\alpha = K^{-1}\mathbf{y}$ (see Rasmussen and Williams [2006] for a derivation). This allows for a precomputation of the weight vector α which is also used for mean prediction computation. However, this simplified form of the likelihood derivative seems unsuitable for computations with IFGT. This is because IFGT only works with Gaussian potentials. We can in principle split the computation in two parts:

$$K^{-1}\frac{\partial K}{\partial\theta_j} \tag{4.2}$$

$$\alpha\alpha^T\frac{\partial K}{\partial\theta_j} \tag{4.3}$$

where computation 4.2 is computed using n conjugate gradient runs (as with the variances, the full $n \times n$ linear system needs to be be solved each time K^{-1} is multiplied by a matrix column or a vector). Because it doesn't involve Gaussian potentials, computation 4.3 has to be solved exactly. One way to approach the problem is to expand

Equation 4.1:

$$\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|X,\theta) = \frac{1}{2}\mathbf{y}^T K^{-1}\frac{\partial K}{\partial\theta_j}K^{-1}\mathbf{y} - \frac{1}{2}tr(K^{-1}\frac{\partial K}{\partial\theta_j})$$

where this time we can avoid storing large matrices if each time we compute the derivative matrix column by column to perform the relevant IFGT MVM's. Also storing $K^{-1}\frac{\partial K}{\partial\theta_j}$ is not necessary. Gibbs and MacKay [1997] show that trace of a large matrix can be computed approximately with low error using the following simple randomized method. If

$$\tau = \mathbf{d}^T K^{-1}\frac{\partial K}{\partial\theta_j}\mathbf{d} \tag{4.4}$$

where $\mathbf{d}$ is a vector of zero mean, unit variance Gaussian variables then

$$E[\tau] = tr(K^{-1}\frac{\partial K}{\partial\theta_j}).$$

Equation 4.4 again can be computed using IFGT for inversion of K. For large n several samples of τ can suffice to estimate the expectation reasonably. It should be clear by now that if no better methods are proposed, likelihood optimization is not feasible using IFGT or any MVM method. The covariance matrix needs to be re-inverted using CG numerous times for each derivative computation; the partial derivatives matrix also needs to be recomputed column by column several times unless enough space to store it is available. Furthermore, as with all computations, IFGT would only apply in low dimensional cases where it is much simpler and likely equally efficient to use less sophisticated methods. Alternatively, SoD might be used for hyperparameter estimation and IFGT for MSE computations using SoD-estimating hypers.

4.3 Complexity in practice

4.3.1 Space/time complexity

Once we have the concrete implementations of the approximation algorithms, we can look more closely at their actual runtimes. It is possible (and indeed true, as we show in later chapters) that the asymptotics do not kick in until the number of datapoints N reaches several tens of thousands, and hence a less abstract time and space complexity analysis can prove valuable in assessing an algorithm's usefulness. Tables 4.1 and 4.2 compare approximate complexities of Full GP, SoD, Local GP and FITC as implemented using modified routines from the GPML package. Our modifications mostly

	Asymptotic Space	Actual Space
SoD/Full GP[a]	$O(m^2)$	$4m^2$
Local GP	$O(m^2)$	$4m^2$ or $max(4m^2, mN)$[b]
FITC	$O(mN)$	$max(3Nm+5m^2, 4Nm+3m^2, 5Nm+2m^2)$

[a]SoD and full GPR have the same complexity as SoD is just GPR with reduced training set size.

[b]The latter if Cholesky factors for each cluster need to be stored, when test data is unknown at the time of training.

Table 4.1: Space usage by method. Note the asymptotic data can give very different required storage in practice. In each case we report Actual Space as the largest number of doubles stored in Matlab matrices at any time during the algorithms' run. Temporary space needed for linear algebra operations (Cholesky decomposition, Gaussian elimination, matrix multiplication) is ignored, but since only one such operation is performed at once the difference should not be significant. We also ignore the costs of storing single vectors. For IFGT, our progressively more inexact Conjugate Gradient implementation (based on Matlab's pcg routine) stores at most ten n-element vectors at a time; for almost all practical cases IFGT will use the least memory.

reduce GPML's space complexities by cleaning memory whenever possible, but don't change the core GPML code in other significant ways. Our code should be available together with this dissertation for anyone interested. The reported complexities take any matrix-vector and matrix-matrix (with matrices of size $N \times N$ and N-dimensional vectors) multiplications into account, which arguably use almost all time resources required by the algorithms. The table makes it clear that while all of SoD, FITC and Local GP have the same asymptotic complexities, they will require significantly different resources in practice. These additional resources could potentially be used for example to make SoD competitive by taking the size of the subset m_{SoD} larger than the number of FITC inducing points m_{FITC}. One of the goals of our empirical experiments, reported in following chapters, was to evaluate whether such manipulations can be of much practical use.

4.3.2 Local GP Time behavior

An interesting behavior of Local GP that we observed during the experiments (see Chapter 6) is that the algorithm has a convex time curve - it runs slower for tiny cluster

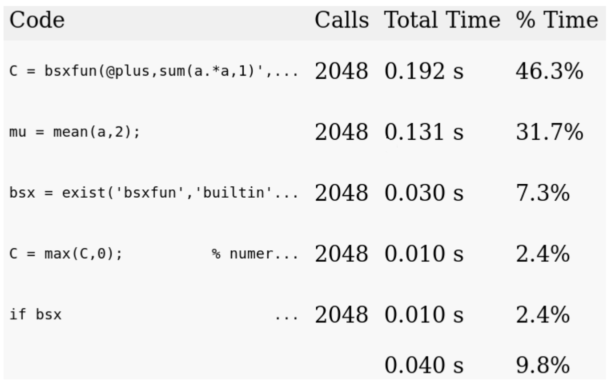
	Chol	**Gauss**	**Mult**
SoD	m^2	$m^2(4m)$	–
Local	$m^2(\frac{N}{m})$	$m^2(4N)$	–
FITC	$m^2 3$	$m^2(2N+4m^2+3)$	$(mN \times Nm)2 + (m^2 \times m^2)$
			$+(m^2 \times mN) + d\left[3(mN \times Nm) + 2(m^2 \times mN)\right]$

Table 4.2: Time usage for SoD, FITC and Local during the training phase. **Chol** row indicates how many Cholesky decompositions of matrices are needed, for example $m^2 3$ indicates three decompositions of $m \times m$ matrices. **Gauss** indicates how many Gaussian eliminations of triangulated linear systems are needed. For example $m^2(2N + 4m^2 + 3)$ indicates a number of operations equivalent to that needed when solving $(2N + 2m^2 + 3)$ triangulated systems of size m^2. **Mult** indicates the number of matrix-vector and matrix-matrix multiplications needed. This *excludes* element-wise multiplications written $X. * Y$ etc in Matlab. For example, $2(mN \times Nm)$ indicates two multiplications AB where A is of size $m \times N$ and B is of size $N \times m$. d is the number of partial derivatives to be computed. These are all lower bounds on the time complexity of the methods; times needed for example for the computations of kernel values, data clusters and many others are not included. d is dimensionality of the data, which indicates the number of lengthscale derivatives the SQ-ARD covariance needs to compute. Note that only FITC requires per-derivate operations that are complicated enough to be included in this table.

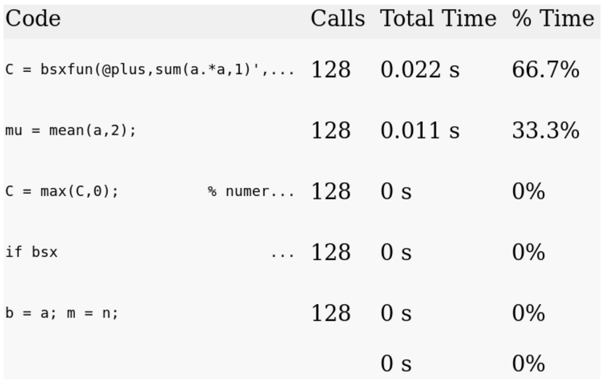
sizes, reaches the best speed somewhere in the middle and again runs slowly for larger cluster sizes. Such behavior does not agree with the estimated time complexity from Table 4.2, which indicates that the time requirement should be a growing function of m. It turned out the reason is that because for small clusters, most time is spent repeating trivial operations on each of the thousands of clusters: setting up/erasing memory, creating namespaces and other bookkeeping etc. Matlab works fast with vectorized algorithms but the smaller the clusters, the less vectorized Local GP is. Figure 4.1 gives a concrete example of how concave time curves such as Local GP's can come about in Matlab. This behavior is unfortunate, as we will see that Local GP appears to be an interesting approximation method in terms of error performance. Implementing optimized Local GP in a lower level language like C could be very useful. However, Local GP is only suited for some applications, which do not require continuity of the regressed function; thus it is useful to have Local GP available in a more general GPR package, and the optimized C implementation would best come with other flavors of GP approximations as well.

4.3.3 IFGT complexity

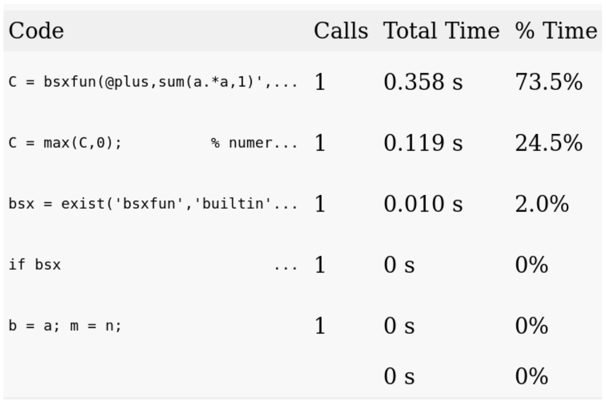
We have not included IFGT complexity in our tables. This is because IFGT does not have one parameter that controls its runtime and space requirements and thus comparison to the other methods in one table would be difficult. As in any iterative method, one can reduce the number of iterations k (of CG in this case) to be smaller than N, the dimensionality of the CG problem we want to solve. In this case, IFGT requires $O(kN^2)$ training time. More interestingly, one might also increase the allowed MVM error on each CG iteration; this would further speed up the process. Raykar and Duraiswami [2007] shows that the maximum MVM error can grow with CG iterations and can be automatically inferred from the CG residuals, given a parameter δ which controls the final accuracy of the CG procedure (see Equation 16 in Raykar and Duraiswami [2007]). We employ the automatic procedure as suggested in the paper, and hence it is difficult if not impossible to say exactly what the actual time complexity of the method will be. Raykar and Duraiswami [2007] shows that asymptotically CG with progressively more inexact IFGT takes $O(N)$ time for training; in our experience, however, there are no actual speedups present on real datasets, and the behavior of the algorithm is sensitive to additional factors. These are analyzed in more detail in Chapter 6.



Code	Calls	Total Time	% Time
`C = bsxfun(@plus,sum(a.*a,1)',...`	2048	0.192 s	46.3%
`mu = mean(a,2);`	2048	0.131 s	31.7%
`bsx = exist('bsxfun','builtin'...`	2048	0.030 s	7.3%
`C = max(C,0);          % numer...`	2048	0.010 s	2.4%
`if bsx                        ...`	2048	0.010 s	2.4%
		0.040 s	9.8%



Code	Calls	Total Time	% Time
`C = bsxfun(@plus,sum(a.*a,1)',...`	128	0.022 s	66.7%
`mu = mean(a,2);`	128	0.011 s	33.3%
`C = max(C,0);          % numer...`	128	0 s	0%
`if bsx                        ...`	128	0 s	0%
`b = a; m = n;`	128	0 s	0%
		0 s	0%



Code	Calls	Total Time	% Time
`C = bsxfun(@plus,sum(a.*a,1)',...`	1	0.358 s	73.5%
`C = max(C,0);          % numer...`	1	0.119 s	24.5%
`bsx = exist('bsxfun','builtin'...`	1	0.010 s	2.0%
`if bsx                        ...`	1	0 s	0%
`b = a; m = n;`	1	0 s	0%
		0 s	0%

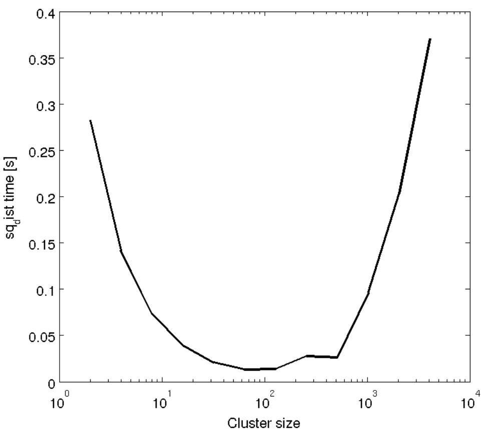


Figure 4.1: Time profiles for running sq_dist, a simple function included with and used by the GPML code, that computes squared Euclidean distances between each pair of elements in a vector. **Top Left** `sq_dist` run on vectors of size 2, 2048 times. **Top Right** 128 vectors of size 32. **Bottom Left** `sq_dist` run on a 4096 element vector once. **Bottom Right** As in Local GP, for each "cluster size" m we run `sq_dist` N/m times. Asymptotically this requires $\frac{N}{m}m^2 = Nm$ operations, so the plot should be linear. Instead, for small cluster sizes non-asymptotic factors dominate the computation time. `bsxfun` is the quadratic operation in this code. For smaller clusters, where all operations must be repeated many times, there is much overhead from setting up the functions' namespaces, stacks etc. That's why in absolute terms, 2048 runs of `bsxfun` on 2-vectors in the first table take almost ten times as much time as 128 runs on 32-vectors in the second table do.

4.4 Clustering and Inducing Points

We have implemented both simple clustering algorithms, RRC and RCP (described in Section 3.5) in Matlab. We tried to make the code simple to ensure it's bug-free, as opposed to optimizing its efficiency, as clustering data using these simple approaches takes negligible time compared to relevant matrix inversions or hyperparameter optimizations.

Section 3.6 describes several methods of choosing the inducing points set. We implemented the trivial random choice. We also implemented the entropy maximizing data subset choice, described in (Rasmussen and Williams [2006] Section 8.3.3) efficiently. The third method of choosing inducing points we use is picking cluster centers returned by the Farthest Point Clustering algorithm. This algorithm is also used in the IFGT package (Morariu [2010]); we integrated that implementation into our code.

4.5 Tests and additional scripts

We have also written scripts to perform testing of the algorithms and scripts to extract and present data gathered during these tests. In addition, many small experiments needed to be implemented to understand some behaviors of the algorithms (described later in this thesis). All this is mostly our own code, with the exception of dimensionality reduction code implementing the `t-SNE` algorithm (see van der Maaten and Hinton [2008]) which was implemented in the Matlab Toolbox for Dimensionality Reduction (van der Maaten [2010]) by Laurens van der Maaten.

Chapter 5

Empirical Comparison Setup

The main goal of our work is to compare the practical usefulness of the GP approximations described in Chapter 3. However, it is not immediately clear what the best way to compare approximations like this should be. We wanted to focus on the practical, not theoretical characteristics of the algorithms, hence first of all we needed to decide which **datasets** to use in our tests. We tried to pick data with different characteristics and of various difficulties. Our choices are discussed in Section 5.1. Once the data is chosen, several different bases of comparison come to mind.

One might care about how well each method approximates the full Gaussian Process– the word "approximation" suggests it would be good to get as close to the exact solution as possible. However, it is not obvious this would actually be a desired behavior in itself. We do not expect real data to be drawn from a Gaussian Process, so the full GP is an approximation itself, which we can only hope to be good if we have enough domain knowledge about the problem to specify good covariance function structures. Hence one might imagine that an approximation to the full GP could possibly do better than the full algorithm itself, if it broke some wrong assumptions present in the latter. As we will see in the next chapters, some approximations can in fact possess more flexibility than the original algorithm.

A different idea, which we decided to follow, is to look at some kind of performance measure of the algorithms. We decided to focus on **Standardized Mean Squared Error** and **Mean Standardized Log Loss** described in Section 5.2.

The four different approximations we analyze can be controlled for the accuracy vs. time/space complexity tradeoff. SoD, Local GP and FITC each use a parameter consistently called m in literature (and in our Chapter 3); IFGT can control the magnitude of the allowed MVM multiplication error. Since each approximation can in theory control the achievable errors up to some degree, it is still not clear how to use error measures to perform a comparison. We have decided to look at **time vs. error plots**. We can vary the value of the complexity parameter in each algorithm and do regression using these values, keeping track of algorithm runtimes; we then measure the error in prediction on a test set, as described below. We can then plot the time it took each algorithm to achieve resulting error values on each dataset. Plotting such performance of all algorithms on the same plot for each dataset and error measure should allow for a fair comparison of their usefulness. However, there might be different kinds of runtime the user cares about. Section 5.3 examines this issue.

5.1 Datasets

We want to use datasets large enough to pose a computational challenge for the GP approximations. The sets we chose are of differing dimensionalities and difficulties (more/less nonlinear, more/less uniformly sampled). We describe those datasets in some detail in this section.

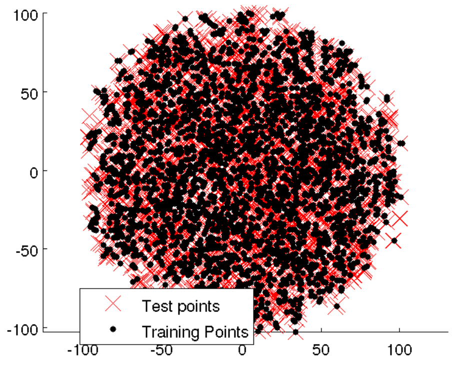
SYNTH2/SYNTH8 Synthetic data created by Carl Edward Rasmussen, generated from a GP with zero mean and isotropic Squared Exponential covariance (lengthscale one in each direction) in 2 and 8 dimensions respectively. We split the data at random into 30543 training points and 30544 test points. SYNTH2 has noise variance 10^{-6}, SYNTH8 was originally noiseless but we added 0.001 variance independent Gaussian noise to it. In both cases the training input locations are sampled from a Gaussian distribution. Figure 5.1(a) shows SYNTH8 embedded in two dimensions. It is evident that the training and test points cover the space well, with no obvious structure. The same is true of SYNTH2 (not shown in the Figure). SYNTH2 is the easiest dataset we use. Its low dimensionality and GP origin make it perfectly suited for GP regression.

The situation is more complicated in 8 dimensions, even though our model will fit the data well. The generating GP's lengthscales are all equal to 1. Consider a point far away from the origin, say on the surface of the hypersphere with radius 1 centered at the origin. In eight dimensions, this hypersphere intersects much smaller proportion of the high-density data regions than it would in the two dimensional space. As a result, the further a point is from the center of the sampling distribution the faster its regression difficulty will grow with dimensionality: the GP algorithms will not be provided with enough data from those regions of input space in which the outputs are well correlated with the test point. The fact that in SYNTH datasets the sampling is Gaussian, so there is much less data on the outskirts of the distribution than on the easier inside region, makes the situation even more complicated.

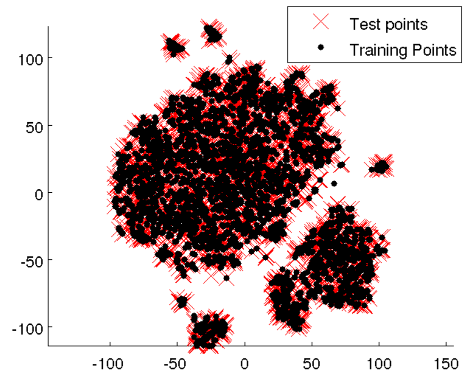
CHEM consists of physical simulations data related to electron energies in molecules, provided by Tucker Carrington from Queen's University (Canada). Thanks to Iain Murray for help with access to this set. The inputs have 15 dimensions (the outputs are scalar as usually), and we split the data into 31535 training cases and 31536 test cases. CHEM consists of one large cluster of input points and several smaller ones added for coverage (Figure 5.1(b)). This dataset was sampled in a complicated way and in several stages involving both analytic calculations and neural network simulation (see Malshe et al. [2007]).

SARCOS is a dataset representing the inverse kinetics of a robotic arm. The inputs have 21 dimensions and we use 44484 training points and 4449 test points. This dataset is often used in literature to assess regression algorithm's performance (for example in Rasmussen and Williams [2006] or Raykar and Duraiswami [2007]; also see Nguyen-tuong et al. [2008] for a more general treatment of inverse dynamic problems and Machine Learning). SARCOS is freely available on the GPML website[1]. A projection in two dimensions, shown in Figure 5.1(c), suggests that SARCOS's inputs are sampled along disconnected trajectories of the robotic arm. Figure 5.1(d) also shows one view of a three dimensional embedding of SARCOS data. The 3D embedding, when looked at from different angles, clearly consists of many disconnected, straight paths. This disconnected sampling structure can influence the way the problem is learned significantly. As we will see, the results of running different approximations on SARCOS will be interesting, most likely partially due to its very structured input space.

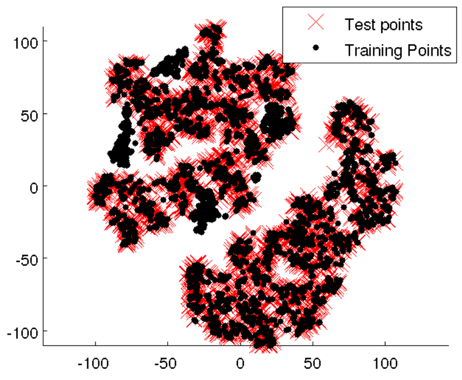
[1]`http://www.gaussianprocess.org/gpml/data` as of March 2011.



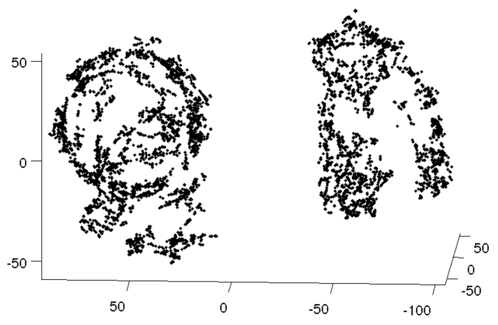
(a) SYNTH8



(b) CHEM



(c) SARCOS, 2D projection



(d) SARCOS, 3D projection (only training inputs)

Figure 5.1: Low dimensional embedding of the datasets. We used the t-SNE algorithm (see van der Maaten and Hinton [2008]) for embedding. SYNTH8, CHEM and SARCOS are shown in 2D. Plot (d) shows SARCOS in 3D viewed from an angle that emphasizes many separate trajectories. SYNTH2, sampled in 2D from a Gaussian, is not shown.

5.2 Error Measures

We use two distinct error measures to assess algorithms' performance. One of them, SMSE, takes only the mean prediction into account and can be used with most regression algorithms. The second measure, MSLL, also looks at the predictive variances: intuitively, if an algorithm makes an error and is very confident in this prediction we would like to penalize it more than when it makes an error but admits wider predictive variance. These two measures are also standardized to make them invariant to data input/output variances. A more formal description follows, in which we use notation $\mathcal{D}$ to symbolize the training data used to compute the predictive mean and variance $\bar{f}(\mathbf{x}_{*,i})$ and $\sigma^2_{*,i}$ on the test point $\mathbf{x}_{*,i}$. σ^2_{test} and σ^2_{train} indicate the variance of the test and training set outputs respectively; μ_{test} and μ_{train} for the respective mean values; and $(\mathbf{x}_{*,i}, y_{*,i})$ is a test input/output pair and $(\mathbf{x_i}, y_i)$ is a training input/output pair. On some occasions we will also denote the mean using a bar to avoid using confusingly many angle brackets, e.g. $\bar{y}$ is the mean training output and $\bar{y_*}$ is the mean test output.

Standardized Mean Squared Error The usual Mean Square Error (often used instead of an absolute error as it is differentiable) is

$$\langle (y_{*,i} - \bar{f}(\mathbf{x}_{*,i}))^2 \rangle, \tag{5.1}$$

the average of squared differences between the test output values and the predicted mean value $\bar{f}(\mathbf{x}_{*,i})$. We want to normalize this so that our error measure is invariant to output scaling. Imagine a trivial method of guessing the output values - fixing all test values to the mean training output values, i.e. $\bar{f}(\mathbf{x}_{*,i}) = \bar{y_j}$ for all i. Instead of computing the "absolute" MSE, we can normalize the quantity from 5.1 by the MSE achieved by such trivial guessing method. This normalizing MSE is

$$\begin{aligned} \text{NMSE} &= \langle (y_{*,i} - \bar{y})^2 \rangle \\ &= \langle (y_{*,i} - \bar{y_*} + \bar{y_*} - \bar{y})^2 \rangle \\ &= \langle (y_{*,i} - \bar{y_*})^2 \rangle - 2(\bar{y} - \bar{y_*})\langle y_{*,i} - \bar{y_*} \rangle + \langle (\bar{y} - \bar{y*})^2 \rangle \\ &= \sigma^2_{test} + (\bar{y} - \bar{y_*})^2 \end{aligned} \tag{5.2}$$

If our datasets are normalized to zero output mean $\bar{y} = 0$, then we have SMSE $= \frac{\text{MSE}}{\text{NMSE}} = \frac{1}{\sigma^2_{test} + \mu^2_{test}} \langle (y_{*,i} - \bar{f}(\mathbf{x}_{*,i}))^2 \rangle$.

Mean Standardized Log Loss To include the predictive variances in error assessment we can look at the negative log likelihood of y_* given the training and test data and the Gaussian noise assumption (we use negative likelihood to obtain a loss, as higher likelihood indicates a better fit so smaller error/loss):

$$-\log p(y_*|\mathcal{D},\mathbf{x}_*) = \frac{1}{2}\log(2\pi\sigma_*^2) + \frac{(y_* - \bar{f}(\mathbf{x}_*))^2}{2\sigma_*^2}. \tag{5.3}$$

Again, we want to normalize this quantity. This time our trivial prediction method is to use the mean training output value as the constant mean prediction, and training output variance σ^2_{train} as the predictive variance (we use the same values for each test prediction). This model would give the following Mean Log Loss:

$$\begin{aligned} \text{NMLL} &= \frac{1}{2}\log(2\pi\sigma^2_{train}) + \frac{\langle(y_* - \bar{y})^2\rangle}{2\sigma^2_{train}} \\ &= \frac{1}{2}\log(2\pi\sigma^2_{train}) + \frac{\sigma^2_{test} + (\bar{y} - \bar{y_*})^2}{2\sigma^2_{train}}, \end{aligned} \tag{5.4}$$

where 5.4 follows similarly to 5.2. We can use this loss as a normalizing constant; the Mean Standardized Log Loss compares the actual loss under the model to this trivial model's loss. If the training variance is 1 and training mean 0, then MSLL has the simple form

$$\begin{aligned} \text{MSLL} &= -\langle\log p(y_{*,i}|data,\mathbf{x}_*)\rangle - \text{NMLL} \\ &= \frac{1}{2}\langle\log(\sigma^2_{*,i}) + \frac{(y_{*,i} - \bar{f}(\mathbf{x}_{*,i}))^2}{\sigma^2_{*,i}}\rangle - \frac{(\sigma^2_{test} + \mu^2_{test})}{2}. \end{aligned} \tag{5.5}$$

5.3 Time Measures

Chapter 2 suggests that there are several distinct time measures of GPR that one might care about. First of all, we might have a limited budget of time altogether, in which case the **hyperparameter training time** will be of most concern. To train the hyperparameters, one will typically run a gradient descent algorithm on the negative log likelihood of the model; each computation of this likelihood will take $O(N^3)$ time (Chapter 2 gives more details on this and other theoretical properties of GP mentioned in this section). In addition, it could be beneficial to optimize the hyperparameters several times starting from different initial values to avoid bad local minima. If we have enough time available for the hyperparameters to be of no serious concern, computing the parameters- that is, inverting the covariance matrix- will not pose a challenge, as this **training time** is roughly as long as one iteration of likelihood-optimizing gradient descent.

However, one might also imagine a situation where fast online predictions must be made, that is the **test time** needs to be short, while the training time budget is of less concern (online exploration in the bandit setting comes to mind, see Srinivas et al. [2010] for theory). The test time scales differently than the training time, taking $O(N^2)$ for computing the predictive variances if the relevant Cholesky factor is precomputed.

Taking this into account, we decided to look at the **hyperparameter training times** and **test times per test point** of the approximation algorithms. As parameter training times are roughly proportional to hyperparameter training times, they don't introduce any additional useful information for the purposes of inter-algorithm comparison.

5.4 Theory vs. Practice

As explained above, we decided to base our comparison on hyperparameter training/test time vs. SMSE/MSLL graphs. Chapter 4 notes that the practical runtimes of approximations can differ significantly, even if they are asymptotically the same; hence we feel that using actual runtimes as a basis for comparison makes sense. All our tests are run on the same machine, and force Matlab to use only one CPU core.

One might argue that it is never clear if the approximations are implemented as efficiently as possible. Indeed, it is conceivable that our comparison is unfair in that we use a much less efficient implementation of for example FITC than IFGT. However, all of SoD, Local GP and FITC implementations we look at are written in Matlab; while IFGT is uses C++ modules, we will see that it is not a competitive algorithm anyway. Since SoD, Local GP and FITC all use common base code from the GPML package, we might say with some confidence that our comparison framework is not terribly biased.

Chapter 6

Results

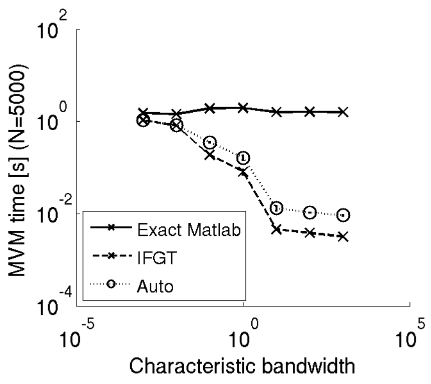
This chapter presents the main results of our work: time-performance plots comparing the different approximation methods on several datasets. Section 6.1 treats the IFGT approximation separately, as it turned out it is not feasible to use it on most datasets – it isn't amenable to our comparison methods. Section 6.2 looks at how the approximations behave when we vary their complexity parameter m (inducing set size or cluster size). This will help us choose the best variant of each method (for example, which clustering method is best suited for use with Local GP on SARCOS). We can also expect that looking at this m-dependent behavior can help us understand the results of Section 6.3, which discusses the time-performance plots of the approximations. This section presents the most important practical result in this dissertation. Finally, Section 6.4 discusses the significance and consequences of our findings and concludes with recommendations on which approximations to use depending on the situation.

Occasionally we refer to hyperparameter tables in this chapter. These can be found in Appendix A. The Appendix shows mean hyperparameter values (lengthscales for each dimension, noise variance, signal amplitude) estimated by each algorithm on each dataset in our experiments, with standard errors also shown. We don't comment on every table explicitly in this thesis; we decided to provide the full tables as they can potentially contain interesting information that we never use in our analysis of the algorithms.

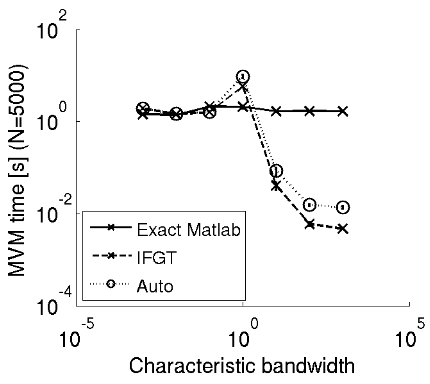
6.1 Problems with IFGT

As Iain Murray suggested in Murray [2009], it is not clear whether IFGT can be of use in practical hyperparameter regimes. In particular, the method is likely to bring speedups only for impractically long data lengthscales, with useful bandwidths scaling as $\sqrt{d}$ with data dimensionality. We wanted to check if indeed the method cannot give speedups on our datasets. To do this, we use IFGT to multiply the Squared Exponential covariance matrix on randomly chosen 5000 training points with a 5000 element vector of random i.i.d. elements (sampled from $\mathrm{U}[0,1]$), for each of the four datasets. Figure 6.1 shows that the method is practical only for the 2D SYNTH2 data. Already for 15d CHEM, the bandwidth of the data would have to be of the order of 100 for IFGT to be of use. We don't know the real hyperparameters of the data, but estimating these with the other methods strongly suggests much shorter lengthscales (on the order of 1).

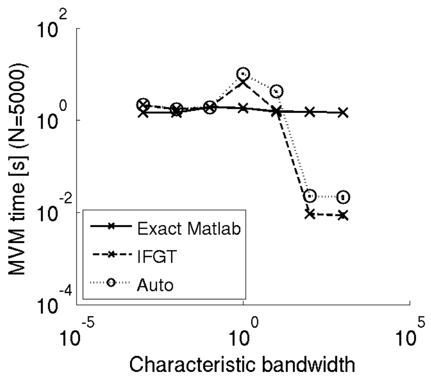
We tried running GPR using IFGT with progressively more inexact MVM's on SYNTH2. We have found that it was difficult for the algorithm to converge. Because of this we stopped the run after 1000 iterations. We also wanted to check whether any iterative method could possibly be useful on more serious data, like SYNTH8. Figure 6.1 shows that IFGT offers no speedups for SYNTH8, hence we used exact MVM with Conjugate Gradient. The advantage of this method over standard GP is that it does not keep the whole covariance matrix in memory, so CG – in theory – allows one to perform exact GP regression notwithstanding the size of the problem, given unlimited computation time budget. Because of time constraints we stopped the algorithm after 10000 iterations. In both cases (SYNTH2 and SYNTH8) we set the hyperparameters to their exact correct values. We did not compute the predictive variances (Section 4.2 explains why this would be difficult). Our results are presented in Table 6.1. The table shows that the MVM methods can give reasonable SMSE values on low dimensional data. In addition, compared to the other methods analyzed in the reminder of this chapter the truncated CG approximation gave the best MSE on SYNTH8. However, the result is of little practical importance as it took over five days to run the algorithm, with pre-set hyperparameters.



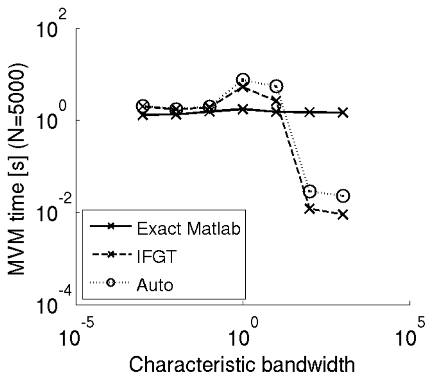
(a) SYNTH2



(b) SYNTH8



(c) CHEM



(d) SARCOS

Figure 6.1: IFGT used for fast matrix vector multiplication at different bandwidths. In all plots, both axes are logarithmic. In each case we chose 5000 points from the dataset at random and multiplied the Squared Exponential covariance matrix on these points with a random (uniform on $[0, 1]^{5000}$) vector $\mathbf{v}$. We used three MVM methods on all sets: Exact Matlab where the whole matrix was stored in memory and multiplied by $\mathbf{v}$ using Matlab matrix multiplication (the complexity of which is, as expected, independent of data dimensionality), IFGT where the Improved Fast Gauss Transform was used and Auto which was proposed by Raykar and Duraiswami [2007] as a way to speed up the IFGT MVM in some regimes. The plots show no difference between the latter two, apart from a small constant overhead. The plots seem to confirm that IFGT can only be useful in low dimension or at large lengthscales.

Dataset	SYNTH2	SYNTH8
MSE	0.2346	0.1845
Time [s]	12396 (approx. 4 hours)	434420 (approx. 5 days)

Table 6.1: GP using CG. For SYNTH2, we used IFGT for approximate MVM. The Conjugate Gradients algorithm had difficulties converging. For SYNTH8 we used exact MVM but stopped CG after $10,000$ iterations.

6.2 SoD, Local GP and FITC: performance as a function of m

This section examines the approximations' behavior as we vary their complexity, controlled by a parameter we call m in each case. We later use these results to choose particular implementations of the algorithms for the time-wise comparison in Section 6.3. Note: in all the plots shown in this section the x-axis uses the log scale. In all the SMSE plots, the y-axis also uses the log scale, but all the MSLL plots have linear y-axes.

6.2.1 Best expected performance

SYNTH2 and SYNTH8 are datasets created from Gaussian Processes with known hyperparmaters. Because we know the data noise levels σ_n^2, we can calculate the best possible expected performance of any regression method on these data. As explained in Chapter 2, GP Regression predicts the value $\hat{f}(\mathbf{x}_*)$ of a function f on a test point $\mathbf{x}_*$. We measure approximation error by comparing these estimates to test outputs y_*; where $y_* = f(\mathbf{x}_*) + \mathcal{N}(0, \sigma_n^2)$ if we assume Gaussian observation noise (which we do in GPR). Thus even if our regression method knows exactly what the function f is (so that $\hat{f}(\mathbf{x}_*) = f(\mathbf{x}_*)$) and reduces predictive variances to noise levels $\sigma_*^2 = \sigma_n^2$, we still expect the errors to be nonzero:

$$\begin{aligned}
\text{SMSE}_{best} &= \frac{1}{\sigma_{test}^2 + \mu_{test}^2} \left\langle (y_{*,i} - \hat{f}(\mathbf{x}_{*,i}))^2 \right\rangle \\
&= \frac{1}{\sigma_{test}^2 + \mu_{test}^2} \left\langle (f(\mathbf{x}_*) + \mathcal{N}(0, \sigma_n^2) - f(\mathbf{x}_{*,i}))^2 \right\rangle \\
&= \frac{1}{\sigma_{test}^2 + \mu_{test}^2} \left\langle (\mathcal{N}(0, \sigma_n^2))^2 \right\rangle \\
&= \frac{\sigma_n^2}{\sigma_{test}^2 + \mu_{test}^2}, \qquad (6.1)
\end{aligned}$$

$$\begin{aligned}
\text{MSLL}_{best} &= \frac{1}{2} \left\langle \log(\sigma_{*,i}^2) + \frac{(y_{*,i} - \hat{f}(\mathbf{x}_{*,i}))^2}{\sigma_{*,i}^2} \right\rangle - \frac{(\sigma_{test}^2 + \mu_{test}^2)}{2} \\
&= \frac{1}{2} \left\langle \log(\sigma_n^2) + \frac{\sigma_n^2}{\sigma_n^2} \right\rangle - \frac{(\sigma_{test}^2 + \mu_{test}^2)}{2} \\
&= \frac{\log(e\sigma_n^2)}{2} - \frac{(\sigma_{test}^2 + \mu_{test}^2)}{2}. \qquad (6.2)
\end{aligned}$$

Above, we derive Equation 6.2 using Equation 6.1 to reduce the squared-error factor in the log loss.

Thus we expect any method to yield Standardized Mean Squared Errors proportional to the noise variance at best, and Mean Squared Log Loss proportional to the log of noise variance. In the approximation performance figures below we computed these baseline values for SYNTH2 and SYNTH8 and note them in relevant figure captions for comparison with actual method performances.

6.2.2 SoD

As explained in Chapter 3, running the SoD approximation requires choosing a method to pick a data subset representative of the full dataset; we experimented with choosing the subset using the Farthest Point Clustering algorithm and at random. We varied the subset size to be $m = 2, 4, \dots, 4096, 6000$ (using $m = 8192$ was problematic due to memory and time constraints, but we did run this test in several particularly interesting cases) and repeated the experiments five times for each inducing points choice method (standard error bars are shown in the plots). We only report results for $m \geq 128$ as the error bars for smaller m were very large. The description of each algorithm's performance on each dataset is accompanied by pointers to tables showing the hyperparameters the method estimated on this dataset. We found the following facts useful in understanding SoD's behavior.

SYNTH2 (Tables A.1, A.2) As will be the case for all other methods, SYNTH2 was an easy dataset to learn. The tables show that the hyperparameters are learned almost perfectly, with very little variance over the trials. The plots in Figure 6.3 show a steadily falling learning curve for both Random dataset selection and the FPC algorithm. The MSLL performance saturates after using 256 inducing points, which constitutes $\frac{1}{128}$ of all data available. Note that Random never reaches FPC's MSE level. This behavior was already explained in Figure 3.3.

SYNTH8 (Tables A.3, A.4) This eight-dimensional dataset is very difficult to learn. Table A.3 shows that the hyperparameter estimation is possible with 256 inducing points already, but MSE and MSLL are far from optimal. In this case Random does slightly better than FPC at estimating the hyperparameters and gives slightly smaller errors. Figure 3.3 suggests that this is because in high dimensions FPC places too many points on the exterior of the sample distribution. As in SYNTH2 experiments, the learning curves have negative slopes as expected, but this time error reduction is much slower.

CHEM (Tables A.5, A.6) The CHEM dataset (Figure 6.5) behaves similarly to SYNTH8 in that Random does better than FPC clustering for most of the time. This suggests that there is no structure in the inputs that would make using FPC beneficial. Thus Random does better, as it simply places more inducing points in the denser data regions with high probability.

SARCOS (Tables A.7, A.8) For SARCOS FPC gives better results than Random clustering (Figure 6.6). Similar behavior is encountered when Local GP algorithm is used. In Section 5.1 we saw that there are strong indicators that SARCOS inputs are not sampled uniformly, but rather have a strong structure that can be well used by the GP approximations. As expected, FPC recognizes this structure better than random subset choice.

6.2.3 FITC

In FITC we can vary the number of inducing points m, which we try to choose using FPC or randomly, similarly to choosing SoD's active set. We varied the number of inducing points to be $m = 8, 16, \ldots, 512$. FITC is much more memory-intensive than SoD (see Table 4.1), so it was impossible to continue experiments with larger m. We repeated each experiment five times, and report standard error bars.

SYNTH2 (Tables A.9, A.10) FITC's performance on SYNTH2 reaches good values quickly, as expected (Figure 6.7). Note that the plots show decreasing performance for largest m. This is because of numerical instability present in FITC– discussed in Section 6.3.1 below. Ignoring the instability, randomly chosen inducing points yield worse results than FPC-chosen inducing points. We suggest that the reason is the same as in the case of SoD choice - Figure 3.3 shows that FPC places the inducing points densely in a regular pattern throughout the input space, while Random fails to do so.

SYNTH8 (Tables A.11, A.12) Also similarly to the SoD case we find that SYNTH8 is a difficult dataset (Figure 6.8). The learning curves decrease very slowly. An examination of the numerical values of the results shows that, as with SoD, FPC does slightly worse than Random in this case, but the difference is negligible. More interestingly, FITC estimates the SYNTH8 hyperparameter lengthscales to be slightly larger than the real values, and the noise levels are severely underestimated (both FITC-FPC and FITC-Random estimate the noise to be around $\exp(-3) = 0.05$ while SYNTH8 has noise with $\sigma_n^2 = 0.001$). Because the experiments were repeated five times with random seeds and the hyperparameter variances over those runs are relatively small, it is unlikely that they result from a local minimum. In fact, we confirmed experimentally that using FITC with the true hyperparameters gives slightly worse SMSE and MSLL than using the estimated hypers. We were unable to fully explain this phenomenon. However, it is a useful reminder that different approximation might have different optimal hyperparameters, and thus for example using the SoD method to optimize the hypers to then be used with another algorithm is not always the best idea.

CHEM (Tables A.13, A.14) Figure 6.9, showing FITC's performance on CHEM, again is very similar to Figure 6.5 which shows the performance of SoD on CHEM. Notably, Randomly chosen inducing points give better results for most m but for the largest values FPC seems to start outperforming the simpler method SMSE-wise.

SARCOS (Tables A.15, A.16) FITC's performance on SARCOS, shown in Figure 6.10, is also similar to that of SoD.

6.2.4 Local GP

Note on the experiments: Local GP as we implement it has four basic variants (RRC+Joint, RRC+Separate, RPC+Joint, RPC+Separate). Unfortunately, we did not manage to perform the full set of five experiment repetitions for each variant, as we did for all other methods; the full Local GP experiment set is still in progress. In the plots below we are able to present results from only one experiment per each variant of Local GP, hence any conclusions we reach are not very reliable; however, an overall trend can often be spotted, in particular when comparing Local GP with other methods later in Section 6.3. In addition, the full experiment set on SYNTH2 is already completed and the results are qualitatively no different from these presented below.

To use the Local GP approximations one must choose a clustering method. We focused on two simple alternatives, Recursive Projection Clustering (RPC) and Random Recursive Clustering (RRC), as described in Section 3.5. The results reported in this section suggest that there is no clearly visible difference between these algorithm's performance. However, choosing the hyperparameter optimization method to be either joint or separate (Section 3.2 explains this distinction) turns out to be a potentially significant choice. In each experiment, we varied cluster size to be $m = 32, 64, \ldots, 4096$.

SYNTH2 (Tables A.17 - A.20) Local GP learns SYNTH2 easily (Figure 6.11), already for small clusters of size 32. In one case, the RRC+separate, the algorithm returns unreasonable solutions MSLL-wise for both for SYNTH2 and SYNTH8. This is very likely due to the fact that *RRC* tends to estimate clusters of size smaller than the fixed upper bound (more so than RPC), and the separate hyperparameter estimation method is unlikely to be able to estimate reasonable hyperparameters in such small clusters. The hyperparameter tables confirm this explanation; in Tables A.17, A.18 (SYNTH2, Local with joint estimation) and A.21, A.22 (SYNTH8, Local with joint estimation) the hyperparameters are estimated almost perfectly even for clusters of size 32. On the contrary, the

"RPC+separate" Tables A.19 (SYNTH2) and A.23 show some misestimation of the hypers, while "RRC+separate" Tables A.20 (SYNTH2) and A.24 show hypers severely misestimated on average, with huge variances between the clusters.

SYNTH8 (Tables A.21 - A.24) Similarly to the SoD case we find that SYNTH8 is a difficult dataset (Figure 6.12). The learning curves decrease very slowly. Again, joint hyperparameter training works better with this dataset (see the explanation for SYNTH2's behavior above).

CHEM (Tables A.25 - A.28) Figure 6.13 shows that for CHEM separate hyperparameter training works better (MSLL-wise) than joint training when the clusters reach sizes above 256. This is probably because the dataset is taken from real world data; Local GP with separate hyperparameter training can handle nonstationary and highly nonlinear data easier than less localized methods. The performance difference is not visible when we look at the SMSEs. More trials might be necessary to understand the behavior of Local GP on CHEM properly.

SARCOS (Tables A.29 - A.32) Local GP performs very well on SARCOS, independent on cluster size (Figure 6.14). This agrees with the experiments conducted by Ed Snelson in his PhD thesis (Snelson [2001], Section 3.3.2) where it is remarked that the Local approximation, using only small block sizes, can achieve an extremely low MSE and MSLL. In our figure the joint hyperparameter training method beats separate optimization by far in the small clusters regime. This can happen for example if the dataset is highly nonsmooth so that it does not have a SE-ARD-GP-like structure (so it's hard to approximate with ordinary GP); but locally it is well approximated by smooth surfaces. If in addition these surfaces are sampled with similar noise levels, Local GP with small clusters and joint hyperparameters becomes local linear regression, and can learn the data well. Figure 6.2 shows that indeed SARCOS has traits of such a dataset. Tables A.29 and A.30 show that Local GP estimates SARCOS's hyperparameters to be very similar for all cluster sizes under joint estimation; but the same is *not* true for Local on CHEM, Tables A.26 and A.25. We were not able to fully understand this result, and we would like to look more experiment runs before drawing any definite conclusions about Local GP's behavior on SARCOS.

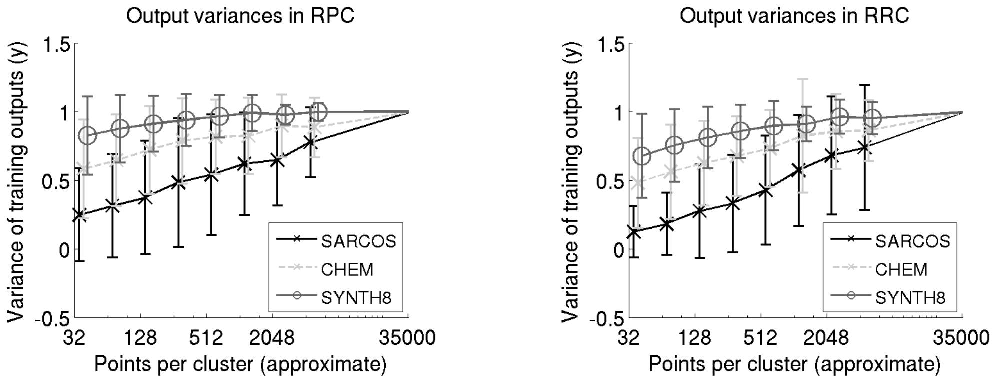


Figure 6.2: Variances of the outputs for clustered datasets. For each dataset, the training points $\mathbf{x}_1, ..., \mathbf{x}_n$ were clustered into m clusters $c_1, ..., c_m$. For each cluster, the variance of the test outputs associated with all the points in the cluster was computed. The plotted values are mean variances over all clusters (one standard deviation is shown). As usual, each dataset's training outputs are normalized to have unit variance. The plots show that SARCOS's clusters have relatively very low output variances, which might explain why Local GP gives good results on this dataset when tiny clusters with large lengthscales are used– in which case the algorithm approximates local linear regression.

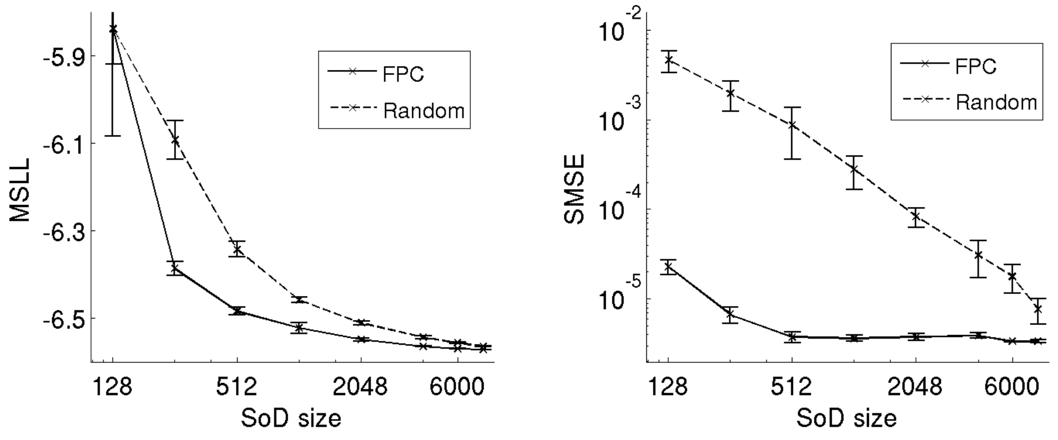


Figure 6.3: SoD: MSLLs and SMSEs for SYNTH2. The best expected possible performance, calculated according to Equations 6.1 and 6.2, is $SMSE_{best} = 9.96 \times 10^{-7}$, $MSLL_{best} = -6.9095$.

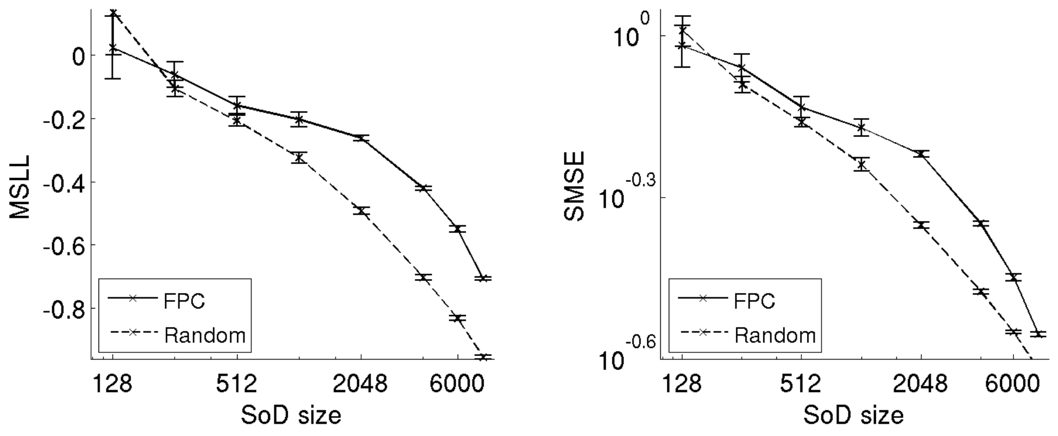


Figure 6.4: SoD: MSLLs and SMSEs for SYNTH8. The best expected possible performance, calculated according to Equations 6.1 and 6.2, is $SMSE_{best} = 0.001$, $MSLL_{best} = -3.449$.

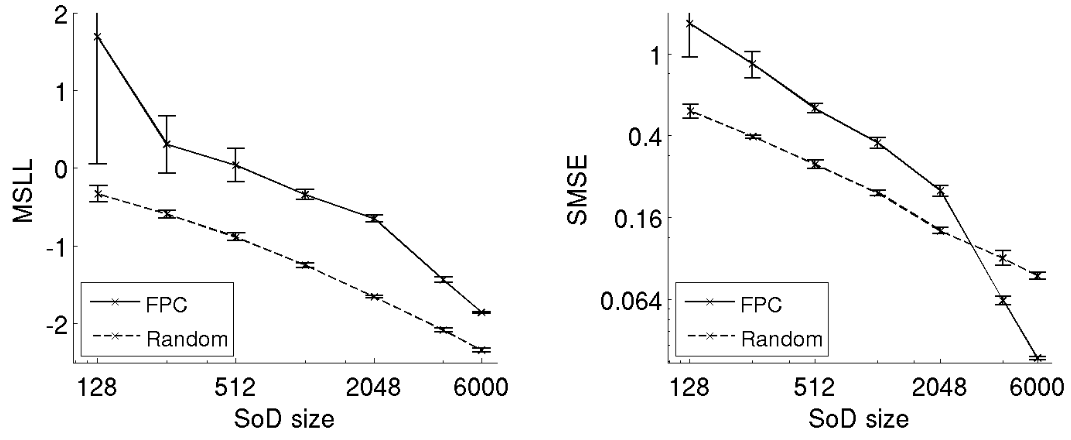


Figure 6.5: SoD: MSLLs and SMSEs for CHEM.

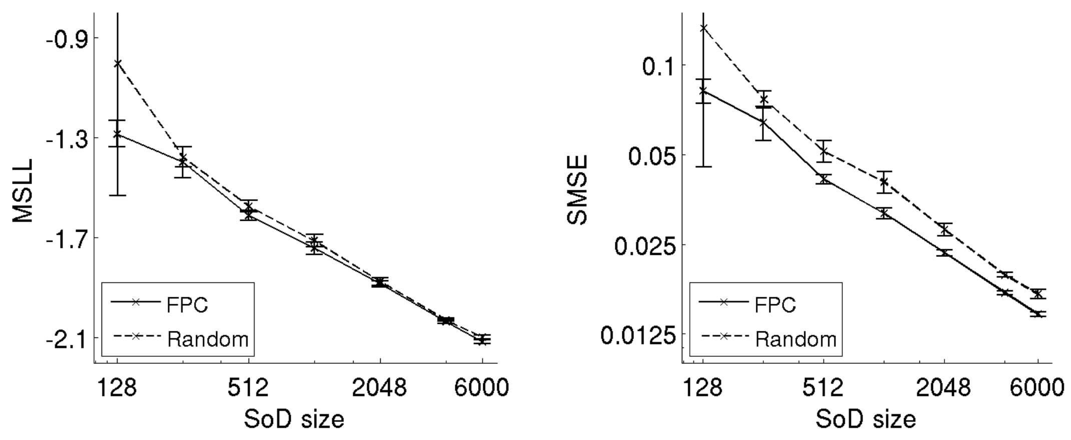


Figure 6.6: SoD: MSLLs and SMSEs for SARCOS.

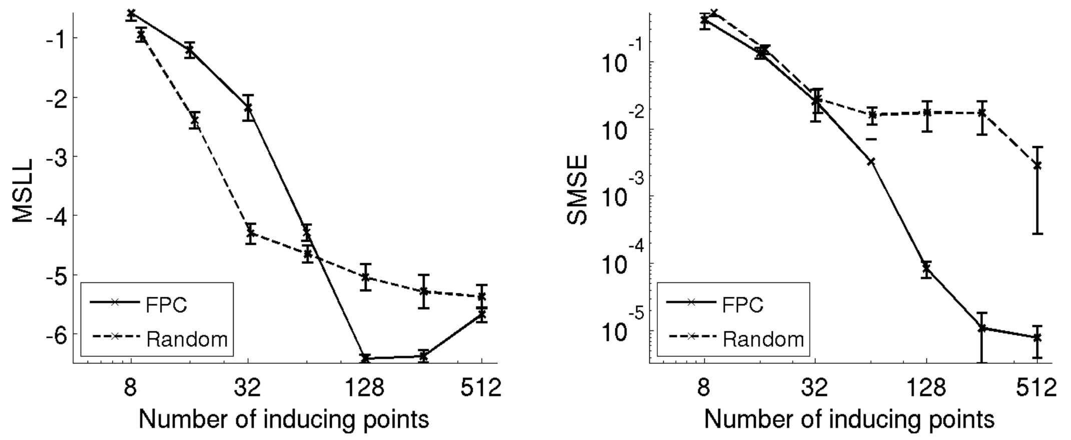


Figure 6.7: FITC: MSLLs and SMSEs for SYNTH2. The best expected possible performance, calculated according to Equations 6.1 and 6.2, is $SMSE_{best} = 9.96 \times 10^{-7}$, $MSLL_{best} = -6.9095$.

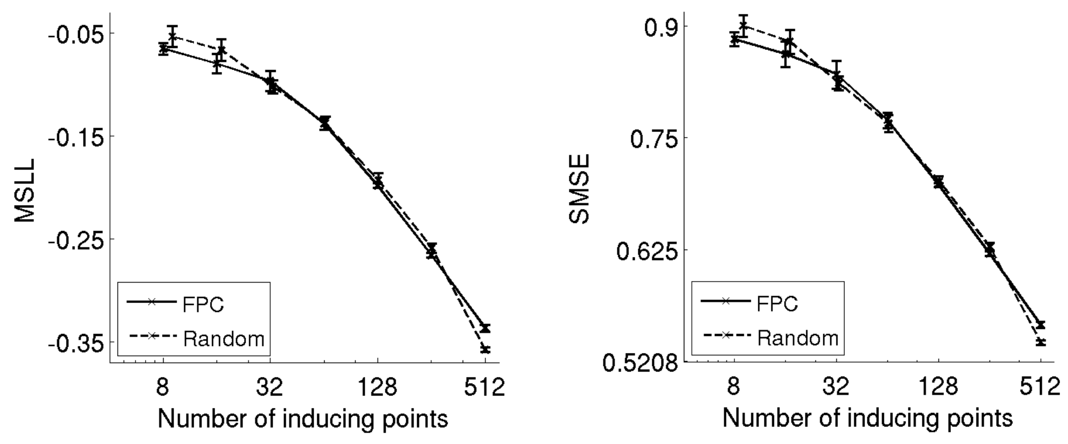


Figure 6.8: FITC: MSLLs and SMSEs for SYNTH8. The best expected possible performance, calculated according to Equations 6.1 and 6.2, is $SMSE_{best} = 0.001$, $MSLL_{best} = -3.449$.

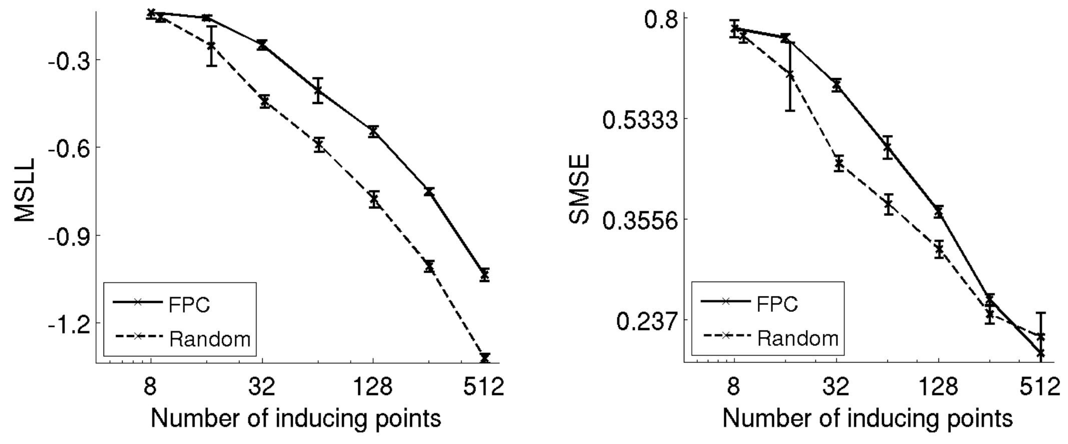


Figure 6.9: FITC: MSLLs and SMSEs for CHEM.

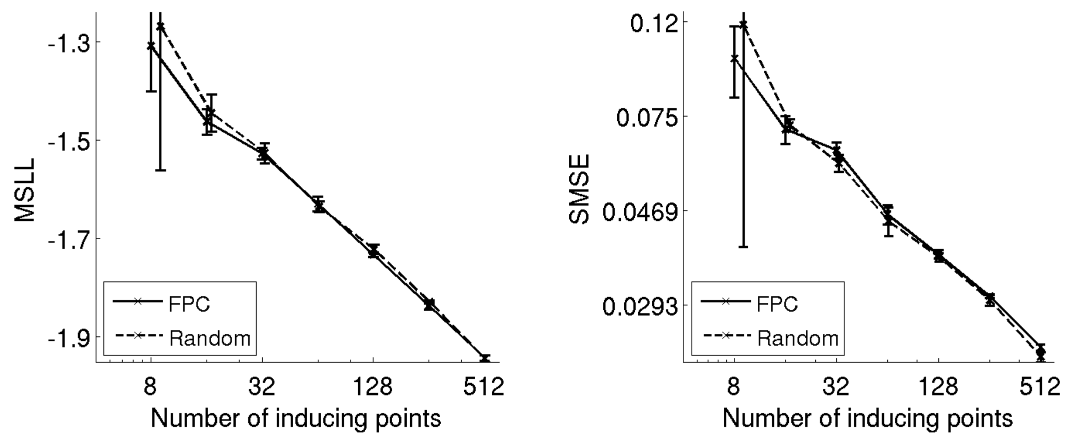


Figure 6.10: FITC: MSLLs and SMSEs for SARCOS.

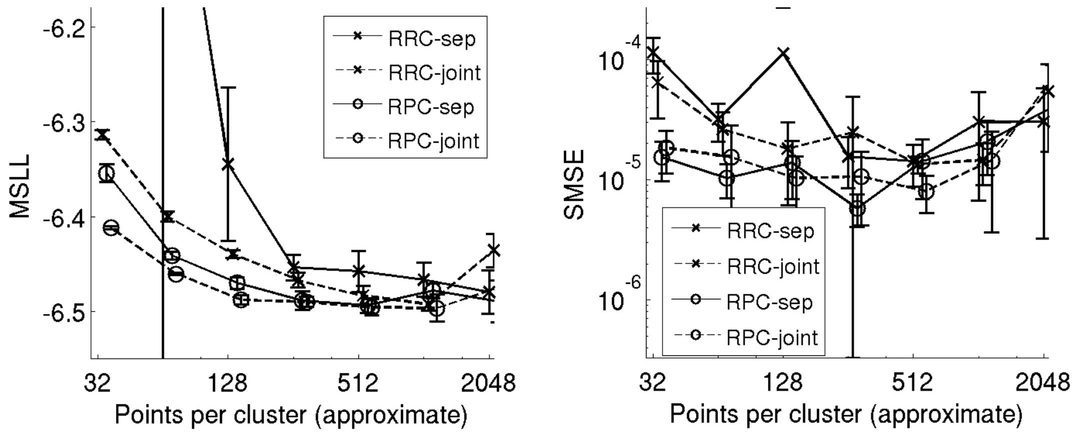


Figure 6.11: Local: MSLLs and SMSEs for SYNTH2. The best expected possible performance, calculated according to Equations 6.1 and 6.2, is $SMSE_{best} = 9.96 \times 10^{-7}$, $MSLL_{best} = -6.9095$.

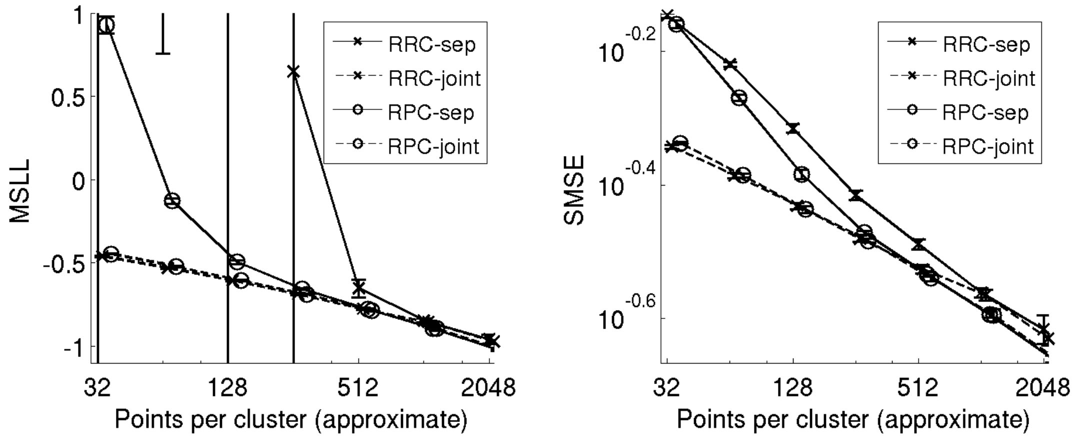


Figure 6.12: Local: MSLLs and SMSEs for SYNTH8. The best expected possible performance, calculated according to Equations 6.1 and 6.2, is $SMSE_{best} = 0.001$, $MSLL_{best} = -3.449$.

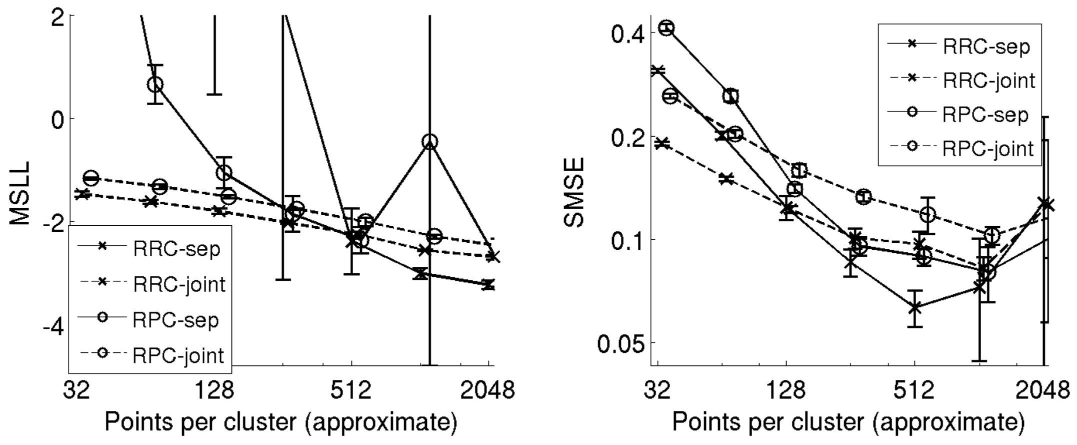


Figure 6.13: Local: MSLLs and SMSEs for CHEM.

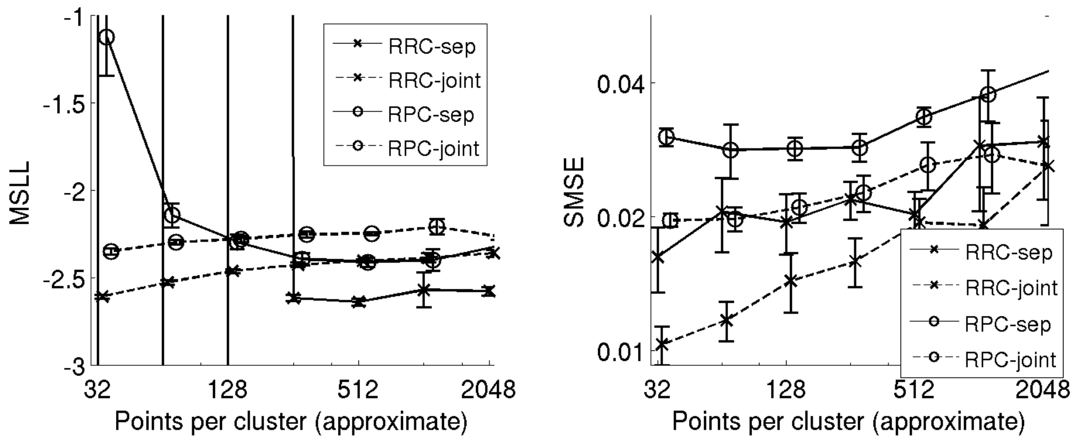


Figure 6.14: Local: MSLLs and SMSEs for SARCOS.

6.3 SoD, Local GP and FITC: time-wise comparison

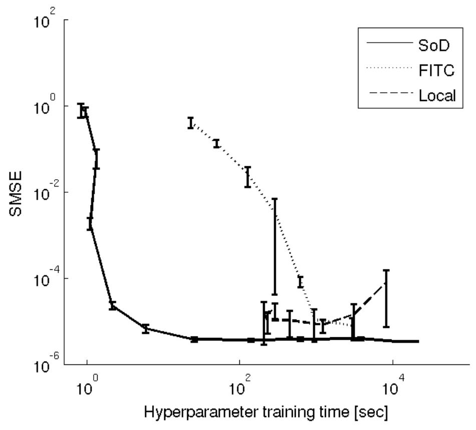
As explained in Chapter 5, we designed experiments showing how much time is needed to achieve given performances using the different approximations. The results presented in Section 6.2 helped us to choose which method variants to use. Table 6.2 lists those variants and briefly comments on the possible reason for why the choice yields best results.We tested the methods on the four datasets SYNTH2, SYNTH8, CHEM and SARCOS. The number of trials per each experiment and various *m*s we use are as described in Section 6.2 above. (Note again that the results for Local GP are not very reliable as only one trial was conducted for each variant of the method).

	SoD	**Local**	**FITC**
SYNTH2	*FPC* fills 2D space densely and regularly.	*Joint* fits data drawn from a GP.	*FPC* (as for SoD)
SYNTH8	*Random* focuses on the center of Gaussian sampled data.	*Joint* fits data drawn from a GP.	*Random* (as for SoD)
CHEM	*Random* finds denser regions in high dimensional data with little structure.	*Separate* fits nonstationary data.	*Random* (as for SoD)
SARCOS	*FPC* places inducing points in interesting regions of structured input space.	*Joint* possibly fits data well approximated by local linear regression.	*FPC* (as for SoD)

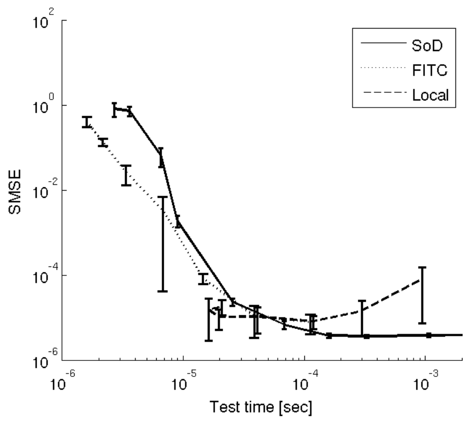
Table 6.2: Our choices of approximation flavors that give best performance on different datasets. *Joint* and *Separate* are two different ways of estimating hyperparameters in Local GP (the clusters can share the hyperparameters or use separate hypers); *FPC* and *Random* are two methods of choosing inducing points either as centers of Farthest Point Clustering clusters or at random (the table suggests that this choice has similar consequences for SoD and FPC). Two clustering methods we tested with Local GP (*RRC* and *RPC*) yielded very similar results, hence we don't mention them in the table.

6.3.1 SYNTH2

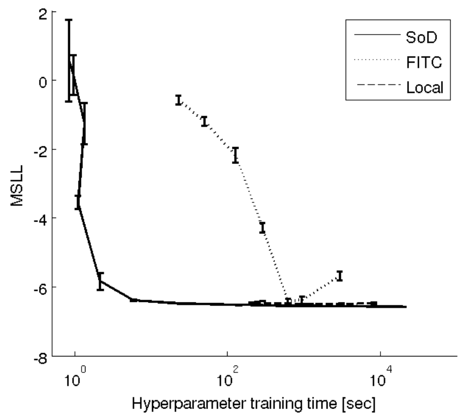
Figure 6.15 shows performance of the different algorithms on SYNTH2. SoD, the simplest method, saturates in performance quickly, and is definitely the best method when **hyperparameter training time** is important, in which case it achieves stable, good SMSE and MSLL values quickly. Note that in Figures 6.15(a) and 6.15(b) SoD's performance is not a *function* of time in the mathematical sense. This is because for small inducing point numbers (less than 100) the computational costs are defined by non-asymptotic factors, as Figure 4.1 exemplified. For similar reasons Local GP's SMSE curves are not functions of time. In the case of SYNTH2, for smaller clusters Local GP's performance was falling, but runtimes were growing; we exclude these datapoints from our plots but keep them in mind for later analysis. In the **test time** plots, FITC and Local GP do better than SoD. Note a significant and unexpected rise of FITC's MSLL at the last time point in Figures 6.15(c) and 6.15(d). This is due to a numerical instability in the algorithm. After its performance saturates, FITC has problems calculating log likelihood derivatives. Rasmussen and Nickisch were working on this problem in the newest GPML version (see Rasmussen and Nickisch [2010]); we have also looked at the issue but we don't see it as crucial for our work, as the algorithms will be unable to saturate in most non-trivial situations (such as on all the other datasets).



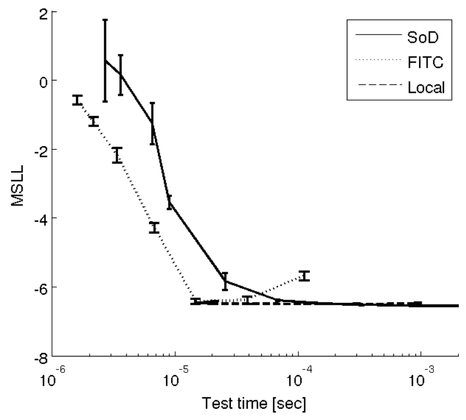
(a)



(b)



(c)



(d)

Figure 6.15: Time-performance plots for GP approximations running on SYNTH2 data. The best expected possible performance, calculated according to Equations 6.1 and 6.2, is $SMSE_{best} = 9.96 \times 10^{-7}$, $MSLL_{best} = -6.9095$.

6.3.2 SYNTH8

Figure 6.16 shows that the algorithms have much trouble working with this dataset; the error curves are very flat, indicating slow convergence to the true solution. However, the overall trends are similar: SoD gives relatively good results quickest when the **hyperparameter estimation times** are taken into account, while FITC and Local work better with **test times**. Note that Local GP's errors are smaller than those of the other methods; however, because of the non-monotone time behavior of the method it never runs quickly enough to compare with FITC or SoD in the short time ranges.

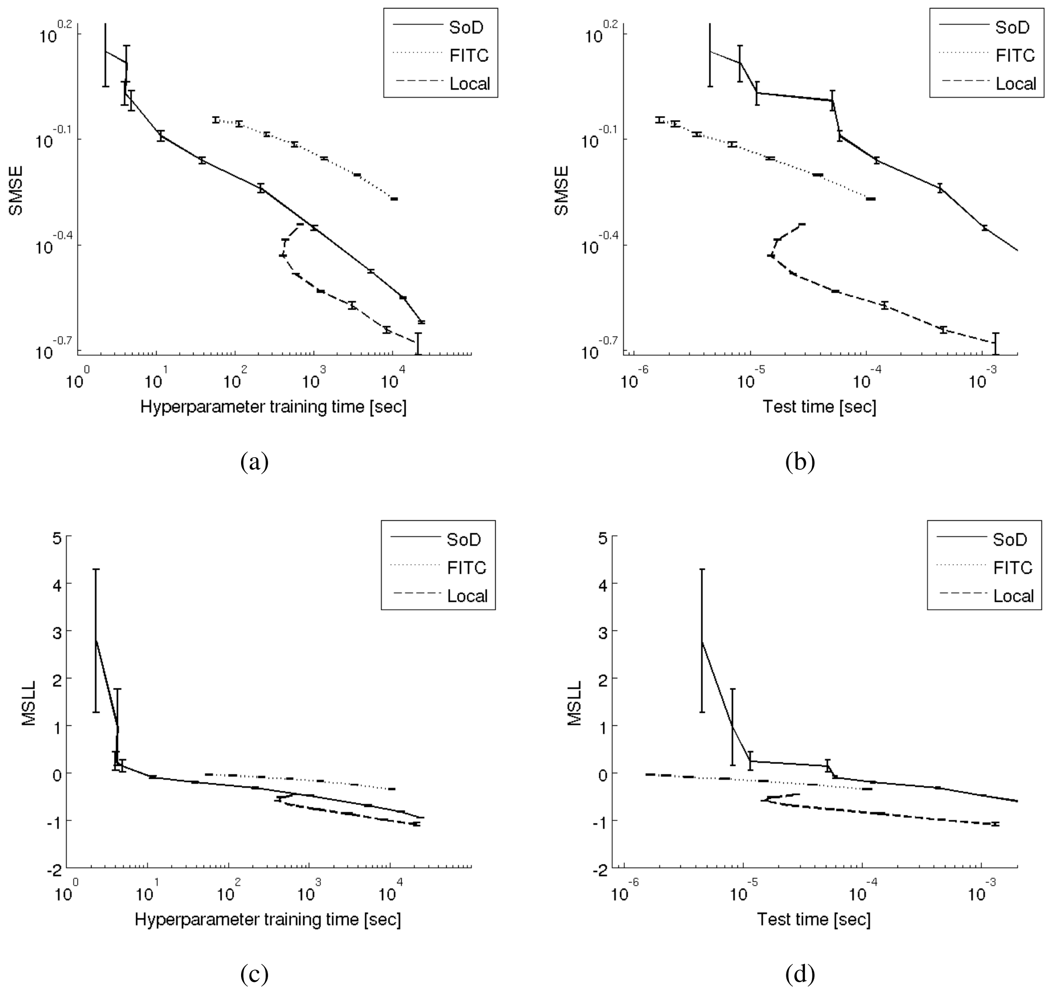


Figure 6.16: Time-performance plots for GP approximations running on SYNTH8 data. The best expected possible performance is shown as a gray constant line. The best expected possible performance, calculated according to Equations 6.1 and 6.2, is $SMSE_{best} = 0.001$, $MSLL_{best} = -3.449$.

6.3.3 CHEM

CHEM's plots (Figure 6.17) show trends similar to those seen on SYNTH8. Again, SoD does best on **training**, while FITC and Local GP are more useful during the **testing** phase. Local GP seems particularly promising given its test-time SMSEs, but we see that for small clusters the MNLP variance is huge, and hence the result is not reliable.

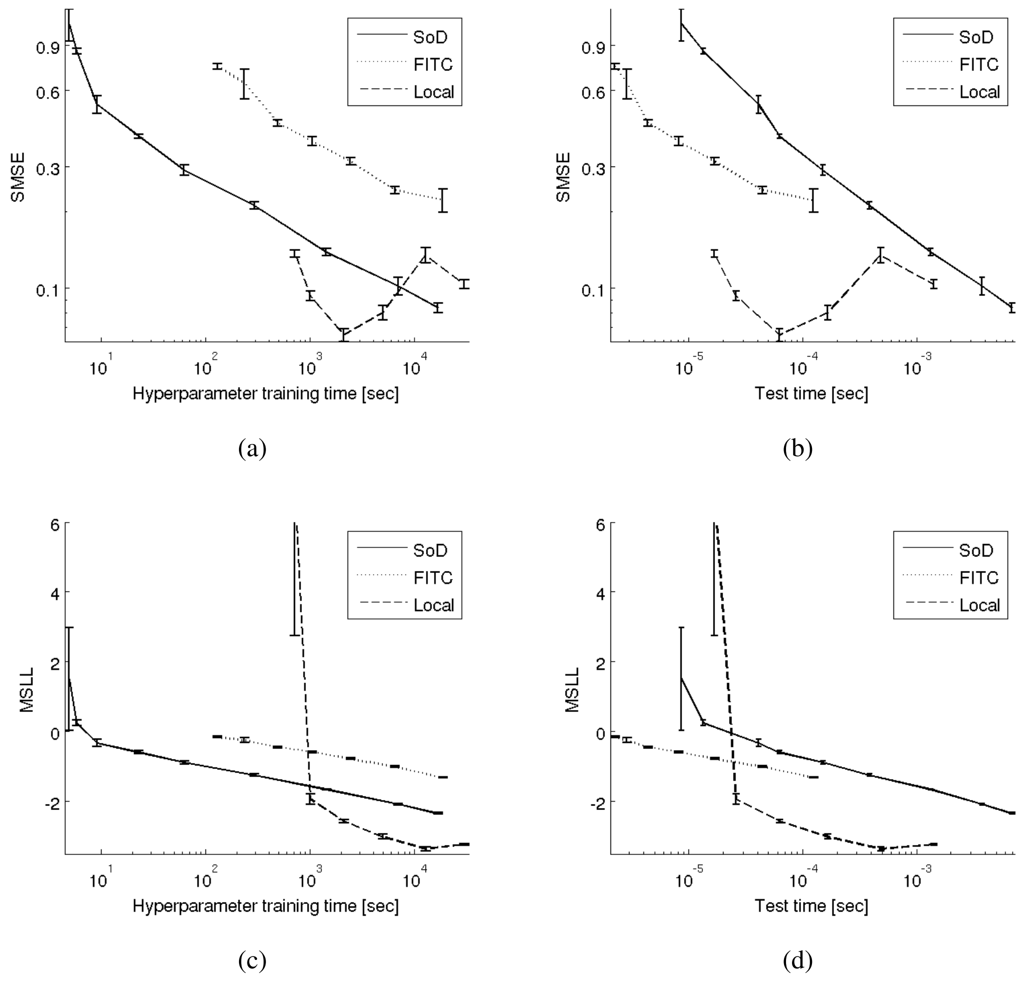
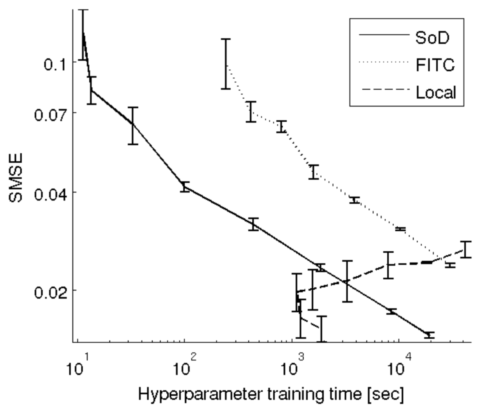


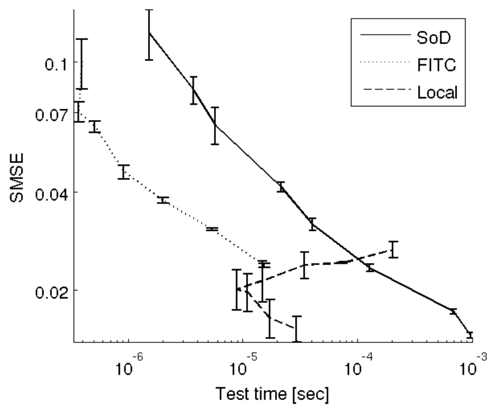
Figure 6.17: Time-performance plots for GP approximations running on CHEM data.

6.3.4 SARCOS

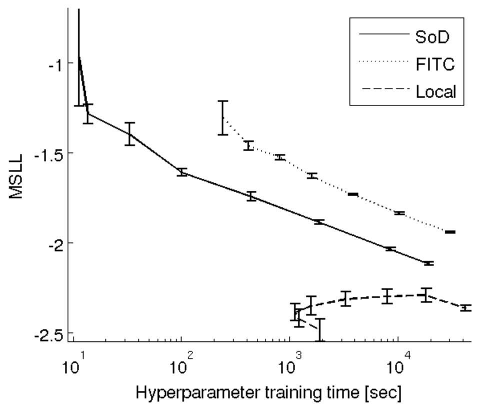
SARCOS was perhaps the most interesting dataset to look at. Figure 6.18 shows no surprises in that again SoD is good in hyperparameter training and FITC/Local do better in the test phase. The thing to note is that this time Local GP's performance is *better* with *smaller* cluster sizes (see Section 6.2 for more discussion on this). Unfortunately, as we already noted, smaller cluster sizes can require growing resources with Local GP, hence the odd-looking performance curves.



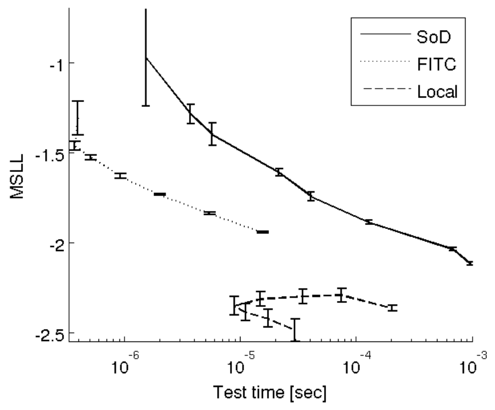
(a)



(b)



(c)



(d)

Figure 6.18: Time-performance plots for GP approximations running on SARCOS data.

6.4 Recommendations

The experimental results presented in Sections 6.1, 6.2 and 6.3 allow us to draw conclusions as to which method is best used, depending on the situation. The conclusions presented in this section sum up the whole of the results and analysis presented in this dissertation, and so it is difficult to provide pointers to specific sections under each claim. We hope that our recommendations follow clearly from the discussion in the three experimental results sections bove, which do provide references to specific evidence whenever appropriate. Table 6.3 attempts to sum up our results telegraphically.

6.4.1 Recommendations by computational budget constraint

If a particular computational resource is limited, it appears clearly that some methods are much better choices for GP regression than others. The list below sums up our findings in this context.

Test times If a limited test time budget is available but we can afford long and accurate hyperparameter training, then **FITC** or **Local GP** are likely to be the best choice. If the time budget is *very* limited, FITC allows for the fastest training with reasonable accuracy. However, at slightly larger test times Local outperforms FITC: it is slower but can be much more accurate.

Training times In some cases, only a limited amount of time might be available for hyperparameter training. All the approximations seem to allow for reasonable training times (the longest we had to wait was about 12 hours for training Local GP with clusters of size 4096 points). However, if care is not taken when choosing an approximation, there do exist potential pitfalls (consider IFGT's hyperparameter optimization times, Section 4.2.2). In sum, in great majority of cases in our experiments we see a similar behavior: **Local GP** gives best results if one can afford long hyperparameter training time. If only limited training time is available, it seems clear that **SoD** should be the approximation of choice.

Storage Space Limited storage space (mobile applications with limited available memory come to mind) can render potentially interesting approximations useless. While we don't provide explicit memory usage vs. performance plots, understanding the methods' space usage is much more straightforward than looking at their runtime. One might try to optimize for small constant factors, but FITC will always need to store an $m \times N$ covariance matrix in memory. Local GP is more flexible: first of all, if the test inputs are known on training, one only needs to keep one local $m \times m$ covariance matrix at a time. This suggests that if the test time is of no concern at all, one can simply wait with training the hyperparameters till the test points are known. If they arrive in an on-line fashion but at long intervals, one could re-compute the appropriate local covariance matrix each time a test point arrives and thus keep the $m \times m$ space complexity while retaining Local GP's performance - but this requires using the separate hyperparameter training approach, which yields worse performances with small cluster sizes. In the end, we can always simply use the Subset of Data method which always keeps only $O(m \times m)$ values in memory and, as we already mentioned, gives reasonable performances if training time is of concern anyway. In addition, in the Future Work Section 7.3 we mention a possible extension to FITC that can make it more useful in limited space regimes.

Above we suggest that SoD does better than FITC training-time-wise, but worse test-time-wise. Why is that? It turns out that the gap between the amount of computation FITC and SoD need to do once the model is trained is much smaller than the gap between the amount of computation FITC and SoD need to do on training. This is because computing the derivatives of log likelihood under FITC is particularly complicated (Snelson [2001] Appendix C derives the relevant derivatives), much more so than computing FITC predictive distribution once the hyperparameters are fixed. For both SoD and Local GP, hyperparameter training is not that much more complex than prediction.

6.4.2 Recommendations by dataset type

We might be less concerned with specific computational constraints, but simply require getting as good results as possible in reasonable time and using feasible space. Section 6.1 shows that again, IFGT is probably not a good choice in any nontrivial case. However, the other approximations can apply with varying efficiency to different problems. This section gathers some observations we made regarding the algorithms' performance on different data.

Input dimensionality The only method that seems to have significant problems with growing dimensionality is IFGT. However, SoD and FITC are not immune to the curse of dimensionality. This is because these methods base their prediction on information flow between a set of inducing points, which are a small subset of the full data. In high dimensions only the points at the center of the input space can communicate efficiently with most other data; the points closer to the edges will only be able to use exponentially less information with growing dimension. In case of Local GP the situation is less clear: on one hand, more local information is retained under this approximation– and only local information is actually important in very high dimensions. On the other hand, one would expect to experience even more of the "edge-effect", that distracts FITC and SoD from performing well, under this approximation, as Local GP creates clusters with "edges" all throughout the space, not only on the outskirts of the input samples. The effect of dimensionality on Local GP seems to be an interesting, but unanswered, question.

Input set structure As expected, if the input data are not structured (e.g. the SYNTH datasets), it is better to use random methods for inducing points choice or clustering. In particular, random methods tended to give better results on high dimensional datasets in our experiments. This makes sense as the lack of obvious structure in the data would require the inducing points or the clusters to ideally be placed in regular patterns in the input space, but in high dimensions this is usually not feasible. If, on the other hand, we see much structure in the data, using more sophisticated clustering/inducing point choice methods can point the approximations at the relevant regions in the input space, improving their performance. In addition, the Local approximation seems likely to use strong dataset structure to achieve good results with little computational effort.

Desideratum	**Best approximation**	**Notes**
Short test and training times, small memory usage	Local GP	Only if test data known in advance and Local implemented efficiently
Short test and training times	Local GP	Test data need not be known in advance
Short test time, small memory usage	Local GP	Only if test data known in advance; otherwise, SoD
Short test time	FITC	
Short training time, small memory usage	SoD	Consider Local GP if test data known in advance
Short training time	SoD	
Small memory usage	Local GP	Recompute the local covariance matrices on each new test point
Need best possible results in feasible time	Local GP	Possibly consider SoD/FITC, see Section 7.2

Table 6.3: Our choices of approximation flavors that give best performance in different situations. Local GP is particularly attractive when one can afford not storing all the local covariance matrices (for example, when the test data is available on training time, or when one can afford recomputing the local covariance matrix on each test point); it often yields best results for mid- and high- range of the timescale. FITC provides good results in short test times, but is slow to train and always requires more space than SoD. SoD is relatively easy to train well. See also Table 6.2 which looks at the methods' performance on different datasets.

Chapter 7

Conclusions and Future Work

We believe that our work shows that a practical comparison of algorithms is both useful and interesting. Presenting asymptotic complexities is not enough to measure an algorithm's usefulness. A theoretical analysis can help understand where an algorithm applies best, but looking at its performances in practice and in diverse contexts is important for a full appreciation of the algorithm's behavior. This chapter sums up our work in Section 7.1. Section 7.2 mentions several GP approximations which we did not consider in our experiments and discusses whether our results can shed any light on these methods' usefulness. Section 7.3 talks about possible future directions of research that would extend and clarify our conclusions.

7.1 Summary of completed work

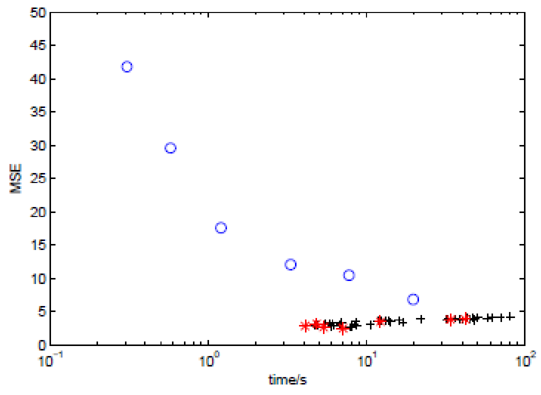
We investigated the practical usefulness of several approximations to the GP regression algorithm, the theory behind which was explained in Chapter 2. The full algorithm scales cubically in the number of training datapoints and requires storing a matrix of dimension equal to the number of datapoints in memory. Chapter 3 presented four proposed approximation algorithms: Subset of Data, Local GP, Fully Independent Conditional and GP using Conjugate Gradients with approximate Matrix-Vector Multiplication. These methods have different asymptotic runtimes and can behave very differently in different contexts (varying datasets or computational constraints); in particular, we showed that it is important to consider not only theoretical-asymptotic properties of the algorithms, but also their practical properties in Chapter 4. We have created a testing framework that visualizes the practical performance of the methods in a clear and easy to interpret way; this framework is explained and justified in Chapter 5. Finally, Chapter 6 presented the results of our tests. In the same chapter we considered a spectrum of possible situations in which one might need to resort to using a GP approximation algorithm and explain, based on our experimental findings, which algorithm should work best in each case.

7.2 Other methods

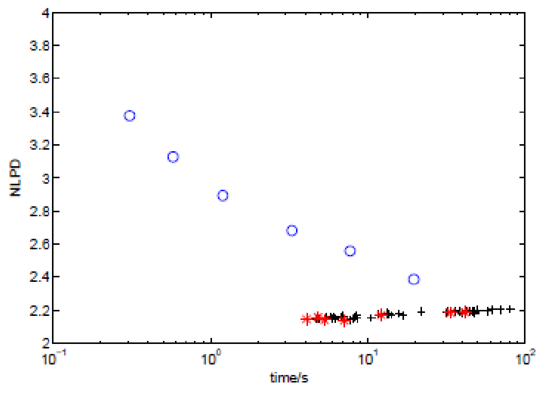
There are several approximation algorithms we know of which we do not consider in this dissertation. Two interesting algorithms that we feel are worth mentioning are PITC and SSGP. In addition we shortly discuss an extension to Gaussian Process algorithms called Warped GP.

7.2.1 Partially Independent Conditional

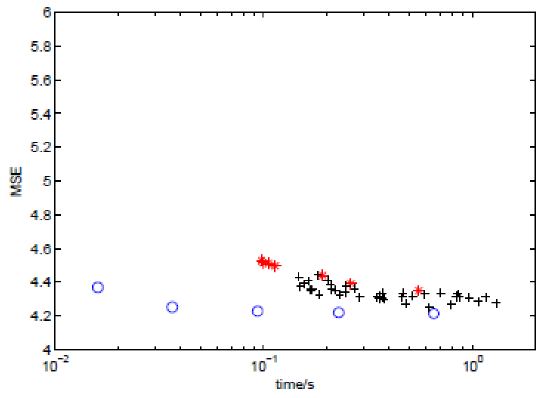
Introduced by Ed Snelson during his PhD research (Snelson [2001]), this approximation connects the ideas of FITC and Local GP. We saw that FITC approximates the full GP by using the *subset of regressors* covariance matrix instead of the full covariance, but with the full variances on the diagonal (see Chapter 3). Local GP, on the other hand, approximates the GP by using only block diagonal elements of the full covariance matrix and setting all the other elements to zero. PIC uses the subset of regressors matrix like FITC, but places the block diagonal with full covariances like Local GP. It turns out that asymptotically PIC training runtimes are as good as FITC, and we would expect to be able to gain accuracy by using this interesting covariance. However, Snelson [2001] performed experiments similar to ours on FITC, PIC and Local GP, but on a smaller range of interesting datasets and using pre-computed hyperparameters. These restricted experiments show that the performance of PIC is very similar to that of Local GP (Figure 7.1 reproduces the plots from Snelson [2001] that show this). In addition, PIC requires larger *test times* than FITC or Local GP.



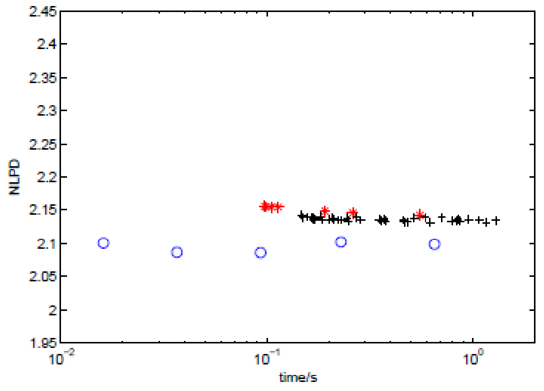
(a) SARCOS. MSE.



(b) SARCOS. NLPD.



(c) Abalone. MSE.



(d) Abalone. NLPD.

Figure 7.1: Comparison of FITC (blue circles), Local GP (red stars) and PIC (black crosses), copied from Snelson [2001]. MSE is the unnormalized Mean Squared Error and NLPD is the negative log predictive density, which is unnormalized version of our MSLL. In these experiments PIC does marginally better than Local GP on the `Abalone` dataset (see Snelson [2001] for more details on this data), and there is no significant difference on SARCOS. Note that this figure confirms our results from Chapter 6, where our experiments suggested that `SARCOS` is easier to learn with *smaller* clusters in Local GP.

7.2.2 Sparse Spectrum Gaussian Processes

This is a relatively new method, introduced in Lazaro-Gredilla et al. [2010], after we had been working on this project for a while. SSGP works with the *spectral representation* of the Gaussian Process it approximates, and thus appears to be interestingly different from the other algorithms we considered. Two characteristics of the method that we notice immediately are:

- Efficient hyperparameter training methods (maximizing the marginal likelihood) are developed in the introducing publications, and the algorithm can compute the full predictive distribution, including the predictive variances.
- The method can only approximate *stationary* Gaussian Processes.

While the second point means that we can expect SSGP to be less flexible than for example Local GP (which does surprisingly well on the difficult SARCOS data), the first point means that the method can still be potentially useful and interesting to look at. Lazaro-Gredilla et al. [2010] attempts to compare SSGP with FITC and another related algorithm. Unfortunately, the reported results appear of little practical significance, as the paper looks at the methods' error performance as a function of the "number of basis functions" - that is, it varies the complexity parameter of each method and reports the results on a joint plot. We reproduce the plot in Figure 7.2; note that even though plots appear to suggest that SSGP is better than the other methods, this conclusion does not actually follow from the figure directly. Unfortunately, we did not manage to include this method in our experiments on time.

7.2.3 Warped Gaussian Process

Warped GP, introduced in Snelson et al. [2004], is not an approximation method *per se*. Instead, it *warps* the dataset – that is, puts the output values through a nonlinear transformation – that makes it better modelled by a GP. This warping process could in principle be applied in the preprocessing stage before any approximation is used, and hence we do not include this algorithm in our comparison. Snelson et al. [2004] reports good results of applying the algorithm on several relatively small datasets. The warping step is automatized and so should not take much effort to use; it seems reasonable to consider using this algorithm in the preprocessing of the data to get best results with any GP algorithm.

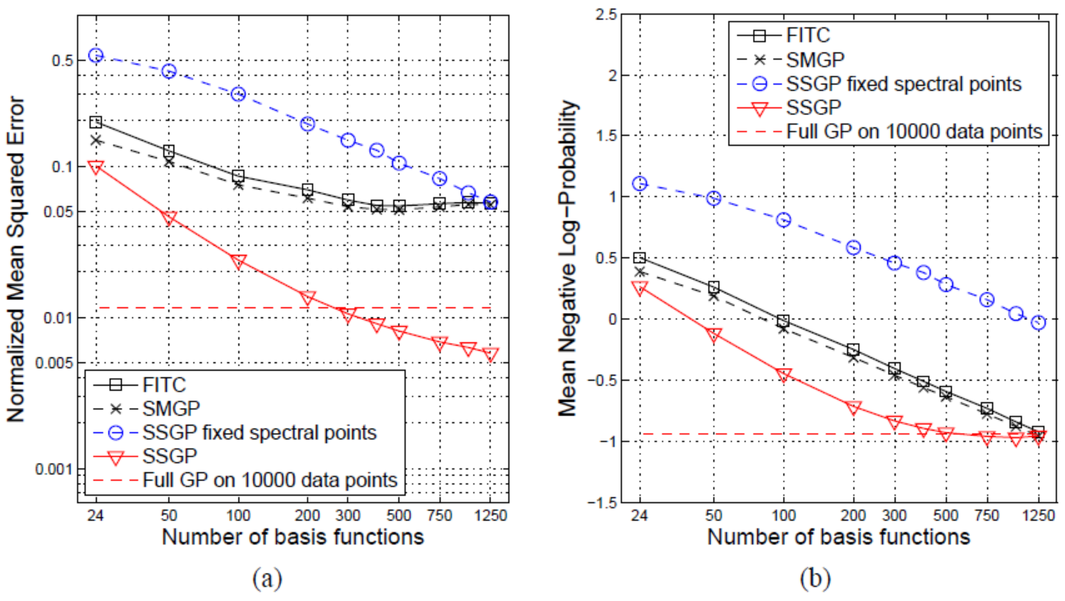


Figure 7.2: Comparison of FITC, two flavors of SSGP and another method which we do not describe in this thesis. The plots are copied from Lazaro-Gredilla et al. [2010] (the dataset used for testing is `Kin40k`, see Lazaro-Gredilla et al. [2010]). The plots shows that SSGP (with optimized spectral points) gives performances superior to FITC, but it is unclear how much more computational effort SSGP required to achieve these results. The figure uses error measures equivalent to the ones used in this thesis (Normalized Mean Squared Error is our SMSE, Mean Negative Log Probability is our MSLL).

7.3 Future Work

We came upon several interesting problems during our investigations. We partially investigated these problems as they appear very important for a full understanding of the discussed algorithms' behavior, but we could not afford putting enough effort into these side-projects to reach definite conclusions.

7.3.1 FITC on a subset of data

It is possible, and indeed can be reasonable, to connect the FITC and SoD methods. One could first choose a subset of data of size $n < N$ (where N is the number of training points available), then run the FITC algorithm on this dataset only, using $m < N$ inducing points. The asymptotic time complexity of this method would thus be $O(nm^2)$ and required storage space $O(nm)$. It seems to us that especially the required space reduction could be beneficial, as in our tests we were unable to run $FITC$ with more than about 1000 inducing points due to space restrictions; however, if the subset of data was chosen in a way that does not delete much useful information, then we could expect significant gains from being able to use more inducing points on an almost equally informative dataset.

The important question that we did not attempt to answer in our work is *how to choose the $\frac{n}{m}$ ratio*? The interplay of n and m does not seem obvious. We might expect choosing $n < m$ (so at least some inducing points are outside of the training subset) to be unreasonable, as the sole function of the m inducing points is to enable approximate communication between the n datapoints, and simply running the SoD GP on the n points should give better results. However, even this is not entirely clear, as again we want to consider the practical performance of the methods, and FITC with m inducing points on a subset of size n would take $O(nm^2)$ time which is less than $O(m^3)$ if $n < m$!

In sum, knowing the optimal n/m ratio could thus possibly make FITC a much more useful method and we see it as an important open question. (Thanks to Iain Murray for drawing our attention to this problem!)

7.3.2 Local GP efficiency

Section 4.3.2 shows that the Local GP algorithm's behavior can be counterintuitive: for very small cluster sizes m the algorithm can actually run *slower* than for mid-range cluster sizes (it slows down again with m growing more, as the asymptotic analysis of the algorithm requires). This behavior is very unfortunate, as we showed in Chapter 6 that Local GP can return accurate predictive distributions, often better than those computed by other approximations.

We suggested that the reason for this behavior is that doing computations with small clusters requires much more bookkeeping. For example, using 1000 clusters of size 32 requires shuffling many small arrays in- and out- of memory and calling many short-lived functions numerous times, which can also create an overhead. It is possible that Matlab, as an interpreted language whose focus is on large matrix algebra, is particularly bad at running such segmented algorithms. It would be very interesting to see if Local GP can be implemented efficiently enough to be even more competitive.

7.3.3 Mixing different methods for training and testing

It is possible to use one algorithm to train the hyperparameters, and another one to do the test predictions. We have seen that SoD appears to be a good hyperparameter training method, but other algorithms do better on test time. Can we use the hyperparameters estimated by SoD with Local GP or FITC? Looking at the hyperparameter tables in Appendix A, we spot some regularities - as we already mentioned in Section 6.2.3, FITC appears to choose the hyperparameters in an intrinsically different manner than SoD (and thus full GP) does. SoD and Local GP with joint hyperparameters also don't seem to converge to the same hyperparameter values. A more detailed research in this direction would be interesting, and it seems to us that most publications introducing new GP approximations omit the topic of how the approximations use the hyperparameters in comparison to the full GP they are meant to approximate.

7.4 Conclusion

There exist many interesting methods for approximate inference with Gaussian Processes, but very often their practical usefulness is clear. We believe that focusing only on the theoretical properties of these algorithms is not reasonable; after all, the goal of approximation algorithms is to make other methods *practically useful*. We hope to have shown that performing an unbiased empirical comparison of approximation algorithms is possible and can bring about both useful conclusions as well as new interesting questions that can deepen our understanding of the considered methods.

Appendix A

Appendix: Tables

Following are hyperparameter tables as optimized by the different methods (excluding IFGT). For SYNTH2 and SYNTH8 the "real" lengthscales $d1, d2$ of the data, as well as the signal amplitude $amp = |\sigma_f|$, are all 1. For SYNTH2, The noise level of the training data is about 10^{-6} so $\log \sigma_n^2 = -13.47$ after normalization. This is only approximate as we only know the exact noise level for the whole dataset, while in our experiments we use a randomly chosen half of it for training only. For SYNTH8, the log noise variance is -6.94.

In any table, entry showing *inf* means that the particular value exceeded 9999. We don't report such large values because the tables would take up too much space. What's more, given the dataset variances are all normalized to unity, we doubt there is any practical distinction between lengthscales of such large values.

The "errorbars" ($\pm$ values) shown in Local GP separate hyperparameter training are variances over the hyperparameters in different clusters. Local GP with joint training uses the same set of hypers for each cluster, so no variance is present.

The errorbars for SoD and FITC are over 5 repeated runs of the experiment for each different inducing point choice or clustering method.

	64pts	**128pts**	**256pts**	**512pts**	**1024pts**	**2048pts**	**4096pts**	**6000pts**	**8000pts**
d1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
amp	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.0
$\log \sigma_{\mathbf{n}}^{2}$	-10.3 ± 0.8	-13.6 ± 1.2	-13.2 ± 0.2	-13.2 ± 0.1	-13.2 ± 0.0	-13.2 ± 0.0	-13.2 ± 0.0	-13.2 ± 0.0	-13.1 ± 0.0

Table A.1: SoD, SYNTH2, FPC

	64pts	**128pts**	**256pts**	**512pts**	**1024pts**	**2048pts**	**4096pts**	**6000pts**	**8000pts**
d1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
amp	1.7 ± 0.1	1.6 ± 0.1	1.5 ± 0.1	1.5 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.5 ± 0.0	1.5 ± 0.0
$\log \sigma_{\mathbf{n}}^{2}$	-12.9 ± 0.3	-13.3 ± 0.2	-13.2 ± 0.0	-13.1 ± 0.1	-13.2 ± 0.1	-13.1 ± 0.0	-13.1 ± 0.0	-13.1 ± 0.0	-13.2 ± 0.0

Table A.2: SoD, SYNTH2, Random

	64pts	**128pts**	**256pts**	**512pts**	**1024pts**	**2048pts**	**4096pts**	**6000pts**	**8000pts**
d1	8.1 ± 8.3	1.3 ± 1.0	1.0 ± 0.2	1.0 ± 0.2	1.0 ± 0.1	1.1 ± 0.0	1.1 ± 0.0	1.1 ± 0.0	1.1 ± 0.0
d2	1.5 ± 0.7	4.0 ± 5.6	1.6 ± 1.2	1.0 ± 0.2	1.2 ± 0.1	1.1 ± 0.1	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d3	1.0 ± 0.4	1.5 ± 0.8	1.9 ± 0.6	1.4 ± 0.3	1.1 ± 0.3	1.1 ± 0.1	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d4	12.0 ± 9.7	4.4 ± 6.5	1.9 ± 0.6	1.5 ± 0.2	1.3 ± 0.1	1.3 ± 0.2	1.2 ± 0.0	1.1 ± 0.1	1.1 ± 0.0
d5	6.2 ± 6.7	5.3 ± 6.8	1.8 ± 1.0	1.6 ± 1.0	1.2 ± 0.1	1.1 ± 0.1	1.1 ± 0.0	1.1 ± 0.0	1.0 ± 0.0
d6	10.6 ± 11.6	1.1 ± 0.3	3.2 ± 4.0	1.2 ± 0.1	1.0 ± 0.1	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d7	4.1 ± 6.4	1.0 ± 0.7	1.0 ± 0.2	1.1 ± 0.2	1.3 ± 0.1	1.1 ± 0.1	1.1 ± 0.0	1.1 ± 0.0	1.0 ± 0.0
d8	9.2 ± 16.5	1.3 ± 0.8	0.8 ± 0.4	1.2 ± 0.3	1.1 ± 0.1	1.2 ± 0.1	1.1 ± 0.0	1.1 ± 0.0	1.1 ± 0.0
amp	1.0 ± 0.1	1.0 ± 0.0	0.8 ± 0.1	0.9 ± 0.1	1.0 ± 0.1	0.9 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
$\log \sigma_{\mathbf{n}}^{2}$	-4.6 ± 1.7	-5.5 ± 1.7	-2.0 ± 1.6	-2.2 ± 1.2	-4.2 ± 2.9	-1.6 ± 0.3	-2.4 ± 0.2	-4.0 ± 1.6	-3.7 ± 0.3

Table A.3: SoD, SYNTH8, FPC

	64pts	**128pts**	**256pts**	**512pts**	**1024pts**	**2048pts**	**4096pts**	**6000pts**	**8000pts**
d1	10.8 ± 20.0	4.4 ± 7.2	1.0 ± 0.2	1.1 ± 0.1	1.0 ± 0.0	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d2	3.7 ± 4.6	18.8 ± 34.8	1.1 ± 0.2	1.0 ± 0.1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d3	21.2 ± 38.6	11.7 ± 12.7	1.2 ± 0.2	1.2 ± 0.2	1.2 ± 0.1	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d4	6.2 ± 7.8	1.1 ± 0.4	1.6 ± 0.8	1.1 ± 0.2	1.1 ± 0.1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d5	45.9 ± 52.4	10.3 ± 16.3	1.2 ± 0.2	1.1 ± 0.1	1.1 ± 0.1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d6	1.0 ± 0.2	2.2 ± 2.7	1.3 ± 0.3	1.2 ± 0.1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d7	18.4 ± 22.9	9.8 ± 18.2	1.1 ± 0.4	1.1 ± 0.1	1.0 ± 0.1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d8	6.4 ± 10.8	3.9 ± 5.8	1.0 ± 0.1	1.0 ± 0.1	1.0 ± 0.0	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
amp	1.0 ± 0.1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
$\log \sigma_{\mathbf{n}}^{\mathbf{2}}$	-6.9 ± 1.4	-5.1 ± 1.8	-6.8 ± 0.8	-5.5 ± 2.9	-4.4 ± 0.7	-4.8 ± 0.6	-5.7 ± 0.7	-6.2 ± 0.4	-6.7 ± 0.0

Table A.4: SoD, SYNTH8, Random

	64pts	**128pts**	**256pts**	**512pts**	**1024pts**	**2048pts**	**4096pts**	**6000pts**
d1	55.9 ± 68.3	3.6 ± 0.4	32.1 ± 57.4	2.9 ± 0.3	3.2 ± 0.3	3.6 ± 0.2	4.4 ± 0.1	4.7 ± 0.1
d2	5.8 ± 8.6	10.9 ± 5.7	21.3 ± 14.1	30.8 ± 4.4	28.6 ± 15.0	7.0 ± 4.1	4.5 ± 0.2	4.6 ± 0.2
d3	21.2 ± 28.5	3.6 ± 1.1	3.6 ± 1.2	2.3 ± 0.2	2.1 ± 0.2	2.1 ± 0.1	1.9 ± 0.0	1.8 ± 0.0
d4	10.6 ± 13.4	1.9 ± 0.5	3.4 ± 1.9	1.5 ± 0.2	1.3 ± 0.2	1.4 ± 0.1	1.4 ± 0.0	1.4 ± 0.0
d5	62.3 ± 52.7	25.7 ± 17.9	20.0 ± 17.3	41.0 ± 9.8	10.9 ± 4.0	7.1 ± 1.2	7.4 ± 0.6	7.9 ± 0.3
d6	20.5 ± 9.7	1.1 ± 0.5	7.5 ± 6.4	3.4 ± 0.9	3.5 ± 0.4	3.0 ± 0.2	1.3 ± 0.1	1.0 ± 0.0
d7	27.3 ± 33.1	36.7 ± 15.8	15.8 ± 17.3	2.1 ± 0.3	2.1 ± 0.0	2.5 ± 0.2	3.2 ± 0.1	3.3 ± 0.1
d8	21.5 ± 22.9	26.6 ± 13.0	15.6 ± 24.9	2.6 ± 0.5	2.5 ± 0.2	3.1 ± 0.1	4.1 ± 0.1	4.5 ± 0.0
d9	6.2 ± 7.0	9.7 ± 10.0	1.9 ± 0.3	3.1 ± 0.5	2.9 ± 0.1	2.8 ± 0.1	2.5 ± 0.2	2.3 ± 0.1
d10	246.3 ± 458.9	37.9 ± 20.3	36.0 ± 5.4	46.4 ± 19.2	23.5 ± 11.7	7.6 ± 2.6	4.1 ± 0.3	3.0 ± 0.1
d11	39.7 ± 21.6	19.0 ± 25.4	20.2 ± 15.6	18.3 ± 5.4	17.3 ± 10.4	13.0 ± 3.9	4.7 ± 0.2	3.8 ± 0.1
d12	26.1 ± 45.2	31.6 ± 24.0	2.2 ± 0.9	1.8 ± 0.4	1.5 ± 0.1	1.6 ± 0.1	1.9 ± 0.0	2.1 ± 0.0
d13	33.3 ± 18.5	35.0 ± 26.7	32.8 ± 13.6	2.3 ± 0.2	2.7 ± 0.3	3.0 ± 0.0	3.8 ± 0.1	3.8 ± 0.0
d14	38.4 ± 44.4	2.6 ± 1.3	3.2 ± 0.9	5.2 ± 0.4	6.6 ± 1.2	5.8 ± 0.3	4.5 ± 0.2	4.4 ± 0.1
d15	49.7 ± 52.2	31.9 ± 19.1	6.0 ± 2.6	5.4 ± 0.5	6.5 ± 0.7	6.0 ± 0.3	4.6 ± 0.2	3.9 ± 0.1
amp	1.5 ± 0.1	1.4 ± 0.1	1.4 ± 0.1	1.6 ± 0.0	1.8 ± 0.1	2.2 ± 0.0	3.1 ± 0.1	3.7 ± 0.0
$\log\sigma_{\mathbf{n}}^{\mathbf{2}}$	-6.3 ± 2.1	-3.1 ± 2.3	-1.6 ± 0.4	-2.3 ± 0.4	-2.2 ± 0.1	-2.5 ± 0.2	-3.2 ± 0.1	-3.9 ± 0.0

Table A.5: SoD, CHEM, FPC

	64pts	**128pts**	**256pts**	**512pts**	**1024pts**	**2048pts**	**4096pts**	**6000pts**
d1	7.8 ± 6.5	2.7 ± 0.5	2.8 ± 0.3	3.1 ± 0.2	3.6 ± 0.3	4.1 ± 0.2	4.6 ± 0.1	4.8 ± 0.1
d2	4.5 ± 4.9	13.9 ± 10.8	6.1 ± 6.9	2.1 ± 0.4	2.3 ± 0.2	3.0 ± 0.1	3.4 ± 0.3	3.4 ± 0.2
d3	3.0 ± 3.5	1.0 ± 0.1	1.1 ± 0.1	1.1 ± 0.1	1.2 ± 0.0	1.4 ± 0.1	1.5 ± 0.1	1.6 ± 0.1
d4	1.5 ± 1.9	0.7 ± 0.1	0.8 ± 0.1	0.8 ± 0.1	0.9 ± 0.1	1.1 ± 0.0	1.2 ± 0.0	1.2 ± 0.0
d5	11.1 ± 7.9	20.6 ± 6.1	5.4 ± 2.8	4.2 ± 1.1	5.0 ± 0.3	5.8 ± 0.8	5.9 ± 0.7	5.8 ± 0.2
d6	0.3 ± 0.1	0.5 ± 0.1	0.5 ± 0.1	0.5 ± 0.1	0.5 ± 0.0	0.6 ± 0.0	0.7 ± 0.0	0.7 ± 0.0
d7	11.7 ± 9.6	13.9 ± 18.2	4.0 ± 1.1	4.4 ± 1.1	4.2 ± 0.4	4.5 ± 0.3	4.8 ± 0.4	4.3 ± 0.5
d8	8.6 ± 9.4	13.3 ± 10.0	9.1 ± 9.9	3.5 ± 0.8	3.6 ± 0.1	4.6 ± 0.2	5.0 ± 0.3	5.1 ± 0.3
d9	3.8 ± 3.9	1.4 ± 0.3	1.6 ± 0.3	1.4 ± 0.1	1.6 ± 0.1	1.8 ± 0.0	1.9 ± 0.1	1.9 ± 0.1
d10	10.1 ± 4.9	21.8 ± 15.6	8.3 ± 7.9	4.5 ± 0.7	3.1 ± 0.8	2.4 ± 0.3	2.4 ± 0.1	2.3 ± 0.1
d11	7.7 ± 5.4	2.3 ± 1.4	1.4 ± 0.3	1.8 ± 0.2	2.3 ± 0.6	2.5 ± 0.3	2.5 ± 0.2	2.3 ± 0.2
d12	17.2 ± 12.9	1.7 ± 0.3	2.1 ± 0.2	2.2 ± 0.2	2.6 ± 0.2	2.9 ± 0.2	2.9 ± 0.2	2.8 ± 0.1
d13	2.6 ± 3.5	1.8 ± 0.5	2.0 ± 0.2	2.3 ± 0.3	2.7 ± 0.2	3.3 ± 0.3	3.4 ± 0.2	3.1 ± 0.1
d14	9.6 ± 10.4	2.1 ± 0.4	2.7 ± 1.1	2.5 ± 0.5	2.3 ± 0.2	2.9 ± 0.3	3.1 ± 0.2	2.7 ± 0.2
d15	15.2 ± 11.5	15.2 ± 8.6	13.0 ± 15.4	5.9 ± 1.0	4.5 ± 0.5	3.7 ± 0.2	3.4 ± 0.1	3.3 ± 0.2
amp	0.9 ± 0.1	1.3 ± 0.1	1.3 ± 0.1	1.5 ± 0.1	2.1 ± 0.2	3.1 ± 0.2	4.4 ± 0.3	4.8 ± 0.4
$\log \sigma_n^2$	-6.3 ± 1.5	-5.1 ± 2.0	-3.9 ± 1.6	-3.9 ± 0.7	-4.9 ± 0.2	-6.0 ± 0.1	-7.0 ± 0.2	-7.8 ± 0.1

Table A.6: SoD, CHEM, Random

	64pts	**128pts**	**256pts**	**512pts**	**1024pts**	**2048pts**	**4096pts**	**6000pts**
d1	9.4 ± 4.6	6.7 ± 2.6	6.8 ± 0.7	7.1 ± 1.4	8.2 ± 1.5	6.2 ± 0.3	5.2 ± 0.7	4.1 ± 0.2
d2	661.8 ± 305.1	4557.8 ± 7996.5	122.7 ± 203.7	7.7 ± 1.1	7.2 ± 0.8	7.9 ± 1.0	5.8 ± 0.2	6.1 ± 0.3
d3	4277.6 ± 7887.3	1222.7 ± 1462.2	3578.2 ± 7045.3	56.8 ± 63.9	23.0 ± 7.7	27.6 ± 5.9	13.1 ± 3.0	9.5 ± 0.6
d4	164.2 ± 171.2	27.2 ± 30.4	82.1 ± 87.0	10.0 ± 5.0	9.8 ± 1.9	8.0 ± 0.7	5.5 ± 0.4	4.6 ± 0.4
d5	3122.9 ± 5898.4	254.3 ± 494.5	39.9 ± 44.5	10.4 ± 11.8	3.4 ± 0.2	2.5 ± 0.2	2.0 ± 0.2	2.0 ± 0.1
d6	372.0 ± 408.2	429.1 ± 658.2	243.6 ± 165.3	25.7 ± 17.5	15.4 ± 8.4	34.9 ± 13.5	16.1 ± 3.2	13.8 ± 2.1
d7	2552.0 ± 4889.6	262.2 ± 258.3	125.2 ± 62.6	41.7 ± 28.8	19.6 ± 12.2	5.6 ± 0.8	4.1 ± 0.6	3.7 ± 0.2
d8	872.7 ± 783.3	233.9 ± 146.1	133.4 ± 146.6	5.1 ± 1.3	3.7 ± 1.0	2.5 ± 0.5	1.6 ± 0.1	1.4 ± 0.1
d9	1506.7 ± 1884.2	575.9 ± 183.1	681.6 ± 570.2	42.4 ± 32.9	15.6 ± 2.9	18.3 ± 2.1	18.7 ± 2.4	17.1 ± 0.8
d10	1060.4 ± 824.4	880.6 ± 402.0	654.6 ± 870.5	161.9 ± 39.9	29.6 ± 8.3	15.4 ± 3.0	13.2 ± 3.4	10.1 ± 1.3
d11	947.6 ± 843.1	206.0 ± 264.5	525.6 ± 649.8	73.4 ± 71.3	17.7 ± 23.4	3.1 ± 0.2	2.7 ± 0.1	2.6 ± 0.1
d12	1976.0 ± 1722.1	963.6 ± 1221.4	741.4 ± 542.7	120.4 ± 56.1	71.2 ± 29.0	21.3 ± 8.1	9.5 ± 1.8	9.3 ± 1.1
d13	$Inf \pm 0.0$	6951.5 ± 12436.4	1281.4 ± 1734.0	179.0 ± 119.1	24.1 ± 20.0	20.9 ± 3.6	13.9 ± 0.5	13.6 ± 1.1
d14	2772.3 ± 2641.1	502.8 ± 441.1	280.1 ± 349.9	6.4 ± 1.4	5.6 ± 1.0	2.9 ± 0.3	3.0 ± 0.3	2.7 ± 0.2
d15	8.9 ± 5.6	4.7 ± 2.0	3.7 ± 0.8	3.4 ± 0.4	3.3 ± 0.3	3.5 ± 0.3	3.3 ± 0.2	3.1 ± 0.1
d16	266.2 ± 366.0	179.2 ± 242.6	19.8 ± 6.9	50.3 ± 33.0	22.7 ± 4.9	15.4 ± 0.4	14.6 ± 1.3	12.5 ± 0.7
d17	40011.8 ± 79693.0	$Inf \pm 0.0$	9054.6 ± 17845.0	18.9 ± 6.6	18.1 ± 3.2	15.4 ± 2.3	11.5 ± 1.5	10.2 ± 0.7
d18	14.2 ± 11.3	6.9 ± 1.6	4.8 ± 1.2	5.2 ± 0.7	4.4 ± 0.5	3.3 ± 0.3	3.3 ± 0.2	3.1 ± 0.2
d19	1276.5 ± 1527.3	216.6 ± 139.5	48.4 ± 74.6	16.5 ± 5.1	23.4 ± 14.7	13.4 ± 3.7	12.3 ± 1.8	9.6 ± 1.0
d20	316.6 ± 432.6	224.6 ± 298.8	268.8 ± 327.1	134.3 ± 104.2	16.5 ± 2.9	15.9 ± 1.8	13.6 ± 1.0	13.2 ± 0.7
d21	94.2 ± 50.1	188.3 ± 130.7	112.9 ± 98.1	14.3 ± 12.4	6.1 ± 1.2	3.1 ± 0.3	2.9 ± 0.2	2.8 ± 0.1
amp	5.8 ± 3.0	3.6 ± 0.9	3.2 ± 0.7	2.3 ± 0.3	2.0 ± 0.1	1.8 ± 0.1	1.6 ± 0.0	1.5 ± 0.0
$\log \sigma_{\mathbf{n}}^{\mathbf{2}}$	-3.5 ± 0.7	-3.3 ± 0.2	-3.3 ± 0.4	-3.6 ± 0.2	-3.7 ± 0.1	-4.0 ± 0.1	-4.3 ± 0.1	-4.4 ± 0.0

	64pts	128pts	256pts	512pts	1024pts	2048pts	4096pts	6000pts
d1	5.2 ± 3.2	7.0 ± 2.5	8.7 ± 2.3	6.2 ± 1.9	4.6 ± 1.8	3.8 ± 0.6	3.2 ± 0.4	3.1 ± 0.2
d2	149.0 ± 173.6	247.9 ± 194.0	50.8 ± 65.9	15.3 ± 6.0	9.1 ± 1.9	7.7 ± 1.4	5.3 ± 0.6	5.4 ± 0.6
d3	3503.1 ± 3656.5	311.4 ± 201.0	298.7 ± 255.5	96.9 ± 53.6	62.6 ± 76.2	28.9 ± 10.9	12.0 ± 3.5	10.5 ± 2.4
d4	90.2 ± 114.4	77.1 ± 115.4	74.2 ± 71.1	8.3 ± 3.3	5.9 ± 2.6	6.8 ± 2.4	4.2 ± 0.9	4.6 ± 0.4
d5	73.7 ± 56.4	135.7 ± 158.3	159.2 ± 143.7	5.4 ± 2.1	3.4 ± 0.7	2.2 ± 0.3	1.9 ± 0.2	1.7 ± 0.1
d6	235.8 ± 275.5	198.8 ± 178.2	165.7 ± 174.7	63.9 ± 56.3	59.0 ± 59.4	24.9 ± 5.0	23.2 ± 7.0	13.0 ± 3.1
d7	242.4 ± 208.0	17.6 ± 16.1	530.3 ± 1008.8	54.5 ± 46.5	19.9 ± 25.2	5.2 ± 1.4	4.6 ± 1.5	3.2 ± 0.4
d8	106.1 ± 142.5	122.1 ± 128.4	22.0 ± 15.7	13.0 ± 9.8	6.0 ± 1.3	5.0 ± 0.9	2.3 ± 0.7	1.5 ± 0.4
d9	182.4 ± 118.8	154.7 ± 112.8	76.4 ± 108.6	73.0 ± 123.4	10.2 ± 2.3	16.3 ± 2.0	16.7 ± 3.2	17.4 ± 2.7
d10	125.9 ± 119.8	155.0 ± 111.3	343.0 ± 294.7	101.8 ± 124.8	34.9 ± 25.1	16.1 ± 3.1	16.4 ± 5.5	14.5 ± 1.5
d11	86.7 ± 143.6	34.5 ± 26.5	90.9 ± 76.7	17.4 ± 13.4	12.1 ± 6.8	4.3 ± 0.8	2.9 ± 0.4	3.0 ± 0.3
d12	145.1 ± 86.1	156.4 ± 169.5	152.7 ± 114.6	67.6 ± 51.2	34.8 ± 25.1	16.6 ± 6.5	10.2 ± 1.3	12.0 ± 4.6
d13	85.6 ± 86.1	173.3 ± 151.4	1305.4 ± 2531.7	152.9 ± 167.6	29.7 ± 26.7	31.3 ± 13.7	16.9 ± 2.2	17.8 ± 2.6
d14	101.8 ± 131.0	151.2 ± 101.4	197.6 ± 176.6	62.2 ± 86.0	6.9 ± 2.2	3.1 ± 0.3	2.8 ± 0.5	2.3 ± 0.2
d15	2.5 ± 1.5	4.2 ± 2.2	5.3 ± 2.7	3.2 ± 0.7	2.7 ± 0.5	2.5 ± 0.3	2.3 ± 0.1	2.3 ± 0.1
d16	189.8 ± 219.0	148.3 ± 106.2	164.4 ± 50.4	106.8 ± 66.4	29.0 ± 16.8	14.9 ± 6.0	10.6 ± 2.8	10.2 ± 1.1
d17	39.6 ± 29.3	161.8 ± 146.7	32.7 ± 33.5	25.4 ± 21.6	11.1 ± 4.2	8.1 ± 1.1	7.8 ± 1.0	7.8 ± 1.1
d18	11.9 ± 12.7	5.7 ± 1.5	8.1 ± 4.1	5.8 ± 2.3	3.1 ± 0.5	2.4 ± 0.3	2.4 ± 0.2	2.6 ± 0.2
d19	63.2 ± 34.7	87.4 ± 99.0	96.2 ± 78.9	11.8 ± 4.8	12.7 ± 6.4	6.7 ± 1.8	6.0 ± 1.7	7.9 ± 0.4
d20	256.0 ± 268.4	162.9 ± 136.1	114.4 ± 92.2	60.9 ± 46.8	14.5 ± 2.6	13.5 ± 2.4	12.5 ± 1.8	12.8 ± 1.2
d21	105.9 ± 170.3	142.1 ± 122.7	193.6 ± 216.1	20.9 ± 29.1	6.6 ± 4.7	2.7 ± 0.4	3.0 ± 0.3	2.8 ± 0.4
amp	2.3 ± 1.2	2.7 ± 0.8	3.7 ± 1.7	2.1 ± 0.3	1.7 ± 0.1	1.5 ± 0.1	1.3 ± 0.0	1.3 ± 0.0
$\log\sigma_n^2$	-9.6 ± 6.3	-5.3 ± 3.2	-3.4 ± 0.2	-3.8 ± 0.1	-4.0 ± 0.1	-4.2 ± 0.1	-4.3 ± 0.0	-4.5 ± 0.0

	8pts	**16pts**	**32pts**	**64pts**	**128pts**	**256pts**	**512pts**
d1	2.9 ± 3.6	1.4 ± 0.2	1.1 ± 0.1	1.1 ± 0.1	1.1 ± 0.1	0.6 ± 0.1	0.5 ± 0.1
d2	1.6 ± 0.7	1.4 ± 0.3	1.3 ± 0.1	1.2 ± 0.1	1.1 ± 0.1	1.0 ± 0.1	1.0 ± 0.1
amp	7.4 ± 13.2	1.8 ± 0.5	1.5 ± 1.2	1.6 ± 1.2	1.2 ± 0.0	1.4 ± 0.0	1.5 ± 0.0
$\log \sigma_{\mathbf{n}}^{2}$	-2.0 ± 0.8	-3.6 ± 0.6	-5.5 ± 0.7	-10.8 ± 0.3	-12.8 ± 0.3	-12.3 ± 0.5	-11.2 ± 0.1

Table A.9: FITC, SYNTH2, FPC

	8pts	**16pts**	**32pts**	**64pts**	**128pts**	**256pts**	**512pts**
d1	1.0 ± 0.2	0.8 ± 0.1	1.0 ± 0.1	0.6 ± 0.0	0.5 ± 0.2	0.3 ± 0.1	0.4 ± 0.1
d2	1.0 ± 0.2	1.3 ± 0.2	1.1 ± 0.1	1.0 ± 0.1	0.9 ± 0.0	0.8 ± 0.1	0.9 ± 0.2
amp	1.3 ± 0.4	1.3 ± 0.5	0.9 ± 0.1	1.1 ± 0.0	1.3 ± 0.0	1.4 ± 0.1	1.4 ± 0.1
$\log \sigma_{\mathbf{n}}^{2}$	-6.4 ± 0.9	-9.8 ± 0.9	-11.9 ± 0.7	-10.7 ± 0.4	-11.1 ± 0.6	-11.7 ± 0.5	-10.9 ± 0.2

Table A.10: FITC, SYNTH2, Random

	8pts	**16pts**	**32pts**	**64pts**	**128pts**	**256pts**	**512pts**
d1	1.0 ± 0.0	1.1 ± 0.2	1.3 ± 0.3	1.3 ± 0.1	1.2 ± 0.1	1.3 ± 0.0	1.3 ± 0.1
d2	0.8 ± 0.1	1.0 ± 0.2	1.2 ± 0.2	1.4 ± 0.2	1.5 ± 0.1	1.3 ± 0.0	1.2 ± 0.0
d3	1.0 ± 0.1	1.5 ± 0.5	3.4 ± 4.0	2.0 ± 0.4	1.8 ± 0.1	1.6 ± 0.2	1.3 ± 0.1
d4	1.6 ± 0.3	5.6 ± 7.7	7.5 ± 6.1	1.9 ± 0.2	1.9 ± 0.4	1.5 ± 0.1	1.4 ± 0.2
d5	1.0 ± 0.1	1.8 ± 0.5	1.8 ± 0.4	1.5 ± 0.2	1.6 ± 0.3	1.4 ± 0.1	1.3 ± 0.1
d6	1.2 ± 0.2	1.5 ± 0.2	2.4 ± 1.6	1.7 ± 0.3	1.3 ± 0.1	1.3 ± 0.0	1.2 ± 0.1
d7	1.0 ± 0.1	1.2 ± 0.2	1.3 ± 0.3	1.4 ± 0.1	1.4 ± 0.1	1.3 ± 0.1	1.3 ± 0.1
d8	0.7 ± 0.0	1.0 ± 0.3	1.3 ± 0.3	1.3 ± 0.1	1.3 ± 0.1	1.3 ± 0.0	1.3 ± 0.0
amp	0.9 ± 0.0	0.9 ± 0.1	0.8 ± 0.1	0.9 ± 0.0	1.0 ± 0.0	1.1 ± 0.0	1.1 ± 0.0
$\log \sigma_{\mathbf{n}}^{2}$	-3.1 ± 0.2	-2.0 ± 1.1	-1.3 ± 1.3	-1.1 ± 0.1	-1.4 ± 0.1	-2.7 ± 0.6	-3.9 ± 0.5

Table A.11: FITC, SYNTH8, FPC

	8pts	**16pts**	**32pts**	**64pts**	**128pts**	**256pts**	**512pts**
d1	0.9 ± 0.1	2.3 ± 1.8	1.4 ± 0.2	1.4 ± 0.4	1.4 ± 0.1	1.4 ± 0.1	1.4 ± 0.1
d2	1.2 ± 0.5	1.4 ± 0.2	1.6 ± 0.3	1.5 ± 0.2	1.5 ± 0.2	1.5 ± 0.1	1.3 ± 0.0
d3	1.3 ± 0.2	4.0 ± 1.9	2.0 ± 0.4	2.0 ± 0.2	1.8 ± 0.2	1.6 ± 0.1	1.4 ± 0.1
d4	3.1 ± 1.7	10.9 ± 9.3	8.1 ± 6.7	7.3 ± 4.6	1.9 ± 0.2	1.6 ± 0.1	1.3 ± 0.1
d5	5.4 ± 8.3	1.6 ± 0.2	1.8 ± 0.6	2.0 ± 0.6	1.8 ± 0.2	1.4 ± 0.2	1.3 ± 0.1
d6	3.5 ± 4.8	4.6 ± 1.8	3.4 ± 0.9	1.7 ± 0.3	1.7 ± 0.6	1.4 ± 0.1	1.3 ± 0.1
d7	2.0 ± 1.5	1.8 ± 0.3	1.6 ± 0.2	1.5 ± 0.1	1.3 ± 0.1	1.4 ± 0.1	1.3 ± 0.1
d8	0.8 ± 0.2	2.9 ± 3.1	1.5 ± 0.2	1.4 ± 0.1	1.4 ± 0.2	1.5 ± 0.1	1.3 ± 0.1
amp	0.7 ± 0.1	0.6 ± 0.1	0.7 ± 0.1	0.8 ± 0.0	0.9 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
$\log \boldsymbol{\sigma}_{\mathbf{n}}^{\mathbf{2}}$	-0.8 ± 0.4	-0.4 ± 0.2	-0.6 ± 0.1	-0.7 ± 0.1	-1.1 ± 0.1	-1.4 ± 0.1	-1.8 ± 0.0

Table A.12: FITC, SYNTH8, Random

	8pts	**16pts**	**32pts**	**64pts**	**128pts**	**256pts**	**512pts**
d1	1.8 ± 1.0	2.1 ± 0.5	4.1 ± 2.6	6.4 ± 2.3	12.5 ± 11.2	11.2 ± 1.8	13.7 ± 1.6
d2	3.2 ± 3.8	8.9 ± 11.7	21.0 ± 15.9	18.4 ± 12.5	6.3 ± 4.7	8.4 ± 1.9	11.2 ± 2.6
d3	1.1 ± 0.7	42.0 ± 67.1	7.5 ± 3.4	5.2 ± 2.6	2.3 ± 1.0	4.3 ± 0.8	3.7 ± 0.3
d4	0.6 ± 0.1	1.6 ± 0.8	1.8 ± 1.6	1.7 ± 0.6	1.7 ± 1.1	2.6 ± 0.3	2.7 ± 0.3
d5	2.1 ± 0.1	8.8 ± 6.6	35.8 ± 32.3	70.5 ± 30.1	80.2 ± 82.4	80.4 ± 60.0	24.5 ± 6.2
d6	3.3 ± 1.6	17.8 ± 20.8	2.0 ± 0.8	2.3 ± 1.4	1.8 ± 1.6	4.0 ± 1.9	1.8 ± 0.1
d7	6.0 ± 6.4	79.2 ± 124.1	24.7 ± 18.9	112.6 ± 174.4	9.1 ± 4.6	15.2 ± 6.0	22.0 ± 7.0
d8	2.5 ± 3.1	11.1 ± 8.4	28.7 ± 30.4	44.7 ± 36.5	54.8 ± 76.9	39.7 ± 36.1	20.6 ± 6.0
d9	7.0 ± 4.6	3.8 ± 3.2	4.4 ± 1.7	5.6 ± 2.6	5.1 ± 2.1	9.8 ± 3.7	7.7 ± 2.1
d10	3.3 ± 2.1	9.3 ± 8.0	21.4 ± 13.2	84.1 ± 83.1	45.2 ± 24.9	238.0 ± 155.6	14.9 ± 9.2
d11	0.7 ± 0.1	17.8 ± 14.9	25.2 ± 21.0	31.9 ± 37.0	19.0 ± 19.0	129.1 ± 16.2	41.4 ± 27.6
d12	1.1 ± 0.5	1.0 ± 0.2	1.6 ± 0.2	4.6 ± 1.9	5.5 ± 4.0	11.8 ± 9.7	9.7 ± 2.8
d13	3.8 ± 6.0	1.5 ± 0.6	95.1 ± 59.7	43.3 ± 58.9	8.5 ± 10.1	13.8 ± 5.1	14.6 ± 3.7
d14	1.1 ± 0.5	16.2 ± 18.1	3.1 ± 1.2	7.7 ± 5.1	28.5 ± 26.9	92.7 ± 36.8	12.7 ± 4.1
d15	2.0 ± 0.7	3.3 ± 1.6	43.5 ± 41.5	76.5 ± 60.2	67.5 ± 22.7	257.2 ± 100.8	50.9 ± 38.1
amp	0.9 ± 0.0	0.7 ± 0.1	0.8 ± 0.1	3.7 ± 3.0	7.5 ± 11.1	28.7 ± 9.7	44.9 ± 12.9
$\log \sigma_{\mathbf{n}}^{2}$	-3.0 ± 0.6	-1.7 ± 1.6	-0.8 ± 0.2	-1.4 ± 0.5	-1.8 ± 0.3	-1.9 ± 0.1	-3.5 ± 0.5

Table A.13: FITC, CHEM, FPC

	8pts	**16pts**	**32pts**	**64pts**	**128pts**	**256pts**	**512pts**
d1	3.2 ± 3.1	3.1 ± 0.7	3.9 ± 0.6	3.7 ± 0.4	4.3 ± 0.2	4.9 ± 0.6	5.3 ± 0.8
d2	6.5 ± 6.0	24.8 ± 34.4	13.8 ± 7.5	16.1 ± 17.4	2.7 ± 1.3	3.2 ± 0.5	4.4 ± 0.4
d3	10.9 ± 18.7	1.5 ± 1.2	2.0 ± 0.4	1.9 ± 0.7	1.4 ± 0.2	1.4 ± 0.1	1.7 ± 0.1
d4	2.5 ± 3.2	1.2 ± 0.9	1.2 ± 0.3	1.0 ± 0.1	1.1 ± 0.1	1.1 ± 0.0	1.5 ± 0.1
d5	5.2 ± 3.2	31.8 ± 21.2	38.3 ± 15.4	19.1 ± 16.7	4.2 ± 0.6	6.0 ± 0.3	7.8 ± 1.3
d6	4.7 ± 4.4	5.8 ± 7.3	0.7 ± 0.3	0.6 ± 0.1	0.6 ± 0.1	0.7 ± 0.0	0.8 ± 0.1
d7	5.5 ± 3.6	37.0 ± 38.4	46.2 ± 45.9	16.3 ± 24.1	3.8 ± 0.6	6.4 ± 0.8	7.0 ± 0.8
d8	10.2 ± 9.6	2.5 ± 1.1	46.2 ± 34.3	18.6 ± 19.6	4.8 ± 1.3	5.4 ± 0.8	6.4 ± 0.6
d9	1.1 ± 0.5	2.4 ± 1.1	2.6 ± 0.4	2.4 ± 0.3	2.1 ± 0.3	1.9 ± 0.2	2.4 ± 0.0
d10	9.9 ± 9.4	19.9 ± 15.2	50.0 ± 28.6	12.2 ± 11.0	11.0 ± 12.2	5.3 ± 0.3	3.9 ± 0.5
d11	8.3 ± 9.8	14.0 ± 19.6	2.9 ± 0.6	4.6 ± 2.9	2.6 ± 0.9	3.1 ± 0.4	4.2 ± 0.6
d12	1.3 ± 0.6	3.9 ± 4.4	2.7 ± 0.3	3.0 ± 0.4	3.0 ± 0.6	3.2 ± 0.4	3.8 ± 0.6
d13	2.3 ± 2.8	2.8 ± 2.1	2.6 ± 0.3	3.0 ± 0.7	3.0 ± 0.3	4.2 ± 0.5	4.6 ± 0.4
d14	1.3 ± 0.5	2.8 ± 2.1	6.1 ± 1.5	5.5 ± 3.4	4.2 ± 1.5	3.6 ± 0.8	4.9 ± 0.6
d15	7.9 ± 6.0	41.0 ± 38.9	75.1 ± 28.8	58.7 ± 38.8	7.7 ± 1.7	9.4 ± 2.6	6.0 ± 0.3
amp	0.8 ± 0.0	0.8 ± 0.1	0.9 ± 0.0	1.1 ± 0.1	1.3 ± 0.1	1.9 ± 0.1	3.9 ± 0.3
$\log \sigma_{\mathbf{n}}^{\mathbf{2}}$	-1.6 ± 0.6	-1.4 ± 0.6	-1.3 ± 0.1	-1.8 ± 0.2	-2.7 ± 0.4	-3.1 ± 0.1	-4.2 ± 0.1

Table A.14: FITC, CHEM, Random

	8pts	**16pts**	**32pts**	**64pts**	**128pts**	**256pts**	**512pts**
d1	166.5 ± 281.1	23.8 ± 11.1	20.0 ± 11.8	27.8 ± 9.4	15.5 ± 6.8	10.7 ± 1.3	6.6 ± 1.6
d2	1190.6 ± 1947.6	184.4 ± 62.3	473.6 ± 261.2	31.0 ± 10.9	13.8 ± 5.6	13.0 ± 2.2	10.9 ± 2.6
d3	693.8 ± 315.7	838.1 ± 944.3	296.7 ± 165.7	70.4 ± 28.6	146.1 ± 89.4	149.6 ± 65.6	46.7 ± 17.9
d4	72.1 ± 63.7	59.3 ± 35.3	23.6 ± 9.2	35.0 ± 24.9	19.7 ± 11.9	9.8 ± 1.9	7.1 ± 2.8
d5	1060.9 ± 630.4	594.7 ± 465.7	475.5 ± 167.0	9.0 ± 4.2	4.8 ± 0.6	2.7 ± 0.5	2.9 ± 1.0
d6	515.9 ± 431.6	408.4 ± 343.4	139.5 ± 148.1	331.6 ± 319.8	132.7 ± 97.0	50.0 ± 22.3	57.9 ± 13.5
d7	558.6 ± 440.2	286.5 ± 345.9	145.2 ± 95.1	190.2 ± 113.0	202.8 ± 195.0	45.9 ± 20.9	32.0 ± 12.3
d8	585.5 ± 617.4	174.2 ± 229.0	199.4 ± 115.9	37.2 ± 44.4	10.2 ± 4.2	10.9 ± 1.3	9.8 ± 3.8
d9	293.2 ± 251.7	292.3 ± 262.8	262.3 ± 347.2	20.5 ± 6.6	24.8 ± 13.2	28.3 ± 6.3	49.2 ± 23.2
d10	435.8 ± 374.2	176.2 ± 204.6	240.4 ± 200.6	84.1 ± 51.8	142.1 ± 69.8	76.7 ± 27.4	38.2 ± 19.6
d11	382.4 ± 376.9	57.4 ± 29.2	91.2 ± 36.1	45.4 ± 33.3	49.7 ± 19.6	7.1 ± 1.7	8.8 ± 1.9
d12	281.6 ± 203.6	116.0 ± 69.8	114.9 ± 81.9	48.4 ± 13.3	128.6 ± 128.7	38.6 ± 13.9	89.3 ± 81.9
d13	2401.4 ± 3276.9	258.1 ± 163.6	192.0 ± 108.7	74.9 ± 40.6	90.5 ± 22.2	104.4 ± 38.0	38.9 ± 16.8
d14	953.0 ± 1130.8	179.3 ± 97.3	136.4 ± 151.8	24.5 ± 22.5	9.9 ± 6.3	3.9 ± 0.5	5.8 ± 2.7
d15	13.8 ± 2.2	13.1 ± 6.3	7.3 ± 2.3	6.6 ± 1.4	3.7 ± 0.8	3.0 ± 0.6	3.6 ± 0.8
d16	543.6 ± 779.2	89.4 ± 102.0	15.3 ± 8.8	97.7 ± 84.8	74.7 ± 42.1	56.0 ± 20.1	27.9 ± 10.1
d17	306.0 ± 369.8	30.4 ± 19.9	15.7 ± 10.7	30.8 ± 5.0	20.9 ± 20.5	8.6 ± 1.0	12.9 ± 2.3
d18	72.0 ± 95.3	5.7 ± 0.9	10.8 ± 4.6	7.0 ± 0.9	5.6 ± 1.0	4.4 ± 1.3	3.0 ± 0.7
d19	178.5 ± 272.2	75.3 ± 50.2	15.7 ± 8.3	17.4 ± 8.9	11.0 ± 6.0	7.4 ± 1.1	19.2 ± 15.0
d20	460.2 ± 521.4	120.6 ± 78.3	44.1 ± 11.1	52.9 ± 15.2	40.2 ± 23.2	24.9 ± 9.0	43.5 ± 17.8
d21	117.9 ± 133.5	67.4 ± 71.2	138.1 ± 217.2	16.7 ± 14.6	4.0 ± 1.3	4.1 ± 0.5	3.0 ± 0.7
amp	8.5 ± 2.7	5.6 ± 1.9	8.7 ± 2.6	7.8 ± 2.8	5.1 ± 2.0	4.2 ± 0.8	8.3 ± 7.0
$\log\sigma_{\mathbf{n}}^2$	−4.1 ± 0.7	−3.9 ± 0.2	−4.1 ± 0.1	−4.1 ± 0.1	−4.3 ± 0.1	−4.5 ± 0.0	−4.5 ± 0.1

	8pts	**16pts**	**32pts**	**64pts**	**128pts**	**256pts**	**512pts**
d1	85117.8 ± 170215.0	21.8 ± 9.4	18.8 ± 8.2	29.6 ± 30.8	23.5 ± 10.6	7.7 ± 4.0	4.1 ± 0.8
d2	1072.7 ± 965.7	128.7 ± 96.7	264.8 ± 306.8	11.5 ± 3.2	24.0 ± 16.5	13.4 ± 4.3	10.5 ± 2.4
d3	$Inf \pm 0.0$	413.0 ± 250.1	270.7 ± 186.4	89.8 ± 46.8	62.3 ± 36.8	34.9 ± 25.1	37.7 ± 19.7
d4	450.8 ± 826.6	40.6 ± 25.9	137.3 ± 128.3	27.6 ± 20.5	16.8 ± 10.0	10.3 ± 8.8	5.4 ± 1.0
d5	694.5 ± 704.1	170.9 ± 165.0	23.7 ± 13.7	10.6 ± 6.2	4.4 ± 0.8	3.6 ± 1.3	2.4 ± 0.2
d6	434.1 ± 431.5	233.4 ± 210.9	350.1 ± 323.5	58.6 ± 45.2	35.9 ± 24.7	36.4 ± 29.2	56.5 ± 36.2
d7	271.3 ± 212.5	126.7 ± 61.1	34.8 ± 17.9	45.9 ± 32.0	22.1 ± 13.4	23.5 ± 16.4	6.0 ± 1.7
d8	782.9 ± 1253.2	57.6 ± 47.3	237.0 ± 230.8	14.3 ± 2.4	16.2 ± 3.0	10.7 ± 5.7	5.4 ± 1.3
d9	252.6 ± 253.9	112.1 ± 64.3	195.2 ± 179.2	17.4 ± 5.2	20.4 ± 7.0	28.8 ± 7.7	34.6 ± 12.9
d10	755.3 ± 1023.4	228.6 ± 205.8	473.2 ± 413.2	139.2 ± 75.2	152.3 ± 109.0	40.4 ± 17.3	30.5 ± 18.8
d11	13141.0 ± 24754.6	822.8 ± 630.3	240.5 ± 324.3	81.7 ± 51.8	48.4 ± 34.3	7.1 ± 2.2	4.7 ± 1.5
d12	446.0 ± 406.1	203.8 ± 204.2	133.4 ± 120.3	40.4 ± 15.5	73.9 ± 14.3	32.7 ± 10.5	37.8 ± 17.1
d13	1172.8 ± 1189.8	395.2 ± 280.5	658.1 ± 357.1	155.2 ± 109.8	49.4 ± 29.6	49.3 ± 59.3	42.8 ± 13.9
d14	1377.1 ± 1884.7	353.5 ± 413.7	588.6 ± 540.2	79.8 ± 77.8	33.5 ± 30.5	5.7 ± 2.3	4.6 ± 1.1
d15	14.6 ± 6.4	9.6 ± 2.4	10.9 ± 6.6	7.9 ± 4.5	4.6 ± 1.1	3.6 ± 0.7	3.5 ± 0.5
d16	86.8 ± 51.1	18.6 ± 11.5	84.1 ± 115.4	35.0 ± 14.5	41.3 ± 1.8	70.3 ± 46.4	40.6 ± 26.2
d17	376.5 ± 402.8	40.2 ± 27.9	78.6 ± 76.2	98.5 ± 100.0	28.6 ± 20.6	12.0 ± 6.1	20.3 ± 17.8
d18	364.1 ± 688.3	15.4 ± 15.9	6.9 ± 2.3	10.4 ± 2.8	6.3 ± 1.6	3.0 ± 0.7	2.9 ± 0.3
d19	127.8 ± 110.4	30.8 ± 10.4	96.1 ± 78.9	44.4 ± 34.5	26.2 ± 12.9	13.4 ± 4.3	18.7 ± 8.1
d20	605.1 ± 645.2	161.2 ± 138.0	68.0 ± 57.2	51.3 ± 39.4	29.2 ± 6.7	27.3 ± 6.8	39.1 ± 12.1
d21	97.3 ± 136.1	96.7 ± 92.8	22.2 ± 12.7	8.3 ± 3.1	5.1 ± 2.3	4.7 ± 0.9	3.9 ± 0.6
amp	6.0 ± 3.2	4.6 ± 1.2	8.0 ± 6.1	4.5 ± 2.7	4.2 ± 1.0	2.3 ± 0.5	2.1 ± 0.3
$\log \sigma_{\mathbf{n}}^{2}$	-3.4 ± 0.8	-3.7 ± 0.2	-3.6 ± 0.1	-3.7 ± 0.1	-3.8 ± 0.0	-4.0 ± 0.0	-4.1 ± 0.0

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	0.9 ± 0.0	0.9 ± 0.0	0.9 ± 0.0	0.9 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d2	0.9 ± 0.0	0.9 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
amp	1.1 ± 0.0	1.2 ± 0.0	1.3 ± 0.0	1.3 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.1
$\log \sigma_{\mathbf{n}}^{\mathbf{2}}$	-13.2 ± 0.0	-13.2 ± 0.0	-13.1 ± 0.0	-13.1 ± 0.0	-13.1 ± 0.0	-13.1 ± 0.0	-13.1 ± 0.0

Table A.17: Local GP, SYNTH2, RPC, joint hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	0.8 ± 0.0	0.9 ± 0.0	0.9 ± 0.0	0.9 ± 0.0	0.9 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d2	0.8 ± 0.0	0.9 ± 0.0	0.9 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
amp	1.1 ± 0.0	1.2 ± 0.0	1.2 ± 0.0	1.3 ± 0.0	1.3 ± 0.0	1.4 ± 0.0	1.4 ± 0.0
$\log \sigma_{\mathbf{n}}^{\mathbf{2}}$	-13.2 ± 0.0	-13.2 ± 0.0	-13.2 ± 0.0	-13.1 ± 0.0	-13.1 ± 0.0	-13.1 ± 0.0	-13.1 ± 0.0

Table A.18: Local GP, SYNTH2, RRC, joint hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	$228.3 \pm Inf$	1.1 ± 0.6	1.0 ± 0.1	0.9 ± 0.0	1.0 ± 0.0	0.9 ± 0.0	1.0 ± 0.0
d2	8.0 ± 50578.9	0.9 ± 0.1	0.9 ± 0.0	0.9 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
amp	1.0 ± 0.4	1.1 ± 0.3	1.1 ± 0.3	1.2 ± 0.3	1.3 ± 0.3	1.3 ± 0.2	1.3 ± 0.2
$\log \sigma_{\mathbf{n}}^{\mathbf{2}}$	-13.2 ± 0.1	-13.2 ± 0.0	-13.2 ± 0.0	-13.2 ± 0.0	-13.2 ± 0.0	-13.1 ± 0.0	-13.1 ± 0.0

Table A.19: Local GP, SYNTH2, RPC, separate hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	$833.9 \pm Inf$	$16.5 \pm Inf$	1.1 ± 0.8	1.0 ± 0.1	0.9 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d2	5.9 ± 6617.7	$39216.9 \pm Inf$	1.0 ± 1.5	0.9 ± 0.2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
amp	1.0 ± 0.5	1.0 ± 0.4	1.1 ± 0.3	1.2 ± 0.3	1.2 ± 0.3	1.3 ± 0.2	1.3 ± 0.2
$\log \sigma_{\mathbf{n}}^{\mathbf{2}}$	-13.3 ± 1.4	-13.2 ± 0.5	-13.2 ± 0.1	-13.2 ± 0.0	-13.2 ± 0.0	-13.2 ± 0.0	-13.1 ± 0.0

Table A.20: Local GP, SYNTH2, RRC, separate hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d3	1.1 ± 0.0	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d4	1.1 ± 0.0	1.1 ± 0.0	1.1 ± 0.0	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d6	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d7	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d8	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
amp	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
$\log \sigma_{\mathbf{n}}^{\mathbf{2}}$	-6.1 ± 1.1	-5.5 ± 0.3	-5.4 ± 0.3	-5.9 ± 0.1	-6.0 ± 0.1	-6.5 ± 0.1	-6.7 ± 0.2

Table A.21: Local GP, SYNTH8, RPC, joint hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d3	1.1 ± 0.0	1.1 ± 0.0	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d4	1.1 ± 0.0	1.1 ± 0.0	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d6	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d7	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d8	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
amp	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
$\log \sigma_{\mathbf{n}}^{2}$	-6.3 ± 0.9	-6.0 ± 0.3	-5.6 ± 0.1	-5.8 ± 0.5	-6.2 ± 0.2	-6.3 ± 0.1	-6.6 ± 0.2

Table A.22: Local GP, SYNTH8, RRC, joint hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	15.4 ± 961.1	4.9 ± 170.8	1.3 ± 4.2	1.1 ± 0.0	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d2	24.2 ± 75072.1	5.5 ± 1495.4	1.1 ± 0.3	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d3	18.0 ± 6111.0	5.6 ± 288.9	1.4 ± 2.3	1.1 ± 0.0	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d4	15.1 ± 1595.7	4.7 ± 130.2	1.6 ± 10.1	1.1 ± 0.0	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d5	16.2 ± 2039.6	4.5 ± 175.1	1.4 ± 4.7	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d6	14.7 ± 1546.7	4.5 ± 140.4	1.9 ± 38.3	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d7	17.4 ± 2935.0	6.0 ± 1042.4	1.2 ± 0.4	1.1 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
d8	13.2 ± 820.7	6.1 ± 923.0	1.2 ± 0.2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
amp	1.1 ± 0.1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
$\log \sigma_n^2$	-6.4 ± 9.4	-5.8 ± 6.8	-5.8 ± 4.3	-6.4 ± 2.8	-6.4 ± 2.1	-6.8 ± 1.6	-7.2 ± 1.2

Table A.23: Local GP, SYNTH8, RPC, separate hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	$19592.8 \pm Inf$	$Inf \pm Inf$	12.0 ± 3029.2	6.9 ± 1651.9	1.1 ± 0.0	1.1 ± 0.1	1.0 ± 0.0
d2	$Inf \pm Inf$	$Inf \pm Inf$	7.7 ± 865.3	4.6 ± 244.0	1.6 ± 10.4	2.1 ± 57.0	1.0 ± 0.0
d3	$Inf \pm Inf$	$12439.1 \pm Inf$	9.6 ± 2355.1	5.8 ± 2342.1	2.3 ± 51.5	1.0 ± 0.0	1.0 ± 0.0
d4	$Inf \pm Inf$	$1185.4 \pm Inf$	9.2 ± 994.5	4.8 ± 1184.6	2.5 ± 59.9	2.1 ± 61.9	1.0 ± 0.0
d5	$Inf \pm Inf$	$Inf \pm Inf$	8.4 ± 1117.5	10.6 ± 10776.4	2.1 ± 51.8	1.0 ± 0.0	1.0 ± 0.0
d6	$Inf \pm Inf$	$Inf \pm Inf$	14.5 ± 22899.0	10.7 ± 9099.3	1.7 ± 40.3	2.8 ± 135.8	1.0 ± 0.0
d7	$83.4 \pm Inf$	$103.8 \pm Inf$	8.0 ± 2456.0	17.1 ± 44689.7	1.3 ± 3.1	1.0 ± 0.0	1.0 ± 0.0
d8	$2681.1 \pm Inf$	$2648.4 \pm Inf$	8.1 ± 966.0	$2274.6 \pm Inf$	1.3 ± 3.0	1.1 ± 0.0	1.0 ± 0.0
amp	1.2 ± 0.4	1.1 ± 0.4	1.1 ± 0.1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
$\log \sigma_n^2$	-7.9 ± 16.3	-7.1 ± 14.8	-6.2 ± 7.3	-6.0 ± 6.1	-6.3 ± 3.7	-6.3 ± 3.1	-6.7 ± 1.5

Table A.24: Local GP, SYNTH8, RRC, separate hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	1.8 ± 0.0	2.1 ± 0.1	2.6 ± 0.1	3.1 ± 0.1	3.8 ± 0.1	4.7 ± 0.1	5.2 ± 0.2
d2	1.9 ± 0.1	2.2 ± 0.1	2.0 ± 0.1	2.1 ± 0.1	2.4 ± 0.1	3.0 ± 0.1	3.4 ± 0.1
d3	0.7 ± 0.0	0.8 ± 0.0	0.9 ± 0.0	1.0 ± 0.0	1.2 ± 0.1	1.3 ± 0.0	1.5 ± 0.1
d4	0.5 ± 0.0	0.5 ± 0.0	0.6 ± 0.0	0.8 ± 0.0	0.9 ± 0.0	1.0 ± 0.0	1.1 ± 0.0
d5	3.2 ± 0.1	3.8 ± 0.4	3.8 ± 0.1	4.0 ± 0.2	4.6 ± 0.2	5.7 ± 0.1	6.5 ± 0.2
d6	0.3 ± 0.0	0.4 ± 0.0	0.4 ± 0.0	0.5 ± 0.0	0.5 ± 0.0	0.6 ± 0.0	0.7 ± 0.0
d7	1.8 ± 0.1	2.5 ± 0.2	2.8 ± 0.1	2.8 ± 0.3	3.3 ± 0.3	3.8 ± 0.1	4.0 ± 0.3
d8	2.2 ± 0.3	2.9 ± 0.3	3.0 ± 0.2	3.3 ± 0.1	3.7 ± 0.1	4.4 ± 0.1	5.3 ± 0.3
d9	0.9 ± 0.0	1.0 ± 0.0	1.2 ± 0.1	1.6 ± 0.0	1.8 ± 0.1	1.9 ± 0.0	2.0 ± 0.1
d10	3.4 ± 0.4	3.2 ± 0.4	2.6 ± 0.1	2.6 ± 0.1	3.0 ± 0.2	2.9 ± 0.2	2.6 ± 0.1
d11	0.8 ± 0.0	0.9 ± 0.0	1.2 ± 0.1	1.5 ± 0.0	1.9 ± 0.1	2.3 ± 0.1	2.4 ± 0.1
d12	1.1 ± 0.0	1.4 ± 0.1	1.7 ± 0.2	2.0 ± 0.1	2.1 ± 0.2	2.4 ± 0.1	2.5 ± 0.1
d13	1.3 ± 0.1	1.7 ± 0.0	2.0 ± 0.1	2.4 ± 0.1	2.8 ± 0.1	3.4 ± 0.1	3.7 ± 0.2
d14	1.1 ± 0.0	1.4 ± 0.0	1.7 ± 0.1	2.2 ± 0.1	2.6 ± 0.1	2.9 ± 0.1	3.0 ± 0.3
d15	3.0 ± 0.2	3.5 ± 0.2	3.5 ± 0.2	3.9 ± 0.1	4.6 ± 0.3	4.7 ± 0.4	3.8 ± 0.4
amp	1.1 ± 0.0	1.3 ± 0.0	1.5 ± 0.1	2.0 ± 0.1	2.7 ± 0.1	4.0 ± 0.2	5.3 ± 0.4
$\log\sigma_{\mathbf{n}}^{\mathbf{2}}$	-12.2 ± 0.5	-11.7 ± 0.3	-10.5 ± 1.2	-9.7 ± 0.9	-9.9 ± 1.0	-8.9 ± 0.3	-9.1 ± 0.6

Table A.25: Local GP, CHEM, RPC, joint hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	2.0 ± 0.0	2.4 ± 0.2	2.8 ± 0.1	3.5 ± 0.0	4.2 ± 0.1	5.0 ± 0.1	5.7 ± 0.1
d2	2.1 ± 0.1	2.1 ± 0.1	2.1 ± 0.1	2.3 ± 0.1	2.7 ± 0.1	3.3 ± 0.0	3.8 ± 0.1
d3	0.7 ± 0.0	0.8 ± 0.0	0.9 ± 0.0	1.1 ± 0.0	1.2 ± 0.1	1.4 ± 0.0	1.6 ± 0.0
d4	0.5 ± 0.0	0.6 ± 0.0	0.7 ± 0.0	0.8 ± 0.0	0.9 ± 0.0	1.1 ± 0.0	1.2 ± 0.0
d5	3.3 ± 0.2	3.7 ± 0.2	4.0 ± 0.2	4.3 ± 0.2	5.2 ± 0.2	6.1 ± 0.2	6.9 ± 0.3
d6	0.3 ± 0.0	0.4 ± 0.0	0.4 ± 0.0	0.5 ± 0.0	0.6 ± 0.0	0.6 ± 0.0	0.7 ± 0.0
d7	2.0 ± 0.1	2.5 ± 0.1	2.8 ± 0.1	3.1 ± 0.1	3.5 ± 0.3	4.0 ± 0.1	3.7 ± 0.2
d8	2.8 ± 0.2	3.2 ± 0.2	3.3 ± 0.1	3.5 ± 0.1	4.1 ± 0.1	4.9 ± 0.1	5.7 ± 0.2
d9	0.9 ± 0.0	1.1 ± 0.0	1.4 ± 0.0	1.7 ± 0.1	1.8 ± 0.0	1.9 ± 0.0	2.1 ± 0.0
d10	2.9 ± 0.4	2.7 ± 0.1	2.7 ± 0.1	2.9 ± 0.2	3.0 ± 0.1	3.1 ± 0.3	2.7 ± 0.2
d11	0.9 ± 0.0	1.1 ± 0.1	1.3 ± 0.0	1.8 ± 0.1	2.1 ± 0.1	2.5 ± 0.1	2.5 ± 0.1
d12	1.3 ± 0.1	1.4 ± 0.0	1.6 ± 0.1	1.9 ± 0.1	2.0 ± 0.2	2.4 ± 0.1	2.4 ± 0.0
d13	1.6 ± 0.0	1.9 ± 0.1	2.2 ± 0.0	2.6 ± 0.1	3.0 ± 0.1	3.6 ± 0.1	4.1 ± 0.2
d14	1.2 ± 0.0	1.6 ± 0.0	2.0 ± 0.1	2.5 ± 0.2	2.9 ± 0.1	3.2 ± 0.2	3.2 ± 0.3
d15	3.1 ± 0.2	3.4 ± 0.1	3.7 ± 0.1	4.3 ± 0.2	4.9 ± 0.2	4.6 ± 0.3	4.0 ± 0.2
amp	1.2 ± 0.0	1.4 ± 0.1	1.7 ± 0.0	2.3 ± 0.1	3.3 ± 0.1	4.8 ± 0.2	6.5 ± 0.3
$\log \sigma_{\mathbf{n}}^{2}$	-12.6 ± 0.5	-11.7 ± 0.2	-11.2 ± 1.1	-9.6 ± 1.2	-10.5 ± 0.8	-8.8 ± 0.1	-9.6 ± 0.7

Table A.26: Local GP, CHEM, RRC, joint hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	$Inf \pm Inf$	42.1 ± 28132.5	11.7 ± 677.6	5.0 ± 6.9	6.2 ± 12.0	6.3 ± 9.1	6.5 ± 5.5
d2	$65.1 \pm Inf$	36.3 ± 9334.9	41.7 ± 43662.8	27.4 ± 3471.1	26.6 ± 4896.1	11.0 ± 446.8	15.6 ± 1017.2
d3	$55.8 \pm Inf$	14.7 ± 4862.6	6.3 ± 1323.9	2.3 ± 47.1	1.6 ± 0.5	1.6 ± 0.3	1.6 ± 0.1
d4	33.5 ± 43997.0	9.0 ± 2103.6	3.9 ± 542.7	2.3 ± 187.2	1.4 ± 0.6	1.4 ± 0.5	1.4 ± 0.2
d5	$Inf \pm Inf$	60.8 ± 62910.3	58.2 ± 41848.3	39.7 ± 5760.3	29.8 ± 3816.4	18.3 ± 881.8	12.6 ± 260.9
d6	$52.1 \pm Inf$	15.6 ± 3965.9	17.7 ± 4108.0	21.4 ± 3329.7	15.8 ± 2209.9	6.5 ± 354.4	5.0 ± 176.4
d7	$186.0 \pm Inf$	50.7 ± 6807.3	$2188.5 \pm Inf$	$79.0 \pm Inf$	8.2 ± 350.4	5.2 ± 5.9	5.4 ± 5.0
d8	$10185.1 \pm Inf$	$53.6 \pm Inf$	30.2 ± 2323.6	12.0 ± 443.2	6.5 ± 74.4	5.8 ± 19.9	6.3 ± 6.0
d9	$58437.2 \pm Inf$	$347.9 \pm Inf$	13.9 ± 1624.0	14.0 ± 1846.4	6.4 ± 142.5	3.6 ± 11.5	3.7 ± 21.8
d10	53.9 ± 35120.0	44.7 ± 5981.0	46.2 ± 62011.1	31.1 ± 6201.6	30.0 ± 5077.5	$57.7 \pm Inf$	23.3 ± 2956.8
d11	$62.1 \pm Inf$	27.4 ± 7766.3	26.8 ± 7149.2	35.9 ± 7889.4	33.3 ± 7633.2	11.7 ± 462.1	19.8 ± 2640.5
d12	$25207.6 \pm Inf$	38.3 ± 11829.5	20.6 ± 2401.6	8.1 ± 734.6	4.7 ± 165.8	3.8 ± 3.0	4.0 ± 3.4
d13	$86.6 \pm Inf$	39.2 ± 6917.2	21.5 ± 2207.9	10.8 ± 912.9	4.4 ± 6.5	5.3 ± 63.0	4.9 ± 3.7
d14	$8400.5 \pm Inf$	31.3 ± 6177.5	28.4 ± 7062.3	31.9 ± 7029.5	37.5 ± 10812.7	$312.8 \pm Inf$	21.4 ± 1949.1
d15	$1444.1 \pm Inf$	52.4 ± 8812.4	54.6 ± 75283.2	38.3 ± 8588.5	35.6 ± 6330.8	31.1 ± 8083.4	16.5 ± 1808.7
amp	1.6 ± 1.5	2.2 ± 2.3	2.9 ± 1.9	3.8 ± 3.8	5.4 ± 6.5	7.1 ± 16.6	10.0 ± 33.6
$\log \sigma_{\mathbf{n}}^{2}$	-11.6 ± 42.0	-12.2 ± 46.7	-13.8 ± 60.1	-14.9 ± 57.9	-15.0 ± 51.4	-14.8 ± 50.7	-13.5 ± 30.9

Table A.27: Local GP, CHEM, RPC, separate hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	$Inf \pm Inf$	$Inf \pm Inf$	$1714.5 \pm Inf$	9.6 ± 1286.3	7.2 ± 75.3	6.8 ± 18.1	7.6 ± 23.1
d2	$Inf \pm Inf$	$Inf \pm Inf$	$49.7 \pm Inf$	30.9 ± 5645.6	23.4 ± 2701.6	17.4 ± 1324.2	10.4 ± 479.3
d3	$Inf \pm Inf$	$102.8 \pm Inf$	$76.5 \pm Inf$	15.7 ± 14671.9	9.1 ± 2205.9	2.1 ± 3.5	2.2 ± 5.6
d4	$Inf \pm Inf$	22.6 ± 35293.7	$25.6 \pm Inf$	11.6 ± 16299.6	$22706.7 \pm Inf$	6.2 ± 1283.6	2.5 ± 46.5
d5	$Inf \pm Inf$	$98.8 \pm Inf$	$75.7 \pm Inf$	46.0 ± 22553.4	29.2 ± 3421.2	18.7 ± 1202.5	18.5 ± 1764.5
d6	$5343.8 \pm Inf$	$52.1 \pm Inf$	$143.2 \pm Inf$	17.9 ± 2138.1	17.9 ± 2503.7	10.1 ± 857.5	4.0 ± 84.4
d7	$Inf \pm Inf$	67.0 ± 48418.4	$74.8 \pm Inf$	21.4 ± 8696.2	$Inf \pm Inf$	8.0 ± 101.0	6.7 ± 60.1
d8	$Inf \pm Inf$	$668.3 \pm Inf$	$203.5 \pm Inf$	20.5 ± 3291.9	15.3 ± 1248.5	11.4 ± 828.6	6.8 ± 71.3
d9	$Inf \pm Inf$	44.8 ± 65266.9	$538.3 \pm Inf$	27.2 ± 24146.9	$62.1 \pm Inf$	5.9 ± 199.4	4.4 ± 83.2
d10	$Inf \pm Inf$	$583.9 \pm Inf$	$67.8 \pm Inf$	38.9 ± 11807.0	$185.2 \pm Inf$	27.6 ± 6167.3	21.4 ± 1917.7
d11	$Inf \pm Inf$	$Inf \pm Inf$	$63.0 \pm Inf$	39.1 ± 10681.6	$94412.8 \pm Inf$	32.5 ± 6183.6	12.1 ± 513.7
d12	$Inf \pm Inf$	$3055.9 \pm Inf$	$148.1 \pm Inf$	13.8 ± 3302.2	14.3 ± 2363.9	7.7 ± 435.9	4.2 ± 3.8
d13	$Inf \pm Inf$	$99.9 \pm Inf$	$54.0 \pm Inf$	26.1 ± 33619.5	$Inf \pm Inf$	15.1 ± 2599.5	5.2 ± 4.9
d14	$Inf \pm Inf$	$94.1 \pm Inf$	52.0 ± 48858.4	58.0 ± 47045.4	$Inf \pm Inf$	29.5 ± 6082.9	15.0 ± 773.6
d15	$Inf \pm Inf$	$188.2 \pm Inf$	$96.0 \pm Inf$	36.1 ± 11671.4	$Inf \pm Inf$	22.6 ± 4547.8	15.7 ± 1044.5
amp	1.9 ± 3.4	2.6 ± 3.7	3.2 ± 3.5	4.3 ± 6.7	6.1 ± 11.8	7.9 ± 25.7	10.1 ± 65.8
$\log \sigma_n^2$	-12.9 ± 106.5	-13.2 ± 59.1	-13.6 ± 61.5	-14.6 ± 56.1	-15.5 ± 91.2	-13.5 ± 39.5	-14.1 ± 44.8

Table A.28: Local GP, CHEM, RRC, separate hyperparameter training

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	2.5 ± 0.0	2.7 ± 0.1	2.9 ± 0.1	3.0 ± 0.2	2.8 ± 0.1	2.8 ± 0.1	2.3 ± 0.1
d2	13.7 ± 0.8	10.3 ± 0.9	8.9 ± 0.4	6.7 ± 0.5	5.5 ± 0.3	4.9 ± 0.3	4.3 ± 0.2
d3	9.0 ± 0.5	10.1 ± 0.5	9.7 ± 0.4	7.1 ± 0.6	6.3 ± 0.2	5.7 ± 0.3	4.3 ± 0.2
d4	4.6 ± 0.2	5.4 ± 0.3	5.7 ± 0.5	4.5 ± 0.4	4.2 ± 0.1	3.9 ± 0.4	3.0 ± 0.1
d5	9.1 ± 1.3	6.4 ± 0.6	4.3 ± 0.4	3.1 ± 0.1	2.3 ± 0.1	1.8 ± 0.2	1.7 ± 0.1
d6	12.6 ± 0.9	14.6 ± 1.1	11.2 ± 1.6	9.2 ± 0.7	7.0 ± 1.2	6.5 ± 0.6	6.1 ± 0.4
d7	8.8 ± 0.5	7.0 ± 0.7	4.9 ± 0.8	3.6 ± 0.3	3.1 ± 0.2	3.1 ± 0.3	2.6 ± 0.1
d8	3.8 ± 0.3	3.2 ± 0.3	2.6 ± 0.3	2.1 ± 0.2	1.6 ± 0.2	1.4 ± 0.1	1.2 ± 0.1
d9	15.2 ± 1.8	12.0 ± 1.7	11.2 ± 1.2	10.1 ± 1.0	8.9 ± 1.2	9.1 ± 0.5	7.6 ± 0.7
d10	7.3 ± 0.6	7.2 ± 0.3	6.8 ± 0.3	5.9 ± 0.3	5.4 ± 0.7	4.7 ± 0.6	4.5 ± 0.4
d11	8.7 ± 1.2	5.2 ± 0.4	3.7 ± 0.1	3.2 ± 0.1	2.7 ± 0.2	2.4 ± 0.2	2.4 ± 0.2
d12	8.7 ± 0.8	9.9 ± 0.6	8.8 ± 1.1	7.8 ± 0.9	6.3 ± 0.8	6.6 ± 0.7	5.7 ± 0.6
d13	17.2 ± 2.4	16.1 ± 1.2	15.4 ± 0.7	13.3 ± 1.5	13.5 ± 1.2	12.9 ± 1.6	12.4 ± 0.8
d14	4.7 ± 0.2	4.3 ± 0.3	3.3 ± 0.1	3.0 ± 0.1	2.5 ± 0.1	2.4 ± 0.2	2.2 ± 0.2
d15	1.4 ± 0.0	1.6 ± 0.0	1.7 ± 0.0	1.7 ± 0.0	1.6 ± 0.1	1.6 ± 0.1	1.6 ± 0.0
d16	8.0 ± 0.4	7.8 ± 0.3	7.5 ± 0.3	7.2 ± 0.4	6.5 ± 0.2	6.4 ± 0.5	6.1 ± 0.4
d17	5.7 ± 0.0	6.1 ± 0.2	6.0 ± 0.4	5.6 ± 0.3	4.9 ± 0.3	5.0 ± 0.3	4.9 ± 0.3
d18	2.8 ± 0.1	2.8 ± 0.1	2.7 ± 0.1	2.6 ± 0.1	2.3 ± 0.1	2.2 ± 0.1	2.2 ± 0.1
d19	7.8 ± 0.5	8.0 ± 0.7	7.4 ± 0.5	7.3 ± 0.3	7.0 ± 0.8	7.3 ± 0.5	7.3 ± 0.4
d20	10.7 ± 0.7	11.8 ± 0.8	11.0 ± 1.1	9.7 ± 0.6	8.8 ± 0.7	9.1 ± 0.8	8.3 ± 0.5
d21	2.9 ± 0.1	2.8 ± 0.1	2.8 ± 0.2	2.7 ± 0.1	2.3 ± 0.2	2.1 ± 0.1	2.1 ± 0.1
amp	1.3 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.3 ± 0.1	1.3 ± 0.0	1.2 ± 0.0
$\log\sigma_{\mathbf{n}}^{\mathbf{2}}$	-5.3 ± 0.0	-5.2 ± 0.0	-5.1 ± 0.0	-5.1 ± 0.0	-5.1 ± 0.0	-5.1 ± 0.0	-5.1 ± 0.0

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	2.4 ± 0.1	2.7 ± 0.1	2.9 ± 0.1	3.1 ± 0.1	2.8 ± 0.1	2.6 ± 0.1	2.3 ± 0.1
d2	14.3 ± 2.3	10.0 ± 0.7	8.3 ± 0.5	6.5 ± 0.4	5.4 ± 0.3	4.6 ± 0.3	4.0 ± 0.3
d3	7.2 ± 0.6	8.2 ± 0.4	8.2 ± 0.2	6.6 ± 0.3	5.5 ± 0.3	4.7 ± 0.4	4.1 ± 0.4
d4	4.2 ± 0.0	5.1 ± 0.4	5.3 ± 0.7	4.4 ± 0.3	4.2 ± 0.6	3.4 ± 0.2	3.0 ± 0.3
d5	9.1 ± 2.0	5.9 ± 0.4	4.3 ± 0.2	3.3 ± 0.2	2.4 ± 0.2	1.9 ± 0.2	1.7 ± 0.1
d6	12.7 ± 1.2	12.2 ± 1.7	10.3 ± 0.9	8.4 ± 0.6	7.4 ± 0.8	6.0 ± 0.4	5.5 ± 0.5
d7	7.9 ± 1.1	5.9 ± 0.6	4.4 ± 0.2	3.4 ± 0.2	2.9 ± 0.2	2.9 ± 0.1	2.7 ± 0.2
d8	2.6 ± 0.1	2.5 ± 0.1	2.0 ± 0.1	1.8 ± 0.1	1.5 ± 0.1	1.3 ± 0.1	1.2 ± 0.1
d9	18.1 ± 1.2	13.7 ± 1.8	11.7 ± 1.7	9.4 ± 0.4	9.2 ± 0.7	8.4 ± 0.6	8.2 ± 0.6
d10	5.7 ± 0.3	6.3 ± 0.4	5.9 ± 0.2	5.4 ± 0.2	4.9 ± 0.3	4.4 ± 0.2	4.2 ± 0.2
d11	4.9 ± 0.3	3.8 ± 0.2	3.3 ± 0.2	2.8 ± 0.1	2.6 ± 0.1	2.4 ± 0.1	2.3 ± 0.1
d12	8.3 ± 0.2	9.1 ± 0.8	9.1 ± 0.8	7.3 ± 0.6	6.6 ± 1.2	6.2 ± 0.2	5.8 ± 0.2
d13	14.1 ± 1.9	14.7 ± 2.5	14.3 ± 1.2	14.8 ± 0.8	13.2 ± 1.3	12.9 ± 0.3	12.5 ± 0.9
d14	3.9 ± 0.1	3.5 ± 0.2	3.1 ± 0.1	2.8 ± 0.1	2.6 ± 0.1	2.3 ± 0.1	2.2 ± 0.1
d15	1.4 ± 0.0	1.5 ± 0.1	1.6 ± 0.1	1.7 ± 0.1	1.6 ± 0.0	1.6 ± 0.1	1.6 ± 0.1
d16	7.5 ± 0.3	7.1 ± 0.2	6.7 ± 0.4	6.8 ± 0.3	6.5 ± 0.2	6.2 ± 0.3	5.9 ± 0.2
d17	5.0 ± 0.2	5.7 ± 0.3	5.9 ± 0.3	5.5 ± 0.4	5.0 ± 0.2	4.8 ± 0.3	4.8 ± 0.3
d18	2.7 ± 0.1	2.8 ± 0.1	2.8 ± 0.1	2.7 ± 0.1	2.4 ± 0.1	2.3 ± 0.1	2.1 ± 0.1
d19	7.5 ± 0.2	7.5 ± 0.5	7.4 ± 0.6	7.4 ± 0.5	7.3 ± 0.2	7.4 ± 0.4	7.2 ± 0.4
d20	10.2 ± 0.8	11.6 ± 0.2	11.1 ± 0.7	9.9 ± 0.5	9.5 ± 1.0	8.7 ± 0.6	8.3 ± 0.7
d21	2.7 ± 0.1	2.8 ± 0.0	2.8 ± 0.1	2.5 ± 0.1	2.2 ± 0.1	2.0 ± 0.1	1.9 ± 0.1
amp	1.2 ± 0.0	1.3 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.4 ± 0.0	1.3 ± 0.0	1.3 ± 0.1
$\log\sigma_n^2$	-5.4 ± 0.0	-5.2 ± 0.0	-5.2 ± 0.0	-5.1 ± 0.0	-5.1 ± 0.0	-5.1 ± 0.0	-5.1 ± 0.0

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	$270.1 \pm Inf$	52.4 ± 33355.9	$54.3 \pm Inf$	13.0 ± 848.5	12.0 ± 5643.6	4.0 ± 8.2	3.0 ± 2.8
d2	$4814.4 \pm Inf$	$173.9 \pm Inf$	$218.8 \pm Inf$	$139.4 \pm Inf$	30.0 ± 11281.5	16.2 ± 1014.1	6.5 ± 40.6
d3	$Inf \pm Inf$	$281.4 \pm Inf$	$1169.9 \pm Inf$	84.2 ± 59594.5	36.8 ± 5675.6	26.5 ± 1536.9	7.9 ± 56.1
d4	$184.2 \pm Inf$	$101.3 \pm Inf$	$304.2 \pm Inf$	46.4 ± 19335.3	30.8 ± 13056.9	9.1 ± 129.1	5.7 ± 20.8
d5	$141.7 \pm Inf$	$103.8 \pm Inf$	$147.1 \pm Inf$	31.2 ± 2599.4	21.5 ± 10359.5	11.7 ± 772.5	2.5 ± 3.4
d6	$633.0 \pm Inf$	$122.9 \pm Inf$	$109.2 \pm Inf$	46.2 ± 12461.9	46.0 ± 55142.0	20.3 ± 558.5	7.5 ± 44.1
d7	$9347.5 \pm Inf$	$131.9 \pm Inf$	$86.0 \pm Inf$	53.6 ± 69725.6	33.0 ± 7032.1	11.8 ± 611.8	5.5 ± 64.8
d8	$Inf \pm Inf$	$197.0 \pm Inf$	67.2 ± 62236.5	$84.3 \pm Inf$	6.2 ± 94.0	3.2 ± 57.6	1.6 ± 0.8
d9	$215.2 \pm Inf$	$196.8 \pm Inf$	$100.3 \pm Inf$	49.1 ± 4786.8	19.6 ± 449.6	16.7 ± 490.8	14.9 ± 265.2
d10	$393.5 \pm Inf$	$174.1 \pm Inf$	$153.1 \pm Inf$	$278.1 \pm Inf$	34.8 ± 6015.6	22.1 ± 3893.2	13.2 ± 432.8
d11	$192.2 \pm Inf$	$144.5 \pm Inf$	$314.0 \pm Inf$	$89.6 \pm Inf$	22.4 ± 4816.8	10.3 ± 388.0	4.6 ± 41.4
d12	$156.2 \pm Inf$	$119.7 \pm Inf$	$278.5 \pm Inf$	52.8 ± 13947.6	49.9 ± 27331.9	15.3 ± 308.3	10.6 ± 91.7
d13	$64077.2 \pm Inf$	$198.5 \pm Inf$	$334.9 \pm Inf$	$124.3 \pm Inf$	71.9 ± 68490.1	35.7 ± 4924.4	26.3 ± 753.1
d14	$499.4 \pm Inf$	$99.6 \pm Inf$	$99.8 \pm Inf$	$69.3 \pm Inf$	12.1 ± 256.4	8.2 ± 154.5	4.2 ± 21.7
d15	$134.7 \pm Inf$	5.8 ± 454.8	2.8 ± 26.8	2.4 ± 5.7	2.2 ± 3.7	2.1 ± 2.6	1.6 ± 1.0
d16	$848.3 \pm Inf$	$191.2 \pm Inf$	$152.2 \pm Inf$	$116.8 \pm Inf$	29.4 ± 3722.0	18.6 ± 617.1	8.6 ± 35.9
d17	$1784.2 \pm Inf$	$42055.7 \pm Inf$	$131.5 \pm Inf$	82.3 ± 37030.8	$67.7 \pm Inf$	22.8 ± 3057.5	6.2 ± 26.9
d18	$346.0 \pm Inf$	$65.3 \pm Inf$	46.9 ± 83468.1	13.3 ± 617.1	6.7 ± 303.7	3.2 ± 5.4	2.5 ± 1.5
d19	$362.4 \pm Inf$	$17307.5 \pm Inf$	118.9 ± 99560.2	65.1 ± 9834.2	$86.4 \pm Inf$	69.5 ± 69304.1	17.2 ± 259.3
d20	$Inf \pm Inf$	$204.4 \pm Inf$	$228.6 \pm Inf$	$123.3 \pm Inf$	$92.7 \pm Inf$	29.2 ± 1699.5	17.8 ± 262.0
d21	$Inf \pm Inf$	$85.1 \pm Inf$	$97.1 \pm Inf$	18.0 ± 1650.7	11.1 ± 600.6	5.7 ± 66.2	2.8 ± 4.5
amp	1.3 ± 1.2	1.5 ± 1.6	1.6 ± 1.4	1.7 ± 2.1	1.5 ± 0.7	1.4 ± 0.6	1.3 ± 0.1
$\log \sigma_{\mathbf{n}}^{\mathbf{2}}$	-8.6 ± 18.1	-6.8 ± 5.2	-6.2 ± 1.9	-5.9 ± 1.5	-5.8 ± 1.1	-5.6 ± 1.0	-5.5 ± 0.4

	32p/c	**64p/c**	**128p/c**	**256p/c**	**512p/c**	**1024p/c**	**2048p/c**
d1	$2686.1 \pm Inf$	$178.4 \pm Inf$	$116.3 \pm Inf$	$72.7 \pm Inf$	17.7 ± 5069.9	23.8 ± 20439.2	3.9 ± 12.1
d2	$Inf \pm Inf$	$18086.5 \pm Inf$	$1919.5 \pm Inf$	$84.7 \pm Inf$	29.5 ± 2310.8	24.4 ± 1752.1	17.6 ± 910.0
d3	$Inf \pm Inf$	$333.1 \pm Inf$	$2770.4 \pm Inf$	$146.4 \pm Inf$	38.2 ± 3570.4	23.8 ± 3498.4	19.7 ± 1624.5
d4	$Inf \pm Inf$	$173.4 \pm Inf$	$Inf \pm Inf$	50.2 ± 13491.5	31.1 ± 2694.9	20.1 ± 2180.8	14.5 ± 541.1
d5	$346.5 \pm Inf$	$139.8 \pm Inf$	$100.7 \pm Inf$	37.7 ± 6435.8	22.3 ± 2804.1	12.3 ± 786.6	8.2 ± 154.8
d6	$Inf \pm Inf$	$102.7 \pm Inf$	$875.7 \pm Inf$	61.6 ± 15555.5	31.6 ± 1835.8	25.6 ± 2805.0	16.2 ± 330.1
d7	$Inf \pm Inf$	$1978.7 \pm Inf$	$579.8 \pm Inf$	51.4 ± 12917.4	29.7 ± 2945.1	18.7 ± 2299.1	15.0 ± 757.1
d8	$13482.6 \pm Inf$	$4535.7 \pm Inf$	78.8 ± 39091.3	52.4 ± 16891.7	17.4 ± 2420.2	4.8 ± 164.2	7.9 ± 628.9
d9	$Inf \pm Inf$	$1345.3 \pm Inf$	$402.6 \pm Inf$	$204.2 \pm Inf$	42.5 ± 15139.6	20.7 ± 1748.0	18.6 ± 871.8
d10	$Inf \pm Inf$	$Inf \pm Inf$	$236.6 \pm Inf$	$216.2 \pm Inf$	47.1 ± 14255.5	20.8 ± 836.4	42.7 ± 9585.3
d11	$Inf \pm Inf$	$42551.4 \pm Inf$	$227.3 \pm Inf$	71.2 ± 59017.8	26.3 ± 3285.5	13.9 ± 1259.7	21.9 ± 2317.5
d12	$Inf \pm Inf$	$129.2 \pm Inf$	$134.7 \pm Inf$	83.9 ± 71073.6	37.2 ± 3335.5	22.8 ± 941.6	22.1 ± 796.0
d13	$Inf \pm Inf$	$1042.5 \pm Inf$	$80795.7 \pm Inf$	$184.1 \pm Inf$	86.6 ± 76855.0	$479.9 \pm Inf$	40.5 ± 3199.5
d14	$Inf \pm Inf$	$1320.7 \pm Inf$	$165.5 \pm Inf$	56.0 ± 21018.7	20.9 ± 1950.1	28.5 ± 23413.4	18.4 ± 1453.8
d15	$Inf \pm Inf$	$14676.0 \pm Inf$	$74.4 \pm Inf$	6.0 ± 525.9	2.7 ± 7.5	2.2 ± 4.3	2.8 ± 13.6
d16	$Inf \pm Inf$	$358.1 \pm Inf$	$9169.2 \pm Inf$	$131.0 \pm Inf$	44.9 ± 5070.8	33.7 ± 7102.9	37.9 ± 6773.9
d17	$Inf \pm Inf$	$1797.6 \pm Inf$	$659.4 \pm Inf$	$426.4 \pm Inf$	$134.9 \pm Inf$	39.5 ± 14637.0	41.0 ± 5662.4
d18	$Inf \pm Inf$	$3019.9 \pm Inf$	$102.9 \pm Inf$	28.2 ± 4293.6	15.5 ± 1346.5	5.2 ± 56.0	6.7 ± 305.5
d19	$Inf \pm Inf$	$787.3 \pm Inf$	$971.4 \pm Inf$	$1262.4 \pm Inf$	81.7 ± 94685.3	60.1 ± 21105.6	49.6 ± 5316.0
d20	$Inf \pm Inf$	$404.9 \pm Inf$	$1027.0 \pm Inf$	$493.0 \pm Inf$	76.3 ± 18236.4	$216.0 \pm Inf$	45.0 ± 5799.1
d21	$Inf \pm Inf$	$Inf \pm Inf$	$167.5 \pm Inf$	$161.9 \pm Inf$	21.8 ± 2626.9	24.9 ± 10522.7	5.7 ± 86.8
amp	1.2 ± 1.1	1.4 ± 1.6	1.6 ± 2.3	1.8 ± 3.5	1.7 ± 1.6	1.6 ± 1.6	1.8 ± 1.5
$\log \sigma_{\mathbf{n}}^{2}$	-9.5 ± 61.4	-7.7 ± 18.6	-6.9 ± 14.0	-6.1 ± 2.0	-5.8 ± 1.3	-5.8 ± 1.1	-5.6 ± 0.8

Appendix B

Appendix: The Code

This appendix describes the code we have used and created: it is intended as a guide should anyone want to repeat or modify the experiments we have performed. Figure B.1 presents the (pruned to contain only the useful information) directory tree of our codebase. The following two sections describe the contents of the two main directories containing the approximation implementations (Section B.1 and the experiments code (Section B.2).

```
/
├── code
│   ├── drtoolbox
│   ├── figtree-0.9.3
│   ├── gpml
│   └── project
│       ├── inducing
│       ├── iterative
│       ├── local
│       └── sod
└── evaluations
    ├── SYNTH
    ├── CHEM
    ├── SARCOS
    └── analysis
        ├── SYNTH2
        ├── SYNTH8
        ├── CHEM
        ├── SARCOS
        ├── SoD
        ├── Local
        ├── FITC
        ├── IFGT
        ├── clusters
        └── dissertation
```

Figure B.1: Directory tree containing the code for approximation algorithms and experiments discussed in this dissertation. The contents of the most important directories are discussed in this appendix.

B.1 `/code`: approximation implementations and external source code

This directory contains some open source code written by other researchers, as well as our own code:

`drtoolbox` : Matlab Toolbox for Dimensionality Reduction, v.0.7.2 (van der Maaten [2010]) that we used to project the datasets we use into two dimensions.

`figtree-0.9.3` : FIGTree, v0.9.3 (Morariu [2010]) that implements the **Improved Fast Gauss Transform** and **Farthest Point Clustering** algorithms.

`gpml` : GPML, v.3.1 (Rasmussen and Nickisch [2010]) that implements the full **Gaussian Process Regression** algorithm, the **Squared Exponential** covariance function and the **FITC** approximation.

`project` : Our code, implementing the **Subset of Data**, **Local GP** and **GP with CG/fast MVM** approximations. This code relies on the packages listed above. In addition, this directory contains `inducing`, where we implemented **Random Recursive Clustering**, **Random Projection Clustering** and three inducing points choice methods: **Recursive Random**, **FPC centers** and **Informative Vector Machine** (these are all described in Chapter 4).

B.2 `/evaluations`: the experiment and analysis framework

The dataset-named directories contain the full datasets we use. It is not clear whether we can release `CHEM`, `SYNTH2` or `SYNTH8` to the public yet, so only `SARCOS` should be included with this code at the moment. `evaluations` also contains scripts that automatize testing the methods, as well as scripts that perform analysis and plotting on the gathered data:

`evaluations` the main **test scripts: `testSod.m`, `testLocal.m`, `testFITC.m`, `testIFGT.m`** are here. They demonstrate how to use the code in `code` directory to perform approximate GP regression with varying accuracy/complexity trade-offs, and store the data gathered in this way for use with our analysis scripts.

`analysis` The directories named after the datasets contain scripts that plot the algorithms' **time vs error** performance on each dataset.

The directories named after the methods plot each method's **complexity vs error** performance (that is, plot the SMSE and MSLL as a function of the inducing points count, cluster size, or allowed MVM error).

`clusters` contains scripts that look at some **cluster characteristics** of the data. Note that more of those is contained in the `Local` directory (understanding the Local GP approximation required looking at data clustering in some detail).

Finally, `dissertation` contains the source code for **this dissertation**, in case anyone needs to find the sources for any of the figures we included in this work.

Bibliography

F. J. Anscombe. Graphs in Statistical Analysis. *The Americal Statistician*, 27(1):17–21, 1973.

C. Archambeau, D. Cornford, M. Opper, and J. Shawe-Taylor. Gaussian Process Approximations of Stochastic Differential Equations. In *JMLR: Workshop and Conference Proceedings 1: 1-16*, 2007.

C. M. Bishop. *Patter Recognition and Machine Learning*. Springer Science+Business Media, 2006.

M. Gibbs and D. J. C. MacKay. Efficient Implementation of Gaussian Processes, 1997. Technical Report.

T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

E. Jackson, M. Davy, A. Doucet, and W.J.Fitzgerald. Bayesian Unsupervised Signal Classification by Dirichlet Process Mixtures of Gaussian Processes. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2007.

A. Krause. Near-optimal sensor placements in Gaussian Processes. *ICML*, 2005.

N. D. Lawrence, M. Seeger, and R. Herbrich. Fast Sparse Gaussian Process Methods: The Informative Vector Machine. *Advances in Neural Information Processing Systems*, 15, 2003.

M. Lazaro-Gredilla, J. Quinonero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal. Sparse Spectrum Gaussian Process Regression. *Journal of Machine Learning Research*, 11, 2010.

D. J. C. MacKay. *Information Theory, inference, and learning algorithms*. Cambridge University Press, 2003.

M. Malshe, L. Raff, M. Rockley, M.Hagan, P. M. Agrawal, and R. Komanduri. Theoretical investigation of the dissociation dynamics of vibrationally excited vinyl bromide on an ab initio potential-energy surface obtained using modified novelty sampling and feedforward neural networks. II. Numerical application of the method. *The Journal of Chemical Physics*, 127, 2007.

V. Morariu. FIGTree: Fast Improved Gauss Transform with Tree Data Structure version 0.9.3, 2010. URL http://www.umiacs.umd.edu/~morariu/figtree/.

V. Morariu, B. V. Srinivasan, V. Raykar, and R. Duraiswami. Automatic online tuning for fast Gaussian summation. *Advances in Neural Processing Systems*, 2008.

I. Murray. Gaussian processes and fast matrix-vector multiplies, 2009. Presented at the Numerical Mathematics in Machine Learning workshop at the 26th International Conference on Machine Learning (ICML 2009), Montreal, Canada. URL http://www.cs.toronto.edu/~murray/pub/09gp_eval/ (as of March 2011).

D. Nguyen-tuong, J. Peters, M. Seeger, and B. Schlkopf. Learning Inverse Dynamics: a Comparison. In *Proceedings of 16th European Symposium on Artificial Neural Networks*, Bruges, Belgium, 2008.

J. Quinonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian Process regression. *Journal of Machine Learning Research*, 6:1939–1959, December 2005.

J. Quinonero-Candela, C. E. Rasmussen, and C. K. I. Williams. *Large Scale Kernel Methods*, chapter Approximation Methods for Gaussian Process Regression. MIT Press, Cambridge, MA, 2007.

C. E. Rasmussen. The Gaussian Process Web Site, 2011. URL http://www.gaussianprocess.org.

C. E. Rasmussen and H. Nickisch. Gaussian Process Regression and Classification Toolbox version 3.1, 2010. URL http://www.gaussianprocess.org/gpml/code/matlab/doc/.

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

V. C. Raykar and R. Duraiswami. Fast large scale Gaussian Process regression using approximate matrix-vector products. In *Learning workshop*, San Juan, Puerto Rico, 2007.

V. C. Raykar, C. Yang, R. Duraiswami, and N. Gumerow. Fast computation of sums of gaussians in high dimensions, 2005. Technical Report CS-TR-4767, Dep. of Computer Science, University of Maryland, College Park.

B. Scholkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.

E. Snelson. Matlab code for Sparse pseudo-input Gaussian processes (SPGP), 2006. http://www.gatsby.ucl.ac.uk/~snelson/SPGP_dist.tgz.

E. Snelson, C. E. Rasmussen, and Z. Ghahramani. Warped Gaussian Process. In *Advances in Neural Information Processing Systems*, volume 14, 2004.

E. L. Snelson. Flexible and efficient Gaussian process models for machine learning, 2001. DPhil dissertation, University of London.

N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.

L. van der Maaten. Matlab Toolbox for Dimensionality Reduction, v0.7.2, 2010. URL http://homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction.html.

L. van der Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

V. N. Vapnik. An Overview of Statistical Learning Theory. *IEEE Transactions on Neural Networks*, 10(5), 1999.

C. Yang, R. Duraiswami, and L. Davis. Efficient Kernel Machines Using the Improved Gauss Transform. In *Advances in Neural Information Processing Systems*, volume 17, 2004.