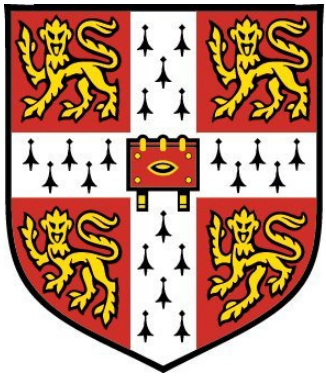
Effective implementation of Gaussian process regression for machine learning



Alexander Davies

Department of Engineering,
University of Cambridge

A thesis submitted for the degree of
Doctor of Philosophy

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains less than 65,000 words including appendices, bibliography, footnotes, tables and equations and has less than 150 figures.

CONTENTS

1

ABSTRACT

This thesis presents frameworks for the effective implementation of Gaussian process regression for machine learning. It addresses this in three parts: effective iterative methods for calculating the predictive distribution and derivatives of a Gaussian process with fixed hyper-parameters, defining three broad classes of kernels of controllable complexity that allow for an order of magnitude scaling in the previous framework and an investigation into alternative objective functions and improved derivatives for the optimization of model hyper-parameters.

2

INTRODUCTION

This thesis is about the effective implementation of Gaussian process regression (GP regression). GP regression is a non-linear regression technique, whose major drawback is that the computational cost of a naive implementation is $O(N^3)$ operations, where N is the number of training points. We show how an effective implementation can reduce this cost to $O(NMI)$ operations, where M and I are related to the complexity of the type of function being learnt and the difficulty of the particular regression problem respectively. We provide a number of classes of useful functions with small M and explore methods for reducing I as much as possible.

2.1 REALITIES OF SCALE

Effective implementation of GPs are important because, without some unexpected revolutions in underlying computational power, it will never be feasible to apply an algorithm that requires $O(N^3)$ operations to datasets of the size that are now common in large scale applications. To illustrate this point, take an example of a dataset with an entry for every person in the world, $N = 7$ billion. This would be considered relatively large, yet still much smaller than the current largest tasks of $N \approx 100$ billion. Currently, the fastest super-computer in the world is the Tianhe-2 **?**, a supercomputer developed by China's National University of Defense Technology, which has a listed peak operation rate of 33.86 petaflops/s. The time required to performing N^3 operations on this dataset, on this machine would be on the order of 10^{17} seconds, which is the current estimated age of the universe Collaboration [2013].

It is commonly accepted that we are unlikely to ever be able to apply algorithms of cubic scaling to dataset of this size and that in most cases quadratic scaling is also too expensive. Ideally, implementations of regression algorithms should be able to scale linearly in N, which is what we present in this thesis.

2.2 A CAUSE FOR HOPE

In this thesis we uncover the hidden assumptions that underlie the standard implementations of Gaussian processes to show how we can expect to reliably perform inference faster than the $O(N^3)$ computational bound. The overarching theme can be summarized in two points.

Firstly, we don't need exact answers. The statement that GP inference requires $O(N^3)$ operations assumes that we need answers to machine precision, which is rarely true in practice. Secondly, $O(N^3)$ is a worst case bound. There are *certain* problems for which $O(N^3)$ operations are required. However, this does not discount the possibility that the are a large set of problems that can be solved more efficiently, and moreover *these are the ones we are actually likely to encounter*. We can either believe that we live in a generally safe and predictable universe, so that the problems we want to solve are (or are very close to) the easy ones, or more pessimistically we can simply be content that the best we can ever do for large scale problems is focus on this efficient subset.

2.3 A FRAMEWORK, NOT ANOTHER APPROXIMATION

Until now, the issue of computational cost has largely been addressed by constructing approximations to the full GP model and using these models in place of the full GP. These approximations exploit the observation in the previous section - they correspond to problems for which $O(N^3)$ operations are not required. When judiciously chosen, these approximations can be calculated substantially faster than a full GP for, in some cases, little loss in accuracy. However, if the approximation chosen is poor, the result can be arbitrarily bad compared to the full GP begin approximated. Moreover, most implementations calculate predictions to machine precision, when this is often not required and a sizeable computational burden.

Instead, what this thesis outlines is a general framework for implementation. This includes a unified computational specification that allows for the calculation of quantities only to the required precision. It provides for the incorporation of most existing GP approximation methods in one of two ways: either directly using the model, as is already done, or by using the approximation to aid in the solution of the full GP. We show examples where using the approximate GP methods in this way allows us to quickly solve the full GP to a bounded error tolerance.

Arising naturally from this framework is a definition of the "complexity" of a kernel, with respect to its computational cost. We define the concept of M-efficient kernels, showing that when interpreted as exact kernels, most existing GP approximations fall under this definition.

Also, too often ignored from the discussion of efficient GP implementations (with the notable exception of), is the computationally efficient optimization of the hyper-parameters. We show again how our framework extends naturally to this optimization in a way that removes the sharp delineation between the solution of a GP and the optimization of its hyper-parameters.

2.4 SCOPE OF EXPERIMENTS

This thesis argues for an effective framework for the implementation of Gaussian processes. By effective, we refer to the required amount of computation for calculating the ultimately desired quantities. We prove this effectiveness by cal-

culating the required number of computations for the different quantities within the framework. In this sense, the main contribution of the thesis is laying down this theoretical foundation.

Throughout the thesis is experimental validation of all these quantities on a variety of datasets. Due to implementation considerations, the experiments were run on relatively small datasets for which extensive analysis could be performed, which is not ideal given that the effective implementation of methods is most important in the regime of large amounts of data. However, in all aspects of the framework, the theoretical guarantees for the computational requirements depend exactly linearly in the size of the data. Further, most experiments are much more representative of the relationship between the kernel and dataset being tested than to any inherent properties of the underlying framework. Thus we encourage the reader to consider all the experiments as basic validation of the theoretical assurances of the underlying framework, and as a rough guide to the general performance of the framework.

2.5 OUTLINE

The thesis consists of six chapters, each covering different aspects of effectively implementing GPs.

Chapter 2 provides a background to GPs by first introducing probabilistic regression, the problem that we attempt to solve with GPs, followed by kernel methods, of which GPs are one. It then covers how exact inference in GPs is traditionally implemented, at a computational cost of $O(N^3)$ operations.

Chapter 3, Iterative GPs, investigates an alternative framework for inference in GPs that reduces the cost of exact inference from $O(N^3)$ to $O(N^2 I)$ operations, where I is dataset dependent. It also shows how approximations with bounded error on final predictions can be introduced to reduced I.

Chapter 4, Fast kernels, describes a number of broad classes of kernels with expressive modelling ability that allow inference to be performed in $O(NMI)$ operations, where M depends on the complexity of the function class.

Chapter 5, Effective optimization, shows how to maintain these efficiency gains when optimizing kernel hyper-parameters.

Chapter 6 reviews the contributions of the three prior chapters, and outlines areas of future work

2.6 PRACTITIONER ROADMAP

As a further assistance to those implementing GP regression, the following is an outline of the key stages to consider with references to details in the thesis.

Stage 1: Select an appropriate kernel. This is by far the most important step for accurate predictions, as all the domain knowledge for the problem needs to

be encoded here. Properties of the function such as smoothness, periodicity, linearity, dependence between dimensions and many more can be encoded by the kernel. There is a vast literature on different kernels that all have different properties. The kernel should be chosen such that it reflects the known properties of the function and also such that it has an appropriate value of M, as defined in Chapter 5. Larger values of M tend to imply a richer function space, at a higher computational cost. For example, in the case of a linear kernel $M = D$ (which is often, though not always, much smaller than N), whereas the squared exponential kernel $M = N$ (which is the largest possible value of M). For convenience, we have defined three classes of kernels with controllable M in Chapter 5.

Stage 2: Find a computationally efficient approximation to use as a pre-conditioner (see Section 4.7). In the same way that there is no "best" kernel, only different encoding of prior knowledge about the function, there is no "best" approximation, only different encodings of knowledge about the computational structure of the problem. There is again a vast literature of approximations, usually termed sparse GP methods or efficient GP methods.

Stage 3: Determine the required accuracy for the posterior mean and posterior variance. This is described in Section4.3 and Section 4.4.

Stage 4 (Optional): If hyper-parameters need to be optimized, select an optimization objective from Section 6.1. The log-likelihood is the traditional choice, though for applications where predictive performance is the goal, our proposed validation error may also be a good choice. If attempting to optimize a kernel with complicated structure, consider pre-conditioning the hyper-parameter optimization as detailed in (Section 6.2).

Stage 5 (Optional): Run pre-calculations to increase speed of posterior variance calculation. In most cases, this will be the SVD pre-calculations described in Section 4.4, with as many singular vectors as is computationally feasible. If this is not possible, then the subset of data method should be used instead.

3

BACKGROUND

In this section we provide a brief background to the important concepts in this thesis. Gaussian Processes are a technique for performing *probabilistic regression*, so we first cover the definition of this problem. Secondly, Gaussian processes are a *kernel method*. We will introduce the concept of a kernel method and primal and dual representations. Finally we will define Gaussian processes and show the implementation upon which we improve in this thesis. For a more comprehensive introduction to GPs and their applications refer to [Rasmussen and Williams, 2006].

3.1 PROBABILISTIC REGRESSION

Regression, in it's simplest form, is the task of learning a function $f : \mathcal{I} \to \mathbb{R}$ (where $\mathcal{I}$ is the space on which we are performing regression), having been shown some evaluations of the function $\{(x_i, f(x_i))\}_{1:n}$. In probabilistic regression on the other hand, the task is to a learn a probability distribution over possible functions, having been shown those same function evaluations.

The main advantage of probabilistic regression is that it is a principled manner of quantifying the uncertainty of our knowledge about the true function based on the observations we have been given. This uncertainty can then be translated to uncertainty about the quantities that we use our regression to calculate. For example, regression is often performed so that predictions can be made at a "test point" x_*: a point in the input space where no observation has been made. With a probabilistic regression technique, the result is a distribution over possible values of $f(x_*)$, rather than a single estimate. Figure 1 illustrates the utility of this view: there are points (such as point A) where we can be fairly certain of the function value, and others where we have very little idea of the true value (point B).

In our treatment of probabilistic regression we assume that there exists a true underlying function $f : \mathcal{I} \to \mathbb{R}$ that we are trying to learn. We assume that the examples we observe have been corrupted by independent Gaussian noise, so rather than observing pairs $\{(x_i, f(x_i))\}_{1:n}$, we observe $\{(x_i, y_i)\}_{1:n}$ from the following model:

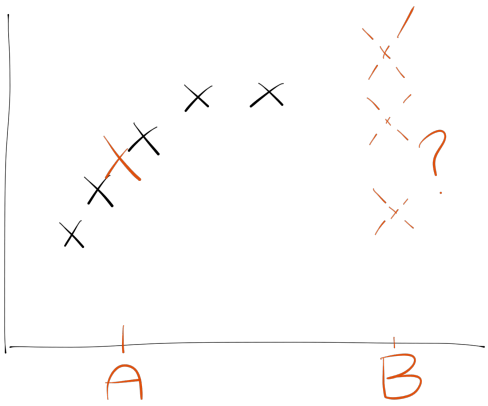


Figure 1: Simple 1D regression problem

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{1}$$
$$y_i = f(x_i) + \epsilon_i \tag{2}$$

3.2 KERNEL METHODS

For a large class of problems in machine learning, there are two different yet equivalent ways to represent data: either as a set of features, or as a set of similarities between all the points. The features are known as the *primal* view and the similarities is the *dual* view. The term kernel methods refers to methods that are defined using the dual view of the data. For a more in-depth introduction see [Shawe-Taylor and Cristianini, 2004][Herbrich, 2002].

3.2.1 *Dual representations*

Data in statistics and machine learning are normally described by vectors in an *input space* $\mathcal{I}$. Each element of this vector is an *input feature*. A dataset with N datapoints is usually represented by these vectors stacked in an $N \times D$ *design matrix* $\mathbf{X}$.

$$x_0 = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} 0 & 1 & 2 \\ 8 & 2 & 3 \\ 4 & 2 & 3 \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{3}$$

However, the input space is often not a good representation of the data for performing effective machine learning. In these cases we attempt to find a better

representation in a *feature space* $\mathcal{F}$; this is just the name for the different representation. This is represented by a non-linear mapping $\phi : \mathcal{I} \to \mathcal{F}$, which is refered to as the feature mapping. Just as we constructed the design matrix for the input space, we also construct the $N \times D'$ *feature matrix* $\mathbf{\Phi}$, where $\mathbf{\Phi}_i = \phi(\mathbf{X}_i)$.

$$\phi(x_0) = \begin{bmatrix} 0 & 8 & 8 & 3 & 4 \end{bmatrix} \qquad \mathbf{\Phi} = \begin{bmatrix} 0 & 8 & 8 & 3 & 4 \\ 5 & 1 & 2 & 7 & 9 \\ 0 & 3 & 15 & 7 & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{4}$$

This is the first representation. This data representation $\mathbf{\Phi}$ is also known as the *primal representation*.

For a large class of machine learning algorithms[1] there is an alternative representation that can be used, called the *dual represenation*. This dual representation is the *Gram matrix*.

$$\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\intercal \qquad \mathbf{K} = \begin{bmatrix} 153 & 81 & 161 & \dots \\ 81 & 160 & 73 & \dots \\ 161 & 73 & 284 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{5}$$

This derivation of the dual has showed us how to move from the input space to a feature space and then from that feature space to the dual representation. It is also possible to move directly from the input space to the dual of a feature space through the use of a *kernel function* $k : \mathcal{I} \times \mathcal{I} \to \mathcal{R}$. A kernel function takes two input points as arguments and returns the corresponding value of $\mathbf{K}$, ie $\mathbf{K}_{ij} = k(\mathbf{X}_i, \mathbf{X}_j)$. This value can be thought of as the similarity between two points. The relationship between the input space, primal and dual representation is illustrated in Figure 2.

The relationship between primal and dual is not 1-to-1: the primal uniquely specifies the dual, but the dual only specifies the primal up to unitary transformations (such as rotations or reflections). A unitary matrix $\mathbf{U}$ (which has the property $\mathbf{U}\mathbf{U}^\intercal = \mathbf{I}$) applied to a primal representation results in the same dual:

$$\tilde{\mathbf{K}} = \mathbf{\Phi}\mathbf{U}\,(\mathbf{\Phi}\mathbf{U})^\intercal = \mathbf{\Phi}\mathbf{U}\mathbf{U}^\intercal\mathbf{\Phi}^\intercal = \mathbf{\Phi}\mathbf{\Phi}^\intercal \tag{6}$$

This means that for a given $\mathbf{\Phi}$ it is easy to construct the unique $\mathbf{K}$ with which it is associated, but for a given $\mathbf{K}$, there are many such feature spaces.

When such a dual representation exists, there are two potential benefits. Firstly, the computational cost of calculating the solution using the primal or dual representation is usually different. We can benefit from this by simply choosing the

1 Those that are invariant to rotations of the dataset.

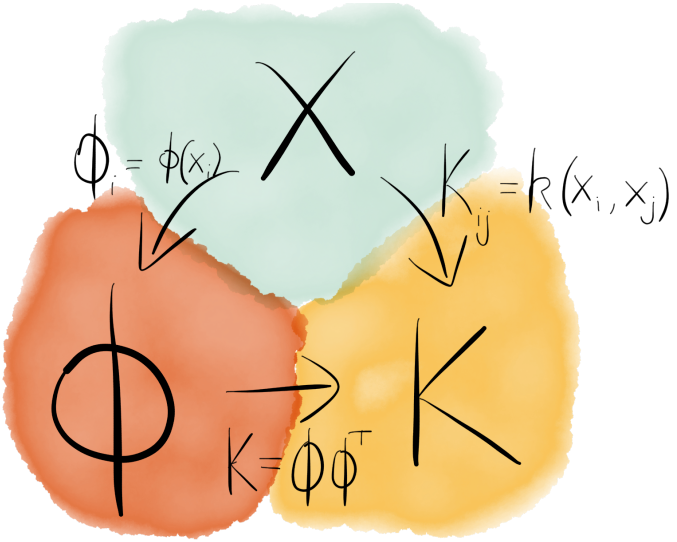


Figure 2: Relationship between the input space, the primal representation and the dual representation

representation depending on which is more efficient in a particular instance, an example of which is shown in Section 3.2.2. Secondly, it allows us to choose which of the two representations of the problem (ϕ or k) is more intuitive to specify. In some cases — say where the input space is the set of strings — it may be far easier to specify the similarity of a string with another, rather than a mapping it into $\mathbb{R}^m$ in a way that meaningfully encodes our knowledge about that string.

3.2.2 *Example: Ridge regression*

Ridge regression (also known as L_2-Penalized Linear Regression) is an example of a technique that has both a primal and dual representation. The model is the same as basic linear regression, but with the addition of a regularizing penalty term on the regression co-efficients in the loss function:

$$\arg\min_{\vec{\beta}} \|\vec{y} - \mathbf{\Phi}\vec{\beta}\|^2 + \lambda\|\vec{\beta}\|^2 \tag{7}$$

The more common closed form solution for the regression co-efficients is:

$$\vec{\beta} = (\lambda\mathbf{I} + \mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\vec{y} \qquad \text{Primal representation} \tag{8}$$

However, if we apply a version of the Woodbury matrix inversion lemma, we get the equivalent:

$$\vec{\beta} = \mathbf{\Phi}^\intercal \left(\mathbf{\Phi}\mathbf{\Phi}^\intercal + \frac{1}{\lambda}\mathbf{I} \right)^{-1} \vec{y} \tag{9}$$

$$= \mathbf{\Phi}^\intercal \left(\mathbf{K} + \frac{1}{\lambda}\mathbf{I} \right)^{-1} \vec{y} \qquad \text{Dual representation} \tag{10}$$

Where $\mathbf{I}$ is the identity matrix of appropriate size. The computational complexity of solving these equations is governed by the size of the matrix within the inversion; since $(\lambda\mathbf{I} + \mathbf{\Phi}^\intercal\mathbf{\Phi})$ is a $D \times D$ matrix, the cost of solving Eq 8 is $O(D^3)$, while Eq 9 relies on the $N \times N$ matrix $\left(\mathbf{\Phi}\mathbf{\Phi}^\intercal + \frac{1}{\lambda}\mathbf{I}\right)$, which requires $O(N^3)$ operations to invert.

3.2.3 *Properties of Gram matrices and kernels*

From the definition of the Gram matrix in the previous section, we can see that it belongs to a restricted class of matrices: those that can be represented as $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\intercal$. These matrices are known as *positive semi-definite* (or PSD for short). They are loosely speaking the analogue of positive numbers, as extended to matrices. Aside from the property that they can be represented as $\mathbf{\Phi}\mathbf{\Phi}^\intercal$, they have two other properties that we will use many times:

1. All of their eigenvalues are positive.
2. $\vec{v}^\intercal\mathbf{K}\vec{v} \geqslant 0 \ \forall \vec{v}$

Kernels are the class of functions that can be used to construct a positive semi-definite matrix and are defined as follows:

Definition 1. A kernel is a function $k(\cdot,\cdot) \to \mathcal{R}$ that satisfies the following criteria:

$$\begin{aligned} k(x',x) &= k(x,x') && (Symmetry) \\ k(ax,bx') &= abk(x,x') && (Bilinearity) \\ k(x,x) &\geqslant 0 \\ k(x,x) &= 0 \iff x = 0 \end{aligned}$$

In most cases the kernels used in Gaussian processes are parameterized by a set of variables $\vec{\theta}$.

3.3 GAUSSIAN PROCESSES

Gaussian processes are one method for performing probabilistic regression. Originally pioneered in Geophysics [Matheron, 1963] and Statistics[O'Hagan and Kingman, 1978], they have subsequently gained a strong following in machine learning.

When viewed as a probabilistic regression method, a Gaussian process takes as input a dataset and a kernel, and as output gives a distribution over functions. The kernel defines the functions that are likely before having seen any data — for example that smooth functions are more likely than sharply discontinuous functions.

The Gaussian process model for regression is defined as follows:

$$f(x) \sim \mathcal{N}(\vec{0}, \mathbf{K}_{-}) \tag{11}$$

where $\mathbf{K}_{-}$ is a kernel matrix: $(\mathbf{K}_{-})_{ij} = k(x_i, x_j)$. This formulation has a $\vec{0}$ mean, which is not the most general formulation, but due to **??**, for most practical applications this suffices. Any results however, can be extended to a GP with a non-zero mean function.

Since our observations $\vec{y}$ are corrupted by independent Gaussian noise, there is a simple closed form for

$$\vec{y} \sim \mathcal{N}(\vec{0}, \mathbf{K}) \tag{12}$$

where $\mathbf{K} := \mathbf{K}_{-} + \sigma^2\mathbf{I}$.

Here, k is a kernel and σ^2 is the noise level. This model can be used to make inferences about many quantities of an underlying function — derivatives at points, integrals over sections to name a few. However, the three quantites that are relevant for probabilistic regression are the predictive distribution, the log-likelihood and the derivative of the log-likelihood with respect to the kernel parameters.

The predictive distribution shown in Equation 13, is the distribution on the value of an unknown datapoint, given the observed training pairs. In a Gaussian process this distribution is a 1D Gaussian, and so is specified by its posterior mean and posterior variance. Since the distribution is Gaussian, the posterior mean is also the most likely value (MAP) of y for that datapoint. This makes it a good candidate if we need to make a point prediction, and it is the optimal choice under the assumption of a squared-loss error metric. The posterior variance provides a measure of the uncertainty in the prediction at a given test point. It can be used to calculate confidence bounds as the model predicts a 95% chance that the point will be within the interval $[\mu - 2\sigma, \mu + 2\sigma]$.

The log-likelihood (Equation 14) is the log of the probability that the model generated the observed data. This quantity is generally used to make comparisons between different models — those with a higher value of the log-likelihood are more likely to be a good model of the data. Commonly in Gaussian process regression, we choose between different GP models (ie ones with different kernel) by picking one with the highest value of the log-likelihood. It is possible to calculate the derivative of the log-likelihood with respect to parameters of the kernel (Equation 15), which allows us to perform gradient optimization of the hyper-parameters to find a kernel that is most appropriate for our data.

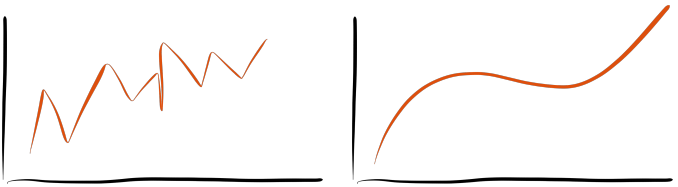
$$\vec{y}_* | \vec{y} \sim \mathcal{N}(\mathbf{K}_{*x}\mathbf{K}^{-1}\vec{y}, \mathbf{K}_* - \mathbf{K}_{*x}\mathbf{K}^{-1}\mathbf{K}_{x*} + \sigma^2\mathbf{I}) \tag{13}$$

$$\mathcal{L} = -\frac{1}{2}\vec{y}^\intercal\mathbf{K}^{-1}\vec{y} - \frac{1}{2}\log(\det(\mathbf{K})) - \frac{N}{2}\log(2\pi) \tag{14}$$

$$\frac{d\mathcal{L}}{d\theta} = -\frac{1}{2}\vec{y}^\intercal\mathbf{K}^{-1}\frac{d\mathbf{K}}{d\theta}\mathbf{K}^{-1}\vec{y} - \frac{1}{2}Tr\left(\mathbf{K}^{-1}\frac{d\mathbf{K}}{d\theta}\right) \tag{15}$$

3.3.1 *Interpretation of kernels*

The only parameter of a Gaussian process is a kernel (though the kernel itself may have parameters). The choice of kernel defines the class of functions which regression is performed over. It not only specifies which functions are considered, and those that are not, but it specifies how likely any individual function is before having observed any datapoints. For example, a kernel that prefers "smooth functions" would find the function illustrated in Figure 3a to be much less probable than the function illustrated in Figure 3b.



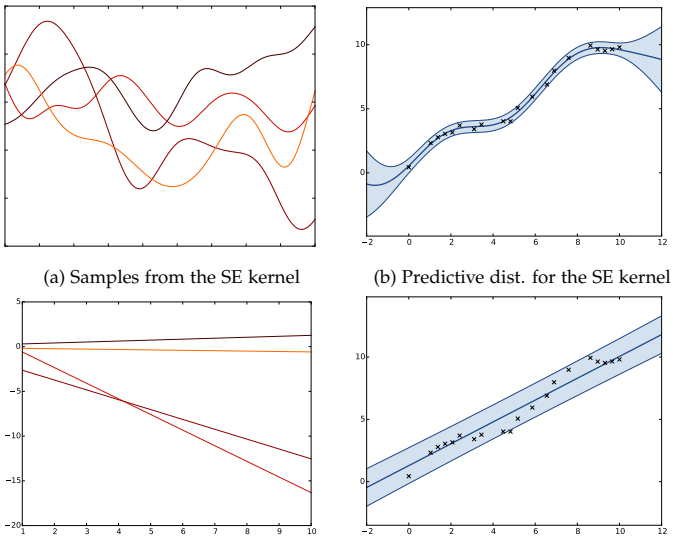
(a) A function with low prior probability (b) A function with high prior probability

Figure 3: A likely and unlikely function based on a kernel that prefers smooth functions

There are two ways to gain insight into the distributions defined by a kernel. One is to look at samples drawn from the Gaussian process with that kernel — essentially giving us an idea of what the "typical" function looks like, and the other is to look at the predictive distribution given an example set of data. The prior samples shown in Figure 4a and Figure 4c shows that the Squared Exponential (or SE) kernel is representative of smooth functions, while the linear kernel represents linear functions. The predictive distributions illustrated in Figure 4b and Figure 4d show that the choice of kernel has a huge influence on the resulting predictions. This is why the selection of an appropriate kernel is such an important task.

3.4 NUMERICAL IMPLEMENTATION

For many numerical implementations of probabilistic models, the calculations are not performed by directly implementing the formula used to define the model. This is due to the limitations of computing hardware and the requirement of fixed precision or floating point representations of numbers. Since computers can only



(a) Samples from the SE kernel

(b) Predictive dist. for the SE kernel

(c) Samples from the linear kernel

(d) Predictive dist. for the linear kernel

Figure 4: Samples from the prior and the predictive distribution of the SE and Linear kernels. The prior samples show functions that are likely under the prior and the predictive distribution is plotted as the posterior mean and a 95% confidence interval.

store a finite representation of any given number, rounding errors are introduced through different calculations. Algorithms which are not sensitive to these errors are known as numerically stable, while those that can be affected by these are numerically unstable.

Matrix inversion, an operation that is central to the definition of GPs is a numerically unstable procedure. While there are algorithms to calculate the inverse, for certain matrices the round-off errors in the calculation can propagate to cause very large errors in the resulting inverse. Fortunately, there are very few occasions where the end result of a calculation is the matrix inverse. Far more commonly the desired quantity is one of the form $\mathbf{K}^{-1}\vec{v}$. In the case of GPs, these quantities are $\mathbf{K}^{-1}\vec{y}$ (in Equation 13 and Equation 14) and $\mathbf{K}^{-1}\mathbf{K}_{x*}$ (in Equation 13). The problem $\vec{x} = \mathbf{K}^{-1}\vec{v}$, often written $\mathbf{K}\vec{x} = \vec{v}$, is known as a system of linear equations and there are numerically stable methods for solving these.

The default methods for solving linear systems of equations, for example **linsolve** or **A\b** (when A is not sparse) in Matlab or **numpy.linalg.solve** in Python, use a *factorization method* to perform this calculation. These techniques are much more numerically stable than explicitly inverting a matrix and come at the same computational cost. Standard implementations of Gaussian processes also rely

on factorization to calculate the other required terms in Equations 13-15, such as the log-determinant, in a numerically stable way.

3.4.1 *Factorization methods*

Factorization methods decompose a matrix $\mathbf{K}$ into a product of simpler matrices such that quantities of interest can be calculated in a much more efficient and stable manner. There are two factorization methods that we will use throughout this thesis, the Cholesky decomposition and the singular value decomposition (SVD).

The Cholesky decomposition

Figure 5: Illustration of the sparsity pattern of a Cholesky decomposition

The canonical decomposition for PSD matrices, and the most commonly used in solving Gaussian processes is the Cholesky decomposition. The Cholesky decomposition decomposes a PSD matrix into the following form:

$$\mathbf{K} = \mathbf{L}\mathbf{L}^{\intercal} \tag{16}$$

The Cholesky factor $\mathbf{L}$ is a lower triangular matrix, meaning that all elements above the leading diagonal are 0. This sparsity pattern is shown in Figure 5. This structure allows linear systems to be solved in $O(N^2)$ operations through the backsubstitution algorithm [Trefethen and Bau, 1997].

In the context of kernel methods, we can interpret the Cholesky factor by noting the equivalence between Equation 16 and Equation 5. The Cholesky factor defines a feature space $\mathbf{\Phi} = \mathbf{L}$ that is consistent with $\mathbf{K}$, such that the matrix $\mathbf{L}$ is lower triangular.

The quantities in a Gaussian process can be stably calculated with the formulas listed in Table 1.

Quantity	Formula	Computational Cost
$\log(\det(\mathbf{K}))$	$2\sum_i^N \log \mathbf{L}_{ii}$	$O(N)$
$\mathbf{K}^{-1}\vec{v}$	$\mathbf{L}^{-\intercal}\left(\mathbf{L}^{-1}\vec{v}\right)$	$O(N^2)$
$Tr\left[\mathbf{K}^{-1}\frac{d\mathbf{K}}{d\theta}\right]$	$\sum_i^N \log \mathbf{L}_i^{-T}\left(\mathbf{L}^{-1}\frac{d\mathbf{K}_i}{d\theta}\right)$	$O(N^3)$

Table 1: Formulas for numerically stable calculation of quantities in GPs

The singular value decomposition

The most common cannonical matrix decomposition is the Singular Value Decomposition. This procedure converts any matrix $\mathbf{X}$ into the form $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal$, where $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices and $\mathbf{\Sigma}$ is a diagonal matrix. In the case of symmetric matrices, such as kernel matrices, $\mathbf{U} = \mathbf{V}$, so $\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\intercal$. The diagonal entries of $\mathbf{\Sigma}$ are called the *singular values* (which for PSD matrices are equivalent to the eigenvalues) of a matrix, and each has a corresponding *singular vector* which is a column of $\mathbf{U}$. These vectors define an orthogonal basis for $\mathbf{X}$, with the length of each component being the corresponding singular value.

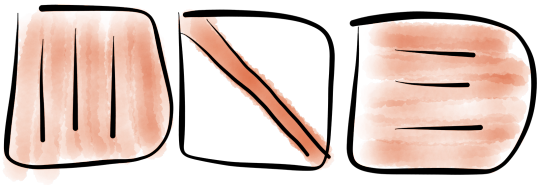


Figure 6: Illustration of the sparsity pattern of a singular value decomposition

Unlike the Cholesky, it is possible to calculate a k-partial SVD, which is the first k singular values and vectors of a matrix. This can be used to construct a low rank approximation to the original matrix $\widetilde{\mathbf{K}} = \mathbf{U}_{1:k}\mathbf{\Sigma}_{1:k,1:k}\mathbf{U}_{1:k}^\intercal$. Under a number of matrix norms[2], this is the optimal rank k decomposition. The inverse of this decomposition is easy to compute, as for a singular matrix $\mathbf{U}^{-1} = \mathbf{U}^\intercal$ and the inverse of a diagonal matrix is simply the elementwise diagonal. So $\mathbf{K}^{-1} = \mathbf{U}^\intercal\mathbf{\Sigma}^{-1}\mathbf{U}$.

If we perform SVD on a kernel matrix, we have a different primal representation of our kernel than with the Cholesky. Because the kernel matrix is positive semi-definite we know that the singular values will be all positive. Thus we can safely take the square root of $\mathbf{\Sigma}$, which is the diagonal matrix of square roots of singular values.

$$\mathbf{K} = \mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\mathbf{\Sigma}^{\frac{1}{2}}\mathbf{U}^\intercal = \left(\mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\right)\left(\mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\right)^\intercal$$

We can interpret this as a feature space $\mathbf{\Phi} = \left(\mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\right)$, which is a set of orthogonal features of descending "importance".

In the next section, we will introduce a framework for solving GPs that does not require the full factorization of the kernel matrix.

2 Frobenius and 2-norm

4

SOLVING GPS WITH ITERATIVE METHODS : $O(N^3) \rightarrow O(N^2 I)$

In karate there is an image that is used to define . . . "mind like water." Imagine throwing a pebble into a still pond. How does the water respond? The answer is, totally appropriately to the force and mass of the input; then it returns to calm. It doesn't overreact or underreact . . .

— David Allen, Getting things done

An iterative framework for solving Gaussian Processes was first proposed in [Gibbs, 1997] and [Gibbs and MacKay, 1997], based on the work of [Skilling, 1993]. This framework demonstrated that the relevant quantities in GP inference could be calculated via iterative methods — methods that continually refine estimates of the desired quantities over multiple iterations.

In this section we will take this framework and extend it in a number of ways. Firstly, we provide specific termination criteria for the iterative methods and bounds on the prediction error that results from their use. We introduce and evaluate different estimators for certain quantities, showing substantial improvement over the originals. We extend the framework to include an approximation of the log-likelihood, which allows for modelling scoring and comparison. Finally we show how including pre-conditioners within a framework for solving GPs ties together previously disparate approximation methods for GPs.

4.1 INTRODUCTION TO ITERATIVE GPS

To introduce iterative methods for solving GPs, we will return to the two observations about the full factorization method for solving GPs from Section 2.2. The first is that all quantities are learnt to machine precision. The second is that for a given training set size the cost of inference is constant, regardless of the complexity of the function being learnt. We will address each of them in turn, showing how these can be limitations, and how these limitations are avoided in iterative GP methods.

Firstly, we address the constant cost of inference. Factorization methods for solving Gaussian Processes always require the same amount of computation for a given dataset size N, regardless of the actual dataset or kernel. This means inference is independent of whether the function being learnt is "complicated" or not. It requires $O(N^3)$ operations, regardless of whether the function is an intricate, subtly varying function or a simple curve. We won't attempt to define

yet what a complicated function is, it is sufficient for motivation that we have an intuition that some functions are harder to solve than others. Figure 7 shows two regression problems, each with N=40.

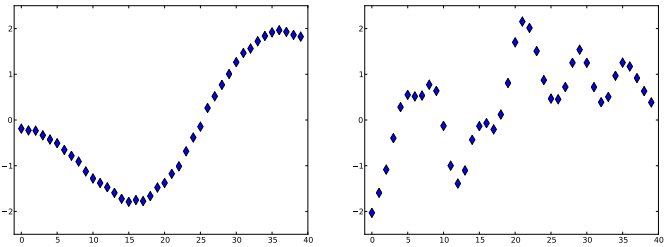


Figure 7: Not all functions are created equal

Now the problem here is clear. If we believe that there are some regression problems that are genuinely hard, and require a large amount of computation to solve, and all solutions to problems of size N require the same amount of computation, then all problems of size N require the same amount of computation as *the hardest possible problem of that size*. This means the time required to solve a simple line in one dimension is always tied to be the same as the most complicated high dimensional function. In the case of iterative methods, this is not true. Some problems, that we will term "harder", will require a larger number of iterations than those that are "easier". Iterative methods allow us to only exercise the computational resources necessary to solve the problem we actually have, not the most difficult in it's class. Moreover, we will see that the hardest functions in the class will only require the same order of computation as the full factorization method, paying at most a 3x penalty, while the easiest functions can be solved a full order of N faster.

Secondly, the fact that full factorization calculates quantities of interest to machine precision is a further limitation. While the gains are entirely dependent on the problem in question, the potential computational savings for a small concession in accuracy is well illustrated by the Travelling Salesman Problem. The Traveling Salesman problem is the most well-known NP-hard problem, and an exact solution requires $O(N!)$ operations. This means that exact solutions can be found in reasonable time for only very small datasets. However, very large TSP problems, such as the World TSP problem[Cook], have been calculated by allowing a small error of .05%. This problem, which is made up of every populated city in the world, has over 2 million points — a size for which exact inference would take longer than the life of the universe. The approximate solution is shown in Figure 8. While the computational savings are not this extreme in Gaussian Processes, there are still benefits to solving quantities only to the level of accuracy that we actually require. Iterative methods can take advantage of these savings by providing a means to calculate solutions to a given error tolerance.

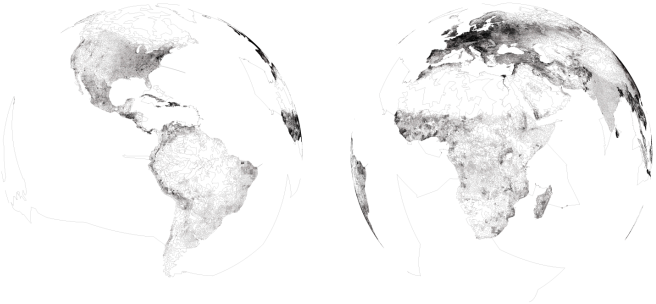


Figure 8: Rendering of the Travelling Salesman path for all populated cities in the world

4.2 BACKGROUND TO ITERATIVE METHODS

This iterative GP framework is built upon iterative methods for calculating two fundamental matrix quantities: solutions to linear systems and matrix traces. Solutions to linear systems are necessary for all the quantities of interest in GPs (Eq 13 - 15) and the matrix trace appears in the log-likelihood and its derivatives (Eq 14 - 15). In this section we will review iterative methods for calculating these quantities.

4.2.1 *Iterative linear systems solvers*

The key equivalence that allows for the iterative solution of systems of linear equations is between such a linear system, notated by finding x in either:

$$\mathbf{K}\vec{x} = \vec{y} \text{ or } \vec{x} = \mathbf{K}^{-1}\vec{y} \tag{17}$$

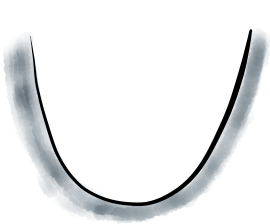
and that of finding the minimum point of a particular quadratic form:

$$\min_x f(x) = \min_x \frac{1}{2}\vec{x}^\intercal \mathbf{K}\vec{x} - \vec{x}^\intercal\vec{y} \tag{18}$$

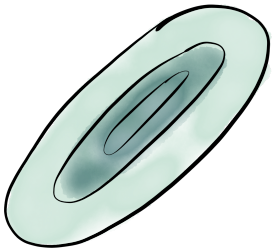
This framework only requires solving linear systems with the Gram matrix, so $\mathbf{K}$ will be PSD. When $\mathbf{K}$ is PSD, this multivariate quadratic form is the multi-dimensional generalization of the 1-D quadratic, illustrated in Figure 9a. It can be validly imagined as a bowl in N dimensions; the 2 dimensional version is illustrated in Figure 9b.

Lemma 4.2.1. *When $\mathbf{K}$ is symmetric and positive semi-definite, the minima of f, the center of the bowl, is at the solution to $\mathbf{K}\vec{x} = \vec{y}$.*

Proof. We can verify that $\vec{x} = \mathbf{K}^{-1}\vec{y}$ is indeed the minimum point of Equation 18 by simple substitution. Since the quadratic function is convex, we only need to check that the derivative is 0.



(a) Illustration of 1D quadratic



(b) Contour illustration of 2D quadratic

$$\frac{d}{d\vec{x}} f(x) = \mathbf{K}\vec{x} - \vec{y} \tag{19}$$

$$\frac{d}{d\vec{x}} f(\mathbf{K}^{-1}\vec{y}) = \vec{0} \tag{20}$$

□

This equivalence provides a link between an algebraic problem and an optimization problem. By considering the problem in the frame of optimization, it is much more obvious how to calculate solutions in an iterative manner. This optimization task can be solved via any gradient based method. There are two particular gradient methods that we will review that take advantage of the quadratic structure of the optimization problem. The first is a class of methods known as conjugate directions. These methods exploit the structure to ensure convergence within N iterations for an $N \times N$ linear system. The second is conjugate gradient, which is the most well-known and widely used method of conjugate directions.

Conjugate directions

Imagine that, while optimizing an N-dimensional quadratic, you are given a set of n linearly independent vectors $\{\vec{d}_0, ..., \vec{d}_n\}$ which we will call "search directions". These are the directions along which you are allowed to move while optimizing the function. Since they are linearly independent, they form a basis for the space $\mathbb{R}^n$ and the vector required to move from the initial point $\vec{v}_0$ to the optimal point $\vec{v}_{opt}$ can be described as a linear combination of these vectors:

$$\vec{e} = \vec{v}_{opt} - \vec{v}_0 = \alpha_1 \vec{d}_1 + ... + \alpha_n \vec{d}_n \tag{21}$$

We use $\vec{e}$ to represent this vector, as we can think of it as the error vector between the initial solution ($\vec{v}_0$) and the correct solution ($\vec{v}_{opt}$). If $\vec{\alpha}$ can be calculated in closed form, then we can take the correct step length in each of these directions, assuring that we will have converged after taking the correct step in each of the n directions. The two steps for a set of search directions on the 2d quadratic is shown in Figure 10.

The method of conjugate directions calculates a set of search directions for which $\vec{\alpha}$ can be calculated in closed form. At every iteration the algorithm takes

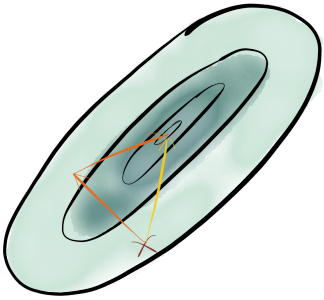


Figure 10: The steps taken (in orange) by conjugate directions for a certain choice of search directions from a given starting point (the red cross). The error vector is shown in yellow.

this optimal step in the direction of one of the $\vec{d}$s and does not have to worry about that direction again.

The property that the vectors must satisfy to qualify as valid search direction is that they are **K**-*conjugate* to each other. Two vectors $\vec{d_i}$ and $\vec{d_j}$ (for $i \neq j$) are **K**-conjugate if:

$$\vec{d_i}^{\intercal} \mathbf{K} \vec{d_j} = 0 \tag{22}$$

Where **K** is the matrix that defines the quadratic, as seen in Equation 18. In order to construct a **K**-conjugate set of search directions from an initial set of basis vectors, we can use the technique of Gram-Schmidt conjugation. Gram-Schmidt conjugation takes a set of linearly independent vectors $\{\vec{u}_1, ..., \vec{u}_n\}$ and iteratively constructs a **K**-conjugate set by setting $\vec{d_i}$ to be $\vec{u}_i$ and then removing all the components of $\vec{d_i}$ that are not **K**-orthogonal to the previous $i-1$ directions.

$$\vec{d_i} = \vec{u}_i - \sum_{j=0}^{i-1} \beta_{ij} \vec{d_j} \tag{23}$$

Since the previous $i-1$ vectors will be **K**-conjugate, the βs can be described in a simple closed form, for $k \in \{1, \ldots, i-1\}$:

$$\vec{d_k}^{\intercal} \mathbf{K} \vec{d_i} = \vec{d_k}^{\intercal} \mathbf{K} \vec{u}_i - \sum_{j=0}^{i-1} \beta_{ij} \vec{d_k}^{\intercal} \mathbf{K} \vec{d_j} \tag{24}$$

$$0 = \vec{d_k}^{\intercal} \mathbf{K} \vec{u}_i - \beta_{ik} \vec{d_k}^{\intercal} \mathbf{K} \vec{d_k} \tag{25}$$

$$\beta_{ik} = \frac{\vec{d_k}^{\intercal} \mathbf{K} \vec{u}_i}{\vec{d_k}^{\intercal} \mathbf{K} \vec{d_k}} \tag{26}$$

In a similar fashion, we can derive the closed form solution for α_i by first looking at the **K**-inner product of the search direction $\vec{d_i}$ with the error vector.

$$\vec{d}_i^{\intercal} \mathbf{K} \vec{e} = \sum_{j=0}^{i} \alpha_j \vec{d}_i^{\intercal} \mathbf{K} \vec{d}_j \tag{27}$$

$$= \alpha_i \vec{d}_i^{\intercal} \mathbf{K} \vec{d}_i \tag{28}$$

Due to **K**-conjugacy, all terms in the summation except for the **K** norm of $\vec{d}_i$ are 0.

$$\alpha_i = \frac{\vec{d}_i^{\intercal} \mathbf{K} \vec{e}}{\vec{d}_i^{\intercal} \mathbf{K} \vec{d}_i} \tag{29}$$

$$= \frac{\vec{d}_i^{\intercal} \mathbf{K} \left(\vec{v}_i - \vec{v}_{opt} \right)}{\vec{d}_i^{\intercal} \mathbf{K} \vec{d}_i} \tag{30}$$

$$= \frac{\vec{d}_i^{\intercal} \left(\mathbf{K} \vec{v}_i - \vec{y} \right)}{\vec{d}_i^{\intercal} \mathbf{K} \vec{d}_i} \tag{31}$$

At every iteration i we need to have stored all the previous search directions, which makes the storage complexity $O(Ni)$. This high storage cost is one of the reasons that the conjugate directions methods were not particularly popular before the advent of conjugate gradient [Shewchuk, 1994].

Conjugate gradient

As mentioned in the previous section, the chief drawback of conjugate directions is that it needs to keep around all the previous search directions in order to calculate the next. Conjugate gradient solves this problem. It is a conjugate directions methods whose set of search directions are chosen such that each search direction depends only on the previous search direction, freeing us from the requirement of keeping all the preious search vectors in memory. This reduces the storage complexity from $O(NI)$ to $O(N)$.

Firstly, this method takes advantage of the fact that the basis vectors do not need to be specified a priori; each basis vector u_i is only required at the ith iteration. At the ith iteration we take the basis vector to be the direction of steepest descent at that point of the optimization. The direction of steepest descent is given by the gradient of the function at that point $\vec{u}_i = \vec{y} - \mathbf{K}\vec{x}_i$. We will show that with this choice of basis vectors, all but one of the β_{ij}s will be 0, removing the dependence of the search directions on all but the current and previous basis vectors. This is due to the fact that these basis vectors are orthogonal to the previous search directions. So, for $i > j$

$$\vec{u}_i^\intercal \vec{d}_j = (\vec{y} - \mathbf{K}\vec{x}_i)^\intercal \vec{d}_j \tag{32}$$

$$= \left(\vec{y} - \mathbf{K}\left(\vec{x}_0 + \sum_i \alpha_i \vec{d}_i\right)\right)^\intercal \vec{d}_j \tag{33}$$

$$= (\vec{y} - \mathbf{K}\vec{x}_0)^\intercal \vec{d}_j \tag{34}$$

$$= \vec{d}_0^\intercal \vec{d}_j \tag{35}$$

$$= 0 \tag{36}$$

Another way to phrase this is that the basis vector is orthogonal to the subspace spanned by the previous search directions. Since the search directions are generated from the basis vectors, this is the same subspace as the span of the previous basis vectors. This implies that

$$\vec{u}_i^\intercal \vec{u}_j = 0 \qquad \text{(for } i \neq j\text{)} \tag{37}$$

Using this, we can derive a closed form for $\vec{\beta}$:

$$\vec{u}_{j+1} = \vec{y} - \mathbf{K}\vec{x}_{j+1} \tag{38}$$

$$\vec{u}_{j+1} = \vec{y} - \mathbf{K}\left(\vec{x}_j + \alpha_j \vec{d}_j\right) \tag{39}$$

$$\vec{u}_{j+1} = \vec{u}_j - \alpha_j \mathbf{K}\vec{d}_j \tag{40}$$

$$\vec{u}_i^\intercal \vec{u}_{j+1} = \vec{u}_i^\intercal \vec{u}_j - \alpha_j \vec{u}_i^\intercal \mathbf{K}\vec{d}_j \tag{41}$$

$$\alpha_j \vec{u}_i^\intercal \mathbf{K}\vec{d}_j = \vec{u}_i^\intercal \vec{u}_j - \vec{u}_i^\intercal \vec{u}_{j+1} \tag{42}$$

$$\vec{u}_i^\intercal \mathbf{K}\vec{d}_j = \begin{cases} \frac{1}{\alpha_i}\vec{u}_i^\intercal \vec{u}_i & \text{where } i = j \\ -\frac{1}{\alpha_{i-1}}\vec{u}_i^\intercal \vec{u}_i & \text{where } i = j+1 \\ 0 & \text{otherwise} \end{cases} \tag{43}$$

Since $\beta_{ij} = -\frac{\vec{u}_i^\intercal \mathbf{K}\vec{d}_j}{\vec{d}_j^\intercal \mathbf{K}\vec{d}_j}$, only $\beta_{i,i-1}$ are non-zero:

$$\beta_{i,i-1} = \frac{1}{\alpha_{i-1}} \frac{\vec{u}_i^\intercal \vec{u}_i}{\vec{d}_{i-1}^\intercal \mathbf{K}\vec{d}_{i-1}} \tag{44}$$

$$= \frac{\vec{u}_i^\intercal \vec{u}_i}{\vec{d}_{i-1}^\intercal \vec{u}_{i-1}} \tag{45}$$

$$= \frac{\vec{u}_i^\intercal \vec{u}_i}{\vec{u}_{i-1}^\intercal \vec{u}_{i-1}} \tag{46}$$

The full algorithm for conjugate gradient can be implemented as follows:

$$\vec{d_0} = \vec{u}_0 = \vec{y} - \mathbf{K}\vec{x}_0 \tag{47}$$

$$\alpha_i = \frac{\vec{u}_i^\intercal \vec{u}_i}{\vec{d}_i^\intercal \mathbf{K}\vec{d}_i} \tag{48}$$

$$\vec{x}_{i+1} = \vec{x}_i + \alpha_i \vec{d}_i \tag{49}$$

$$\vec{u}_{i+1} = \vec{u}_i - \alpha_i \mathbf{K}\vec{d}_i \tag{50}$$

$$\beta_{i+1} = \frac{\vec{u}_{i+1}^\intercal \vec{u}_{i+1}}{\vec{u}_i^\intercal \vec{u}_i} \tag{51}$$

$$\vec{d}_{i+1} = \vec{u}_{i+1} + \beta_{i+1}\vec{d}_i \tag{52}$$

Convergence and pre-conditioning

As pre-conditioning is too often overlooked in implementations, it is worth reflecting on this quote from the book *Numerical Linear Algebra*.

> In ending this book with the subject of preconditioners, we find ourselves at the philosophical center of the scientific computing of the future.... Nothing will be more central to computational science in the next century than the art of transforming a problem that appears intractable into another whose solution can be approximated rapidly. For Krylov subspace matrix iterations, this is preconditioning.[Trefethen and Bau, 1997]

The numerical stability and convergence rate of conjugate gradient (and other iterative methods for linear systems) are very sensitive to the condition number of the Gram matrix $\kappa(\mathbf{K})$. The condition number of a matrix is the ratio between its largest and smallest eigenvalues.

$$\kappa(\mathbf{K}) = \frac{\lambda_{max}(\mathbf{K})}{\lambda_{min}(\mathbf{K})} \tag{53}$$

A low value of the condition number will lead to a numerically stable solution in a small number of iterations, whereas a problem with a very high condition number may not converge at all. So ideally we want the eigenvalues of $\mathbf{K}$ to be as tightly clustered as possible, giving it a low value of κ and fast convergence of our algorithm.

The number of iterations required for Conjugate Gradient to achieve a given error tolerance is bounded by $\frac{1}{2}\sqrt{\kappa}\ln\left(\frac{2}{\epsilon}\right)$, where κ is the condition number of $\mathbf{K}$ and ϵ is the relative error [Shewchuk, 1994]. Thus the two factors that dictate the speed of calculating a solution are the condition number and the error tolerance. Since the number of iterations is proportional to the square root of the condition number, and the log of the error tolerance, the condition number is usually the important term.

While we cannot choose the condition number of our matrix, we can instead attempt to find a transformed linear system that has a smaller condition number.

More concretely, if we can find a matrix $\mathbf{P}$, such that $\kappa(\mathbf{P}^{-1}\mathbf{K}) \ll \kappa(\mathbf{K})$, then we can solve the linear system $\mathbf{P}^{-1}\mathbf{K}\vec{x} = \mathbf{P}^{-1}\vec{b}$ more efficiently to recover the same value of $\vec{x}$. This matrix $\mathbf{P}$, known as a *pre-conditioning* matrix, can be thought of as an "approximate inverse", that brings $\mathbf{K}$ closer to the identity matrix (which has the lowest possible condition number of 1 and would require only one conjugate gradient iteration to solve).

The preconditioner can be visualized as follows: in the context of optimizing a quadratic form, the singular values of the matrix correspond to an orthogonal basis of the quadratic bowl. If all the singular values are equal, the bowl is perfectly spherical. If the singular values are spread out, the bowl is very narrow in some dimensions and very broad in others. So geometrically, the preconditioner attempts to stretch the quadratic bowl such that it is as spherical as possible, as shown in Figure 11.

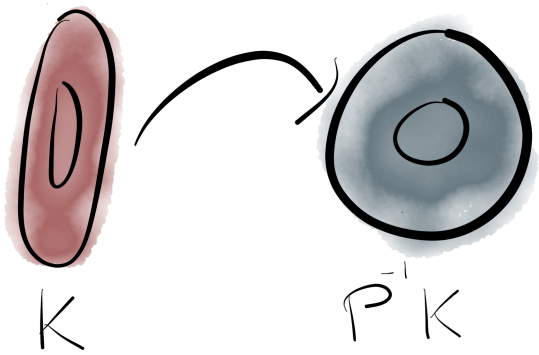


Figure 11: Original and transformed problem

The choice of a pre-conditioner is problem specific, but is guided by the following considerations:

1. It must be computationally efficient to calculate $\mathbf{P}^{-1}\vec{v}$
2. $\mathbf{P}^{-1}$ should be in some sense close to $\mathbf{K}^{-1}$

At one end of the spectrum of choices is the identity matrix $\mathbf{P} = \mathbf{I}$. It adds no computational overhead as $\mathbf{I}^{-1}\vec{v} = \vec{v}$, however it also does not affect the condition number of the problem. Alternatively we could set $\mathbf{P} = \mathbf{K}$. This would reduce the condition number of the problem to 1, however solving $\mathbf{K}^{-1}\vec{v}$ is exactly the original problem. There are many different practical choices that have been developed that make different trade-offs between computational complexity and reduction in condition number, some of which we will use later in this thesis.

4.2.2 *Stochastic trace estimators*

The matrix trace — the sum of the diagonal elements of a matrix — is trivial to calculate if we have an explicit representation of our matrix. However, we will

require the ability to calculate the trace of matrices for which we do not have an explicit representation $\mathbf{K}$, but for which we can calculate $\mathbf{K}\vec{v}$ for an arbitrary vector $\vec{v}$.

There are four estimators that we will consider:

$$\widetilde{Tr}_{Gaussian}(\mathbf{K}) = \frac{1}{m}\sum_{i=1}^{m} \vec{v}_i^{\intercal}\mathbf{K}\vec{v}_i \qquad \text{where } \vec{v}_i \sim \mathcal{N}(\vec{0}, \mathbf{I}) \tag{54}$$

$$\widetilde{Tr}_{Rayleigh}(\mathbf{K}) = \frac{1}{m}\sum_{i=1}^{m} \frac{\vec{v}_i^{\intercal}\mathbf{K}\vec{v}_i}{\vec{v}_i^{\intercal}\vec{v}_i} \qquad \text{where } \vec{v}_i \sim \mathcal{N}(\vec{0}, \mathbf{I}) \tag{55}$$

$$\widetilde{Tr}_{Hutchinson}(\mathbf{K}) = \frac{1}{m}\sum_{i=1}^{m} \vec{v}_i^{\intercal}\mathbf{K}\vec{v}_i \qquad \text{where } \vec{v}_{ij} \sim Bernoulli(.5) \tag{56}$$

$$\widetilde{Tr}_{Unit}(\mathbf{K}) = \frac{N}{m}\sum_{i=1}^{m} \mathbf{I}_{r_i}^{\intercal}\mathbf{K}\mathbf{I}_{r_i} \qquad \text{where } \vec{r}_i \sim Categorical(N) \tag{57}$$

In Equation 56, the Bernoulli distribution has outcomes of $\{+1, -1\}$, and in Equation 57, $\mathbf{I}_{r_i}$ refers to the r_ith column of the identity matrix.

General bounds

General bounds for each method are derived in [Roosta-Khorasani and Ascher, 2013] and [Avron and Toledo, 2011] that give the number of random vectors required to satisfy the following inequality:

$$Pr\left(\left|Tr(\mathbf{K}) - \widetilde{Tr}(\mathbf{K})\right| \leqslant \epsilon Tr(\mathbf{K})\right) \geqslant 1 - \delta \tag{58}$$

That is, it provides a formula for the number of samples N, required to achieve a desired degree of relative accuracy ϵ with probability $1 - \delta$. These bounds, as well as the variance of the result from a single sample vector are lised in Table 2.

Vector	(ϵ,δ)-Bound	Variance
Hutchinson	$6c(\epsilon,\delta)$	$2\left(\|\mathbf{K}\|_{Fr}^2 - \sum_{i=1}^{N}\mathbf{K}_{ii}^2\right)$
Gaussian	$8c(\epsilon,\delta)$	$2\|\mathbf{K}\|_{Fr}^2$
Rayleigh	$\frac{1}{2}\left(\frac{rank(\mathbf{K})\kappa(\mathbf{K})}{N}\right)^2 c(\epsilon,\delta)$	$N\left(\sum_{i=1}^{N}\lambda_i(\mathbf{K})^2 - \left(\sum_{i=1}^{N}\lambda_i(\mathbf{K})\right)^2\right)$
Unit	$\frac{1}{2}\left(\frac{N\max_i \mathbf{K}_{ii}}{Tr(\mathbf{K})}\right)^2 c(\epsilon,\delta)$	$N\sum_{i=1}^{N}\mathbf{K}_{ii}^2 - Tr(\mathbf{K})^2$

Table 2: Bounds and variance for the three trace estimators

Where:

$$c(\epsilon,\delta) = \epsilon^{-2}\ln\left(\frac{2}{\delta}\right) \tag{59}$$

As these are bounds that hold for all matrices, they are not necessarily tight for the matrices that we will apply them to. Any given kernel will only generate a subset of PSD matrices, which is itself a subset of all matrices.

However, these bounds do draw a picture of the quantities to which the different estimators may be sensitive. From these, as well as experiments in [Roosta-Khorasani and Ascher, 2013] [Avron and Toledo, 2011], it is show a number of properties of these estimators. Both the Gaussian and Rayleigh are strongly affected by the condition number and clustering of eigenvalues, the Rayleigh estimator can effectively estimate low-rank matrices, whereas Gaussian is heavily penalized in this regime. The Unit vector is penalized in cases where the diagonal of the kernel matrix is highly varied and there is also evidence that the Hutchinson outperforms the others in some sparse matrix settings.

4.3 POSTERIOR MEAN

As we saw in Section 3.3, the equation for the posterior mean of a GP is:

$$\vec{\mu} = \mathbf{K}_{*x}\mathbf{K}^{-1}\vec{y} \tag{60}$$

At training time we have access only to $\mathbf{K}$ and $\vec{y}$, since we don't know the location of the test points. Therefore the quantity that we focus on calculating is:

$$\vec{\alpha}_{opt} = \mathbf{K}^{-1}\vec{y} \tag{61}$$

As reviewed in Section 4.2.1, solutions to problems of the form $\mathbf{K}^{-1}\vec{\alpha}$ can be solved iteratively using conjugate gradient. Like any optimization algorithm, we do not need to run it to full convergence, we can instead perform iterations until a convergence criterion is reached. The only remaining question is how to set the convergence criterion. Setting the tolerance too large will result in a poor solution, while setting it too low will result in unnecessary computation.

In order to derive a meaningful tolerance bound, we first look at bounds on the objective function from Eq 18. Throughout the execution of conjugate gradient, we can calculate converging upper and lower bounds on the value of the quadratic objective at the optimal point $\vec{\alpha}_{opt}$. Since we know the optimum occurs at $\vec{\alpha}_{opt} = \mathbf{K}^{-1}\vec{y}$, by substitution we have:

$$f_{opt} = -\frac{1}{2}\vec{y}^{\intercal}\mathbf{K}^{-1}\vec{y} = -\frac{1}{2}\vec{\alpha}_{opt}^{\intercal}\mathbf{K}\vec{\alpha}_{opt} \tag{62}$$

The upper bound requires that $\mathbf{K}$ is of the form $\mathbf{K}_{-} + \sigma\mathbf{I}$, where $\mathbf{K}_{-}$ is also PSD, but this is always the case in our construction of GPs (Equation 12). The bounds at iteration i, derived in [Gibbs and MacKay, 1997][1], are as follows:

1 In their work, the lower bound is calculated using this formula, but using a different set of vectors $\vec{\alpha}_i'$ that are derived from a related conjugate gradient system. This bound results in a monotonically decreasing lower bound at roughly the same computational cost, but increased implementation complexity. In our experiments, simply taking the running minimum of the lower bounds using the $\vec{\alpha}$s from the solve that we describe was found to be simpler and as efficient.

$$U_i = \frac{1}{2}\vec{\alpha}_i^\intercal \mathbf{K}\vec{\alpha}_i - \vec{\alpha}_i^\intercal \vec{y} \tag{63}$$

$$L_i = \frac{1}{2\sigma^2}\left(\vec{y}^\intercal\vec{y} - 2\vec{\alpha}_i^\intercal \mathbf{K}_{-}\vec{y} + \vec{\alpha}_i^\intercal \mathbf{K}_{-}\mathbf{K}\vec{\alpha}_i\right) \tag{64}$$

Below, we prove these bounds and that they converge to the correct solution throughout the course of conjugate gradient.

Lemma 4.3.1. *U_i is an upper bound on f_{opt}*

Proof. At any iteration of conjugate gradient i, we can describe our current solution vector $\vec{\alpha}_i$ in terms of the optimal point and an error term: $\vec{\alpha}_i = \vec{\alpha}_{opt} + \vec{\epsilon}_i$

$$U_i = \frac{1}{2}\left(\vec{\alpha}_{opt} + \vec{\epsilon}_i\right)^\intercal \mathbf{K}\left(\vec{\alpha}_{opt} + \vec{\epsilon}_i\right) - \left(\vec{\alpha}_{opt} + \vec{\epsilon}_i\right)^\intercal \vec{y} \tag{65}$$

$$= \frac{1}{2}\vec{\alpha}_{opt}^\intercal \mathbf{K}\vec{\alpha}_{opt} - \vec{\alpha}_{opt}^\intercal\vec{y} + \vec{\alpha}_{opt}^\intercal \mathbf{K}\vec{\epsilon}_i + \frac{1}{2}\vec{\epsilon}_i^\intercal \mathbf{K}\vec{\epsilon}_i - \vec{\epsilon}_i^\intercal\vec{y} \tag{66}$$

$$= \frac{1}{2}\vec{y}^\intercal \mathbf{K}^{-1}\vec{y} - \vec{y}^\intercal \mathbf{K}^{-1}\vec{y} + \vec{\epsilon}_i^\intercal\vec{y} + \vec{\epsilon}_i^\intercal \mathbf{K}\vec{\epsilon}_i - \vec{\epsilon}_i^\intercal\vec{y} \tag{67}$$

$$= -\frac{1}{2}\vec{y}^\intercal \mathbf{K}^{-1}\vec{y} + \vec{\epsilon}_i^\intercal \mathbf{K}\vec{\epsilon}_i \tag{68}$$

Since $\mathbf{K}$ is PSD and $\vec{\epsilon}_i\mathbf{K}\vec{\epsilon}_i$ converges monotonically to $\vec{0}$ throughout conjugate gradient[Shewchuk, 1994], U_i is a montonically converging upper bound on f_{opt}. □

Lemma 4.3.2. *L_i is a lower bound on f_{opt}*

Proof. We repeat the same procedure as for the upper bound, replacing $\vec{\alpha}_i$ with $\vec{\alpha}_{opt} + \vec{\epsilon}_i$.

$$L_i = -\frac{1}{2\sigma^2}\left(\vec{y}^\intercal\vec{y} - 2\left(\vec{\alpha}_{opt} + \vec{\epsilon}_i\right)^\intercal \mathbf{K}_{-}\vec{y} + \left(\vec{\alpha}_{opt} + \vec{\epsilon}_i\right)^\intercal \mathbf{K}_{-}\mathbf{K}\left(\vec{\alpha}_{opt} + \vec{\epsilon}_i\right)\right) \tag{69}$$

$$= -\frac{1}{2\sigma^2}\left(\vec{y}^\intercal\vec{y} - 2\vec{\alpha}_{opt}^\intercal \mathbf{K}_{-}\vec{y} - 2\vec{\epsilon}_i^\intercal \mathbf{K}_{-}\vec{y} + \vec{\alpha}_{opt}^\intercal \mathbf{K}_{-}\mathbf{K}\vec{\alpha}_{opt} + 2\vec{\epsilon}_i^\intercal \mathbf{K}_{-}\mathbf{K}\vec{\alpha}_{opt} + \vec{\epsilon}_i^\intercal \mathbf{K}_{-}\mathbf{K}\vec{\epsilon}_i\right) \tag{70}$$

$$= -\frac{1}{2\sigma^2}\left(\vec{y}^\intercal\vec{y} - \vec{y}^\intercal \mathbf{K}^{-1}\left(\mathbf{K} - \sigma^2\mathbf{I}\right)\vec{y} + \vec{\epsilon}_i^\intercal \mathbf{K}_{-}\mathbf{K}\vec{\epsilon}_i\right) \tag{71}$$

$$= -\frac{1}{2}\vec{y}^\intercal \mathbf{K}^{-1}\vec{y} - \frac{1}{2\sigma^2}\vec{\epsilon}_i^\intercal \mathbf{K}_{-}\mathbf{K}\vec{\epsilon}_i \tag{72}$$

Since $\mathbf{K}_{-}\mathbf{K}$ is PSD (the product of two PSD matrices) and $\vec{\epsilon}_i$ converges to $\vec{0}$, L_i is a lower bound on f_{opt}. □

We can use the upper and lower bounds to bound the overall approximation error of f_{opt}:

$$B_i = U_i - L_i \tag{73}$$
$$= \frac{1}{2}\vec{\alpha}_i^\intercal \mathbf{K}\vec{\alpha}_i - \vec{\alpha}_i^\intercal \vec{y} + \frac{1}{\sigma^2}\left(\frac{1}{2}\vec{y}^\intercal\vec{y} + \frac{1}{2}\vec{\alpha}_i^\intercal \mathbf{K}\mathbf{K}_{-}\vec{\alpha}_i - \vec{y}^\intercal \mathbf{K}_{-}\vec{\alpha}_i\right) \tag{74}$$
$$= \frac{1}{2\sigma^2}\left(\vec{\alpha}_i^\intercal \mathbf{K}\vec{\alpha}_i - 2\vec{\alpha}_i^\intercal\vec{y} + \vec{y}^\intercal\vec{y} + \vec{\alpha}_i^\intercal \mathbf{K}\mathbf{K}_{-}\vec{\alpha}_i - 2\vec{y}^\intercal \mathbf{K}_{-}\vec{\alpha}_i\right) \tag{75}$$
$$= \frac{1}{2\sigma^2}\left(\vec{y}^\intercal\vec{y} + \vec{\alpha}_i^\intercal \mathbf{K}\mathbf{K}\vec{\alpha}_i - 2\vec{y}^\intercal \mathbf{K}\vec{\alpha}_i\right) \tag{76}$$
$$= \frac{1}{2\sigma^2}\left|\vec{y} - \mathbf{K}\vec{\alpha}_i\right|^2 \tag{77}$$

Gibbs and Mackay[Gibbs and MacKay, 1997] identified Equation 77 as a useful guide to the accuracy of the approximation of $\vec{\alpha}_i$. However, they do not give more proscriptive advice on how to set the convergence criteria, and the quantity itself is not necessarily intuitive to set directly. A criteria that we can sensibly set is a bound on the resulting approximation error in our predictions:

$$\epsilon_y^2 = \left\|\mathbf{K}_{*x}\vec{\alpha}_i - \mathbf{K}_{*x}\vec{\alpha}_{opt}\right\|^2 \tag{78}$$
$$= \left\|\mathbf{K}_{*x}\left(\vec{\alpha}_i - \vec{\alpha}_{opt}\right)\right\|^2 \tag{79}$$
$$= \left(\vec{\alpha}_i - \vec{\alpha}_{opt}\right)^\intercal \mathbf{K}_{x*}\mathbf{K}_{*x}\left(\vec{\alpha}_i - \vec{\alpha}_{opt}\right) \tag{80}$$
$$= \left(\vec{\alpha}_i - \vec{\alpha}_{opt}\right)^\intercal \mathbf{\Phi}_x\mathbf{\Phi}_*^\intercal\mathbf{\Phi}_*\mathbf{\Phi}_x^\intercal\left(\vec{\alpha}_i - \vec{\alpha}_{opt}\right) \tag{81}$$

Since $\mathbf{\Phi}_*^\intercal\mathbf{\Phi}_*$ is a rank 1 matrix, its largest (and only) eigenvalue is $\mathbf{\Phi}_*\mathbf{\Phi}_*^\intercal$. Combined with the identity $\vec{v}^\intercal\mathbf{X}\vec{v} \leqslant \lambda_{max}(\mathbf{X})\vec{v}^\intercal\vec{v}$:

$$\leqslant \mathbf{\Phi}_*\mathbf{\Phi}_*^\intercal\left(\vec{\alpha}_i - \vec{\alpha}_{opt}\right)^\intercal\mathbf{\Phi}_x\mathbf{\Phi}_x^\intercal\left(\vec{\alpha}_i - \vec{\alpha}_{opt}\right) \tag{82}$$
$$= \mathbf{K}_*\left(\vec{\alpha} - \vec{\alpha}_{opt}\right)^\intercal\mathbf{K}\left(\vec{\alpha}_i - \vec{\alpha}_{opt}\right) \tag{83}$$

Since $\lambda_{min}\left(\frac{1}{\sigma^2}\mathbf{K}\right) \geqslant 1$

$$\leqslant \frac{1}{\sigma^2}\mathbf{K}_*\left(\vec{\alpha}_i - \vec{\alpha}_{opt}\right)^\intercal\mathbf{K}^2\left(\vec{\alpha}_i - \vec{\alpha}_{opt}\right) \tag{84}$$

and from Equation 77

$$= 2\mathbf{K}_*B_i \tag{85}$$

In practice, it is likely that this error quantity will need to be bounded in one of two ways: either in absolute terms, or relative to the underlying noise level. Firstly, in some applications, a fixed error tolerance $\epsilon_y^2 \leqslant \eta_{abs}^2$ is required, in which case we can use:

$$2\mathbf{K}_*B_i \leqslant \eta_{abs}^2 \tag{86}$$
$$B \leqslant \frac{\eta_{abs}^2}{2\mathbf{K}_*} \tag{87}$$

Where there is no fixed error bound, it is likely that the tolerance should be set as a reasonable fraction of the noise level $\epsilon_y^2 \leqslant \eta_{rel}^2 \sigma^2$. This means that in the worst case, the error in the mean is a fraction η_{rel} of the posterior variance.

$$B \leqslant \frac{\eta_{rel}^2 \sigma^2}{2\mathbf{K}_*} \tag{88}$$

An advantage of these criteria is that they are easy to implement in standard conjugate gradient solvers, whose convergence tolerance is usually with respect to the residuals $|\vec{y} - \mathbf{K}\vec{\alpha}|^2$.

$$|\vec{y} - \mathbf{K}\vec{\alpha}|^2 \leqslant \frac{\eta_{rel}^2 \sigma^2}{\mathbf{K}_*} \qquad \text{Absolute bound}$$

$$|\vec{y} - \mathbf{K}\vec{\alpha}|^2 \leqslant \frac{\eta_{rel}^2 \sigma^4}{\mathbf{K}_*} \qquad \text{Relative bound}$$

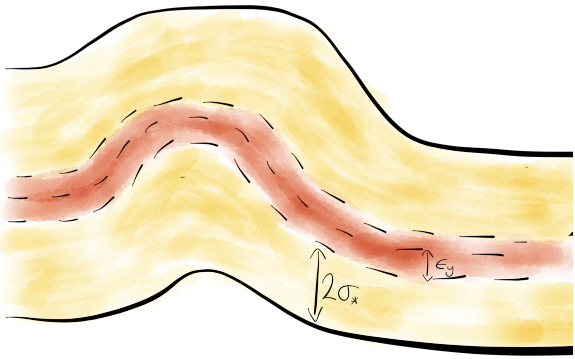


Figure 12: 95% confidence interval on a 1D function, composed of the error in prediction (red) and posterior variance (yellow)

If desired, this also allows us to fully propagate our uncertainty in the solution to our predictions: we can calculate a confidence bound on predictions by taking the posterior variance calculated confidence bound and adding the error bound term on the posterior mean.

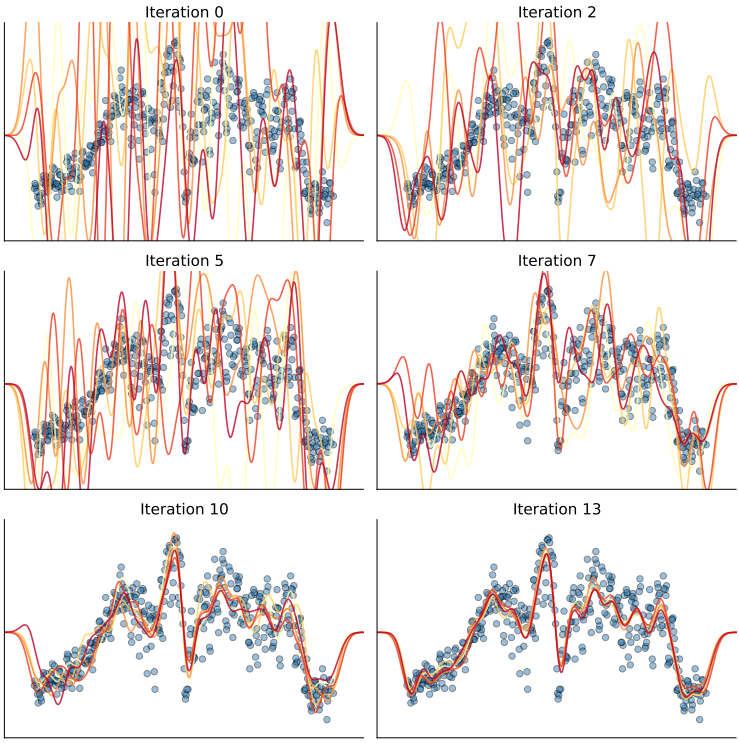
$$CI_{95} = \pm (2\sigma_* + \mathbf{K}_* B) \tag{89}$$

4.3.1 *Example datasets*

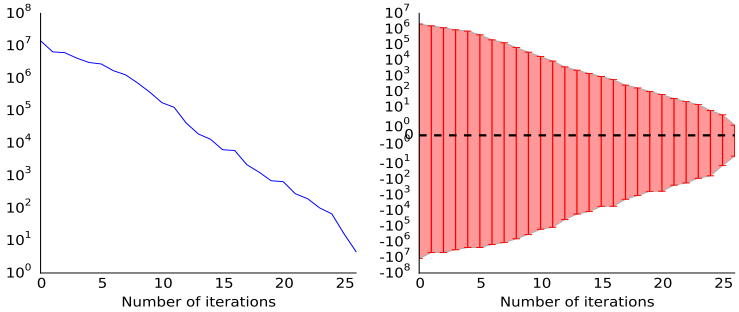
Figure 13 and Figure 14 show the execution of conjugate gradient on two datasets: the Production of H_2SO_4 and Daily min temp datasets. In Figure 13a and 14a we can see how the function converges through the course of conjugate gradient; each image shows the approximate mean at a given iteration for 5 separate runs

of conjugate gradient with a different starting vector.

Figure 13b and 14b show the behaviour of the bound at each iteration. This illustrates the exponential convergence rate of conjugate gradient, seen as a linear function on the log-scaled plot.

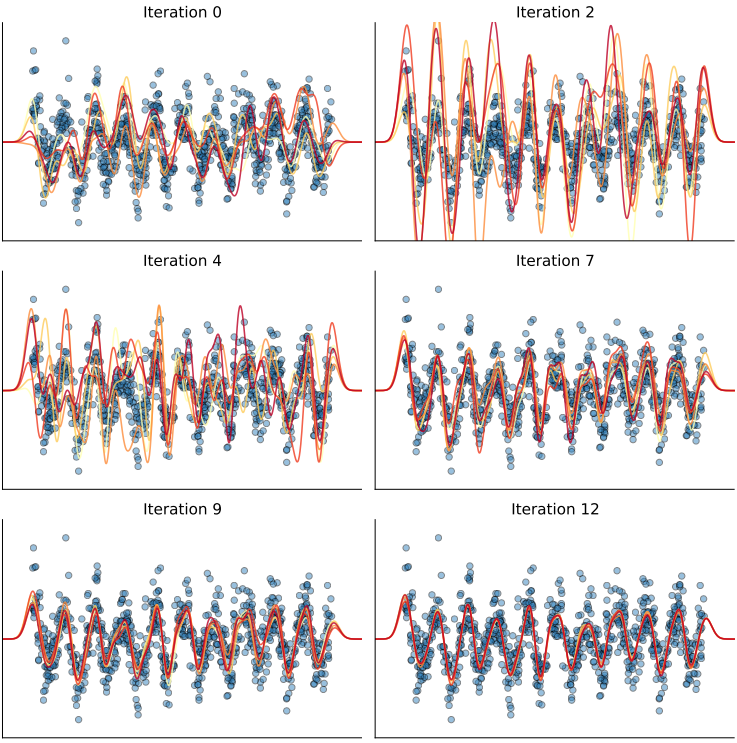


(a) Function convergence

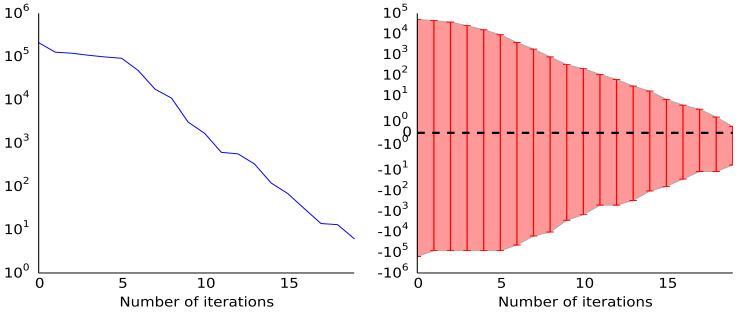


(b) Bounds throughout execution

Figure 13: Monthly production of H_2SO_4 dataset



(a) Function convergence



(b) Bounds throughout execution

Figure 14: Daily minimum temperature dataset

4.3.2 *Overall performance*

Table 3 shows the number of iterations required for convergence for a number of datasets, at the optimal hyper-parameter setting for that dataset. We can see from the table that sensibly setting the convergence tolerance on these datasets can lead to between 1x to 8x improvement in convergence rate and an overall efficiency gain of 1 to 2 orders of magnitude over the Cholesky method.

Dataset	N	D	# Iterations		Speed-up
			Machine tol	$\epsilon^2 < .1\sigma^2$	$\epsilon^2 < .1\sigma^2$
MPG	235	7	21	14	5.6x
Births in Quebec	900	1	17	2	150.0x
Bodyfat	151	14	1	1	50.3x
Boston housing	450	13	77	60	2.5x
Australian beer	428	1	15	7	20.4x
Sea levels	450	1	185	130	1.2x
Daily min temp	900	1	53	24	12.5x
Prod of H_2SO_4	415	1	53	15	9.2x

Table 3: Iterations required to solve different datasets to machine tolerance and low relative error, as well as the speed-up factor of low relative error over Cholesky factorization.

4.4 POSTERIOR VARIANCES

The posterior variance of a test point under the GP model is calculated as follows:

$$\sigma_*^2 = \mathbf{K}_* - \mathbf{K}_{*x}\mathbf{K}^{-1}\mathbf{K}_{x*} + \sigma^2 \tag{90}$$

Since posterior variances are the quantification of prediction uncertainty, in cases where it needs to be approximated it is prudent to calculate an upper bound. This allows quantities such as "a 95% confidence interval" to be translated to "a greater than 95% confidence" intervals.

We investigate two methods for calculating such upper bounds: those based on approximations to $\mathbf{K}$ and through calculating individual posterior variances with conjugate gradient, as in [Gibbs, 1997]. We propose two approximate factorizations bounds: one based on the partial SVD, similar to one proposed in[Freytag et al., 2013] and another based on a subset of the data approximation. We then show how to combine these two methods into a single consistent framework and take advantage of properties of both in the computation of posterior variances. By allowing different choices at the factorization and refinement stages, we can optimize for either space or computation constraints, take advantage of global and local structure in the variance and adapt the computation in a custom fashion to the required accuracy of individual datapoints.

4.4.1 *Subset of the data*

One of the simplest variance approximation schemes is a subset of the data approximation. It is easy to see how using a subset of the data will result in an upper bound on the variance: adding a point to a GP always reduces the uncertainty about the function (or at worst, give us no new information). By removing points from the dataset, we re-introduce this uncertainty i.e. posterior variance. More precisely if we split the datapoints into two sets, 1 and 2, we have:

$$\sigma_*^2 = \mathbf{K}_* - \begin{bmatrix}\mathbf{K}_{*1} & \mathbf{K}_{*2}\end{bmatrix}\begin{bmatrix}\mathbf{K}_{11} & \mathbf{K}_{21}\\ \mathbf{K}_{12} & \mathbf{K}_{22}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{K}_{1*}\\ \mathbf{K}_{2*}\end{bmatrix} + \sigma^2 \tag{91}$$

$$= \mathbf{K}_* - \begin{bmatrix}\mathbf{K}_{*1} & \mathbf{K}_{*2}\end{bmatrix}\begin{bmatrix}\mathbf{K}_{11}^{-1} + \mathbf{K}_{11}^{-1}\mathbf{K}_{12}\mathbf{S}^{-1}\mathbf{K}_{21}\mathbf{K}_{11}^{-1} & -\mathbf{K}_{11}^{-1}\mathbf{K}_{12}\mathbf{S}^{-1}\\ -\mathbf{S}^{-1}\mathbf{K}_{21}\mathbf{K}_{11}^{-1} & \mathbf{S}^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{K}_{1*}\\ \mathbf{K}_{2*}\end{bmatrix} + \sigma^2 \tag{92}$$

$$= \mathbf{K}_* - \mathbf{K}_{*1}\mathbf{K}_{11}^{-1}\mathbf{K}_{1*} - \left|\mathbf{K}_{*2} - \mathbf{K}_{*1}\mathbf{K}_{11}^{-1}\mathbf{K}_{12}\right|_{\mathbf{S}^{-1}} + \sigma^2 \tag{93}$$

Where $\mathbf{S}$ is the Schur complement of $\mathbf{K}$. This term can be bounded using the conjugate gradient bounds in Eq 64 and 63 on the final term with $\vec{\alpha}_i = 0$:

$$\hat{\sigma}_*^2 = \mathbf{K}_* - \mathbf{K}_{*1}\mathbf{K}_{11}^{-1}\mathbf{K}_{1*} + \sigma^2 \tag{94}$$

$$\check{\sigma}_*^2 = \max\left(0, \mathbf{K}_* - \mathbf{K}_{*1}\mathbf{K}_{11}^{-1}\mathbf{K}_{1*} - \frac{1}{\sigma^2}\|\mathbf{K}_{*2} - \mathbf{K}_{*1}\mathbf{K}_{11}^{-1}\mathbf{K}_{12}\|\right) + \sigma^2 \tag{95}$$

This formula shows us that as our chosen inducing set is able to fully replicate the covariance between the test point and the remaining training points, the lower bound becomes tight. In order to select the inducing set, we suggest a randomized greedy scheme similar to that in the Informative Vector Machine[**?**], though with a different objective function. The algorithm is shown in Figure 15.

$\mathcal{A} = \{1, ..., n\}$, $\mathcal{B} = \varnothing$
for i in $1...m$ **do**
  $\mathcal{C} = \text{Sample}(\mathcal{A}, \text{num\_samples})$
  $x = \arg\min_{c \in \mathcal{C}} \mathbf{K}_{c\mathcal{B}}\mathbf{K}_{\mathcal{B}}^{-1}\mathbf{K}_{\mathcal{B}c}$
  $\mathcal{B} = \mathcal{B} \cup \{x\}$
  $\mathcal{A} = \mathcal{A} - \{x\}$

Figure 15: Point selection scheme

Example

Figure 16 shows the true posterior variance and the upper bound obtained with the subset of data approximation for different values of M. The boudns are tighter near to the regions where there are inducing points, which are shown in red. For a good overall bounding of the variance, the approximation requires the inducing points to cover the space of training points well.

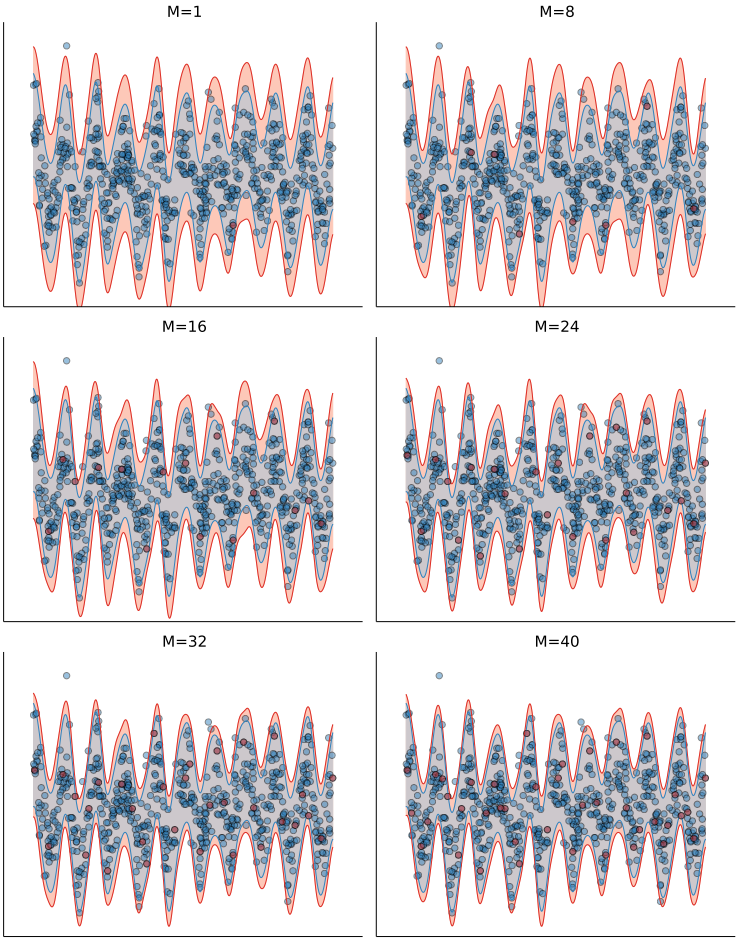


Figure 16: Subset variance bounds for the Daily min temp dataset. True variance in blue, bound in red. The inducing points are also highlighted in red.

Overall performance

The computational cost for variance bound prediction using the subset of the data approximation is $O(M^3 + M^2 N_*)$, where N_* is the number of test points. Table 4 shows the decrease in required operations relative to Cholesky decomposition by using the subset of data approximation to reach a given average bound tolerances τ. A bound tolerance of τ means that the average relative error of the bound across all test points was less than τ, ie $\tau = .5$ means that on average, the variance bound was 1.5x the true variance. For some datasets, the savings for the listed τs range from 1 to 4 orders of magnitude, which provides a compelling

justification to consider the actual requirement for posterior variance accuracy in any application of GPs.

Dataset	$\tau = .5$	$\tau = .2$	$\tau = .1$	$\tau = .05$	$\tau = .01$
MPG	1659.3x	118.0x	29.5x	7.4x	1.3x
Births in Quebec	97351.8x	97351.8x	97351.8x	97351.8x	12.6x
Bodyfat	2740.4x	2740.4x	2740.4x	2740.4x	2740.4x
Boston housing	48.1x	9.2x	3.2x	1.5x	0.5x
Australian beer	22016.4x	3522.6x	220.2x	43.5x	2.8x
Sea levels	3.2x	1.5x	1.1x	0.7x	0.3x
Daily min temp	3894.1x	155.8x	25.7x	6.1x	1.0x
Prod of H_2SO_4	368.0x	51.8x	11.2x	3.8x	0.9x

Table 4: Speed increase for different bound tolerances τ

4.4.2 *Partial SVD*

The second bound we propose is a partial SVD bound. As we know from Section 3.4.1, a valid feature decomposition of a Gram matrix is the SVD, which results in the following features:

$$\boldsymbol{\Phi}_x = \mathbf{U}\boldsymbol{\Sigma}^{\frac{1}{2}} \tag{96}$$

$$\boldsymbol{\Phi}_* = \mathbf{K}_{*x}\mathbf{U}\boldsymbol{\Sigma}^{-\frac{1}{2}} \tag{97}$$

This allows the follwing alternative formulation for the posterior variance:

$$\sigma_*^2 = \mathbf{K}_* - \mathbf{K}_{*x}\left(\mathbf{K}_- + \sigma\mathbf{I}\right)^{-1}\mathbf{K}_{x*} \tag{98}$$

$$= \boldsymbol{\Phi}_*\boldsymbol{\Phi}_*^\intercal - \boldsymbol{\Phi}_*\boldsymbol{\Phi}_x^\intercal\left(\boldsymbol{\Phi}_x\boldsymbol{\Phi}_x^\intercal + \sigma^2\mathbf{I}\right)^{-1}\boldsymbol{\Phi}_x\boldsymbol{\Phi}_*^\intercal \tag{99}$$

Using the Woodbury matrix inversion lemma:

$$= \boldsymbol{\Phi}_*\left(\sigma^{-2}\boldsymbol{\Phi}_x^\intercal\boldsymbol{\Phi}_x + \mathbf{I}\right)^{-1}\boldsymbol{\Phi}_*^\intercal \tag{100}$$

$$= \boldsymbol{\Phi}_*\left(\sigma^{-2}\boldsymbol{\Sigma} + \mathbf{I}\right)^{-1}\boldsymbol{\Phi}_*^\intercal \tag{101}$$

and since $\boldsymbol{\Sigma}$ is a diagonal matrix of the eigenvalues of $\mathbf{K}$:

$$= \sum_i \frac{\sigma^2}{\lambda_i + \sigma^2}\boldsymbol{\Phi}_{*i}^2 \tag{102}$$

If we calculate the first m singular values, then we can bound this quantity on either side, as we know that the remaining singular values will all be bounded $0 < \Sigma_j \leqslant \Sigma_m$:

$$\sigma_*^2 \geqslant \breve{\sigma}_*^2 = \sum_{i=1}^{m} \frac{\sigma^2}{\lambda_i + \sigma^2} \mathbf{\Phi}_{*i}^2 + \sum_{i=m+1}^{n} \frac{\sigma^2}{\lambda_m + \sigma^2} \mathbf{\Phi}_{*i}^2 \tag{103}$$

$$\sigma_*^2 \leqslant \widehat{\sigma}_*^2 = \sum_{i=1}^{m} \frac{\sigma^2}{\lambda_i + \sigma^2} \mathbf{\Phi}_{*i}^2 + \sum_{i=m+1}^{n} \mathbf{\Phi}_{*i}^2 \tag{104}$$

Since we only have access to the first m elements of $\mathbf{\Phi}_*$ through the partial SVD, we rephrase both the bounds in terms of only those elements.

$$\widehat{\sigma}_*^2 = \sum_{i=1}^{m} \frac{\sigma^2}{\lambda_i + \sigma^2} \mathbf{\Phi}_{*i}^2 + \sum_{i=m+1}^{n} \mathbf{\Phi}_{*i}^2 \tag{105}$$

$$= \sum_{i=1}^{m} \frac{\sigma^2}{\lambda_i + \sigma^2} \mathbf{\Phi}_{*i}^2 + \left(\sum_{i=1}^{n} \mathbf{\Phi}_{*i}^2 - \sum_{i=1}^{m} \mathbf{\Phi}_{*i}^2 \right) \tag{106}$$

Using the identity that $\sum_{i=1}^{n} \mathbf{\Phi}_{*i}^2 = \mathbf{\Phi}_* \mathbf{\Phi}_*^\intercal = \mathbf{K}_*$

$$= \mathbf{K}_* - \sum_{i=1}^{m} \left(1 - \frac{\sigma^2}{\lambda_i + \sigma^2} \right) \mathbf{\Phi}_{*i}^2 \tag{107}$$

$$= \mathbf{K}_* - \sum_{i=1}^{m} \frac{\lambda_i}{\lambda_i + \sigma^2} \mathbf{\Phi}_{*i}^2 \tag{108}$$

$$= \mathbf{K}_* - \mathbf{K}_{*x} \mathbf{U}_{1:m} \mathbf{\Sigma}_{1:m}^{-1} \widehat{\mathbf{Q}} \mathbf{U}_{1:m}^\intercal \mathbf{K}_{x*} \tag{109}$$

Where $\widehat{\mathbf{Q}}$ is defined as a digonal matrix where $\widehat{\mathbf{Q}}_{ii} = \frac{\Sigma_i}{\Sigma_i + \sigma^2}$

Combining Equations 104 and 103 we can obtain a bound on the error of the variance.

$$\widehat{\sigma}_*^2 - \sigma_*^2 \leqslant \widehat{\sigma}_*^2 - \breve{\sigma}_*^2 \tag{110}$$

$$= \sum_{i=m+1}^{n} \mathbf{\Phi}_{*i}^2 - \sum_{i=m+1}^{n} \frac{\sigma^2}{\lambda_m + \sigma^2} \mathbf{\Phi}_{*i}^2 \tag{111}$$

$$= \sum_{i=m+1}^{n} \frac{\lambda_m}{\lambda_m + \sigma^2} \mathbf{\Phi}_{*i}^2 \tag{112}$$

$$= \frac{\lambda_m}{\lambda_m + \sigma^2} \left(\mathbf{K}_* - \sum_{i=1}^{m} \mathbf{\Phi}_{*i}^2 \right) \tag{113}$$

$$= \frac{\lambda_m}{\lambda_m + \sigma^2} \left(\mathbf{K}_* - \mathbf{K}_{*x} \mathbf{U}_{1:m} \mathbf{\Sigma}_{1:m}^{-1} \mathbf{U}_{1:m}^\intercal \mathbf{K}_{x*} \right) \tag{114}$$

Example dataset

We now look at the resulting variance bounds on the same datasets as before. In this case we can see that the results are no longer local — whereas the subset bounds are affected by the location of the points in the subset, the svd bounds appear equally effective in all regions of the data and converge at a similar rate towards the correct posterior variance. For each value of M the SVD also performs better than its equivalent subset approximation.

Overall performance

As we also stated in Section 3.4.1, the $k-$partial SVD is the optimal rank-k approximation to a matrix, so it is unsurprising that this is a tighter bound than the subset of the data for a given k. However, it is also substantially more expensive, requiring $O(kN^2I)$ operations to calculate. Because of this, we would only recommend this decomposition in cases where training time computation is not at a premium, but storage or test time prediction are. Table 5 shows the decrease in required storage and required computation at test time by using the partial SVD approximation to achieve various average bound tolerances.

Dataset	$\tau = .5$	$\tau = .2$	$\tau = .1$	$\tau = .05$	$\tau = .01$
MPG	39.2x	19.6x	15.7x	7.8x	7.8x
Births in Quebec	150.0x	150.0x	150.0x	150.0x	75.0x
Bodyfat	25.2x	25.2x	25.2x	25.2x	25.2x
Boston housing	10.0x	5.0x	3.3x	2.1x	1.2x
Australian beer	71.3x	71.3x	47.6x	35.7x	28.5x
Sea levels	1.2x	1.0x	1.0x	1.0x	1.0x
Daily min temp	100.0x	15.0x	12.0x	10.0x	7.5x
Prod of H_2SO_4	13.8x	6.9x	5.5x	5.5x	4.0x

Table 5: Reduction in storage and test time computation for different bound tolerances

4.4.3 *A unified representation*

The second method for calculating an upper bound on the posterior variance of a test point is through conjugate gradient. Returning to the equation for the posterior variance:

$$\sigma_*^2 = \mathbf{K}_* - \mathbf{K}_{*x}\mathbf{K}^{-1}\mathbf{K}_{x*} + \sigma^2 \tag{115}$$

the second term is equal to twice the optimal value of the function we optimize when solving $\mathbf{K}^{-1}\mathbf{K}_{x*}$ with conjugate gradient. At each iteration of conjugate gradient we can use the bounds we have already derived in Section **??** to bound the posterior variance:

$$\sigma_*^2 \leqslant \mathbf{K}_* - \vec{v}^\intercal\mathbf{K}\vec{v} + 2\vec{v}^\intercal\mathbf{K}_{x*} + \sigma^2 \tag{116}$$

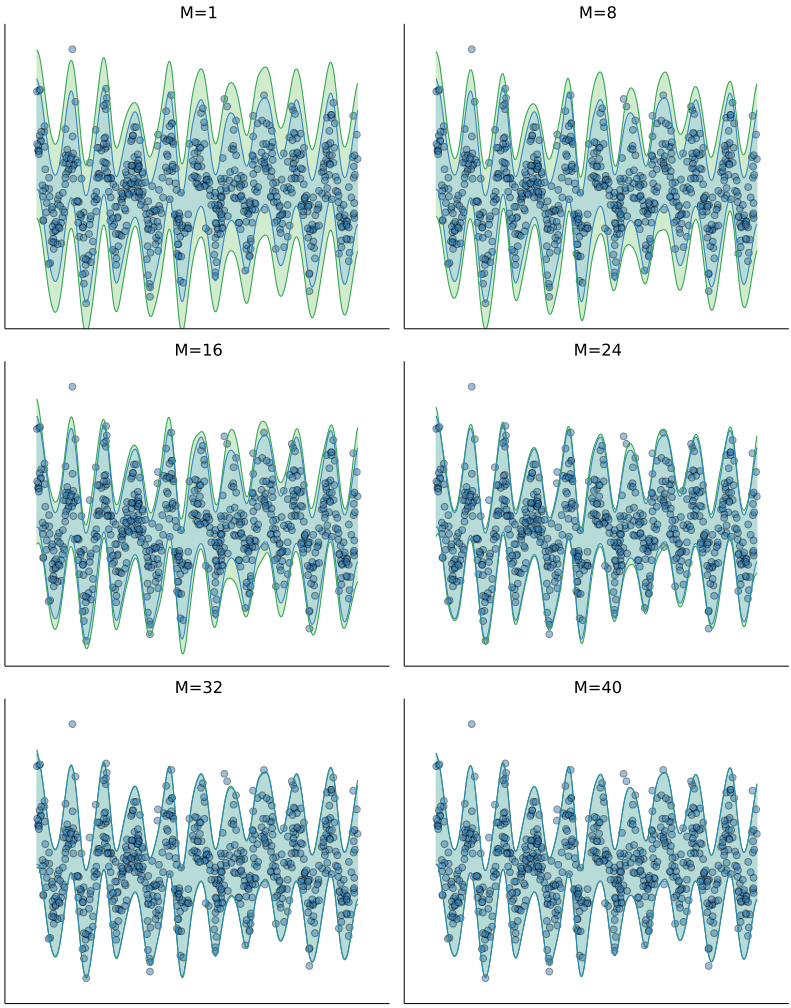


Figure 17: SVD variance bounds for Daily min temp dataset. True variance in blue, bound in green.

Where $\vec{v}$ is the vector being optimized with conjugate gradient. We can bring the subset and partial SVD bounds into this framework by considering them as a means to calculate an approximate solution to the quadratic form, which can then be used as the starting point for conjugate gradient $\vec{v}_0$. In this framework, the SVD and subset bounds of the previous two sections can be considered as using the respective approximation to determine a starting point, and then performing 0 iterations. The potential advantage of even a small number of extra

iterations is that the variance structure captured by the approximation is necessarily global, whereas each conjugate gradient system can take advantage of the local properties of the current test point. The two starting vectors can be calculated as follows:

$$\vec{v}_{subset} = \begin{bmatrix} \vec{0} \\ \mathbf{K}_m^{-1}\mathbf{K}_{m*} \end{bmatrix} \tag{117}$$

$$\vec{v}_{SVD} = \mathbf{U}_{1:m}\boldsymbol{\Sigma}_{1:m}^{-1}\hat{\mathbf{Q}}\mathbf{U}_{1:m}^{\mathsf{T}}\mathbf{K}_{x*} \tag{118}$$

In Equation 117, the non-zero entries correspond to the indicies of m.

Performing this refinement of variance at test time with conjugate gradient is comparatively expensive; every iteration on a test point requires $O(N^2)$ operations at test time, whereas the initial bound can be calculated in only $O(NM^2)$ operations. This can be useful in cases where very accurate variances are important on only a small subset of the test points. However, Chapter 5 will introduce sets of kernels for which this refinement can be performed much faster, making this view useful in a broader range of applications.

Example dataset

Figure 18 shows the improvement in posterior variance bounds as conjugate gradient is applied. The dataset chosen is Sea levels for which the subset and SVD bounds showed very little improvement over full factorization. This shows that for this dataset, we can achieve speed-ups over Cholesky decomposition by using conjugate gradient if predictions are only required at a small number of test locations.

Overall performance

To illustrate the effects of running the conjugate gradient system, we look at the number of iterations required to achieve a given error bound on the variance, initializing with the subset of data approximation with $M = \sqrt{N}$. This is shown in Table 6. The table confirms that using conjugate gradient the bounds converge exponentially with the number of iterations.

Dataset	$\tau = .1$	$\tau = .05$	$\tau = .01$	$\tau = .005$	$\tau = .001$
MPG	4	6	8	8	9
Births in Quebec	1	1	1	1	2
Bodyfat	1	1	1	1	1
Boston housing	11	15	21	21	27
Australian beer	1	4	6	6	7
Sea levels	12	18	30	33	42
Daily min temp	1	4	7	9	11
Prod of H_2SO_4	2	4	8	9	12

Table 6: Extra iterations of conjugate gradient required to achieve certain variance bound errors

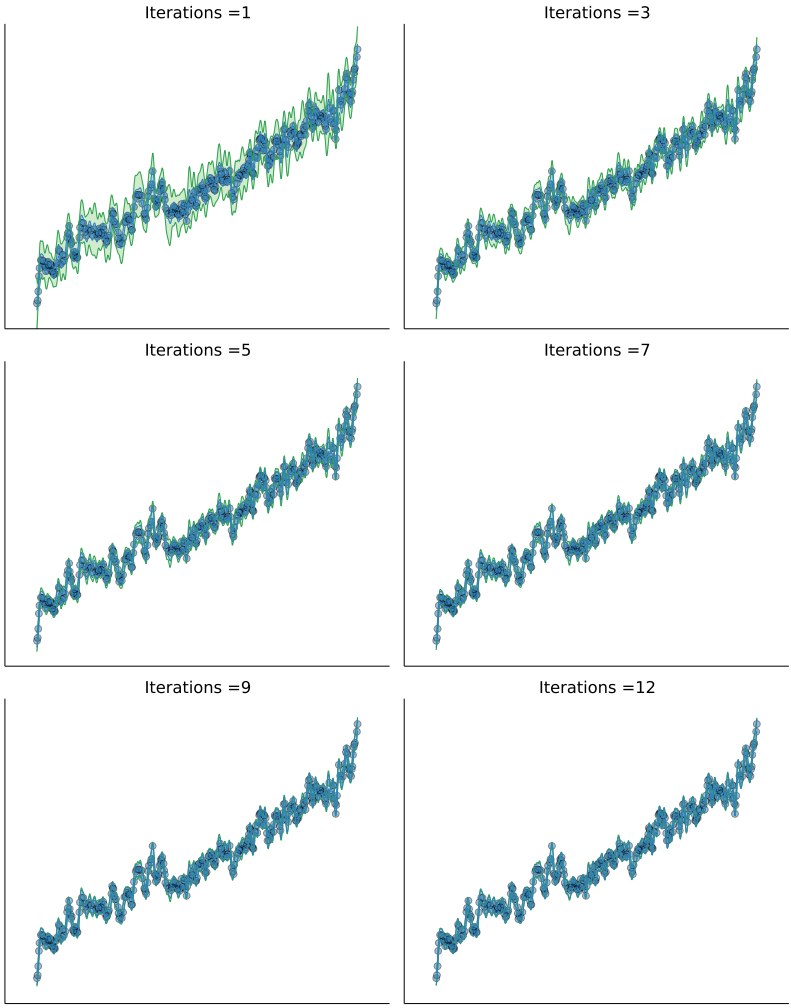


Figure 18: Improvement in variance bound for different number of refining CG iterations on the Sea level dataset.

4.5 DERIVATIVES

The derivative of the log-likelihood has the following form:

$$\frac{d\mathcal{L}}{d\theta_i} = -\frac{1}{2}\vec{y}^{\mathsf{T}}\mathbf{K}^{-1}\frac{d\mathbf{K}}{d\theta_i}\mathbf{K}^{-1}\vec{y} - \frac{1}{2}Tr\left(\mathbf{K}^{-1}\frac{d\mathbf{K}}{d\theta_i}\right) \tag{119}$$

We will need to use a stochastic trace estimator to calculate the second term, so immediately we can see that this will be the more computationally expensive term: for every iteration of the trace estimator, we will need to solve a linear system with $\mathbf{K}$. So while the first term requires $O(N^2 I_1)$ operations, where I_1 is the number of iterations to solve the linear system, the second term requires $O(N^2 I_1 I_2)$ operations, where I_2 is the number of samples required for an accurate estimate from the stochastic trace estimator.

In order to calculate the second term we can choose any of the stochastic trace estimators from Section 4.2.2. The iterative GP framework of [Gibbs and MacKay, 1997] used Gaussian random vectors, though Hutchinson random vectors are more common in other applications, being called the "standard Monte Carlo estimator"[Avron and Toledo, 2011] for this problem. Since each sample for the estimator is quite expensive — it requires the solution of another linear system — it is important to cache the solutions to these linear systems for use in calculating different derivatives. To do this we define $\vec{\beta}_j = \mathbf{K}^{-1}\vec{s}_j$, where $\vec{s}_j$ are sampled from our stochastic trace estimator distribution. For each parameter, we then use the formula:

$$Tr\left[\mathbf{K}^{-1}\frac{d\mathbf{K}}{d\vec{\theta}_i}\right] \approx \frac{1}{M}\sum_{j=1}^{M}\vec{\beta}_j^{\mathsf{T}}\frac{d\mathbf{K}}{d\vec{\theta}_i}\vec{s}_j \tag{120}$$

Where M is the number of samples used to estimate the trace. Since the computation for the first term of Eq requires one linear system solve, whereas the second will require M, it is reasonable to solve $\vec{\alpha}_{opt}$ to a high accuracy. Then, for each iteration of the stochastic trace estimator, we require that the sample be accurate to within a small tolerance of the resulting estimate. To bound the error on the second term, we require a more general bound on the error of the solution vector of a quadratic form than we have so far derived. We derive a bound on the norm of the error vector $\vec{\epsilon}$ for a given solution vector $\vec{v}$ to the system defined by $\mathbf{K}^{-1}\vec{y}$. Starting with the error bound on the objective function from Equation 73:

$$B = \frac{1}{2\sigma^2}\|\vec{y} - \mathbf{K}\vec{v}\|^2 \tag{121}$$

$$= \frac{1}{2\sigma^2}\|\mathbf{K}\vec{\epsilon}\|^2 \tag{122}$$

$$\geqslant \frac{1}{2\sigma^2}\lambda_{min}\left(\mathbf{K}\right)^2\|\vec{\epsilon}\|^2 \tag{123}$$

$$\geqslant \frac{\sigma^2}{2}\|\vec{\epsilon}\|^2 \tag{124}$$

The error bound we wish to enforce is that the error on our sample is a fraction η of the current estimate of the derivative:

$$\vec{\epsilon}_j^\intercal \frac{d\mathbf{K}}{d\theta_i}\vec{s}_j \leqslant \eta\left(\widetilde{\frac{d\mathcal{L}}{d\theta_i}} + \vec{\beta}_j^\intercal \frac{d\mathbf{K}}{d\theta_i}\vec{s}_j\right) \tag{125}$$

$$\left|\vec{\epsilon}_j\right|\left|\frac{d\mathbf{K}}{d\theta_i}\vec{s}_j\right| \leqslant \eta\left(\widetilde{\frac{d\mathcal{L}}{d\theta_i}} + \vec{\beta}_j^\intercal \frac{d\mathbf{K}}{d\theta_i}\vec{s}_j\right) \tag{126}$$

$$\sqrt{\frac{2B}{\sigma^2}}\left|\frac{d\mathbf{K}}{d\theta_i}\vec{s}_j\right| \leqslant \eta\left(\widetilde{\frac{d\mathcal{L}}{d\theta_i}} + \vec{\beta}_j^\intercal \frac{d\mathbf{K}}{d\theta_i}\vec{s}_j\right) \tag{127}$$

$$B \leqslant \frac{2}{\sigma^2}\left(\eta\left(\widetilde{\frac{d\mathcal{L}}{d\theta_i}} + \vec{\beta}_j^\intercal \frac{d\mathbf{K}}{d\theta_i}\vec{s}_j\right)\left|\frac{d\mathbf{K}}{d\theta_i}\vec{s}_j\right|^{-1}\right)^2 \tag{128}$$

This is the termination criteria that we use for the conjugate gradient systems that make up each trace estimate, ensuring this is true for every hyper-parameter $\vec{\theta}_i$. The second element we require is when to terminate the stochastic trace estimator. MacKay and Gibbs[Gibbs, 1997] performed experiments by setting the number of samples to be 2. We show that while in some instances only 2 samples results in very high accuracy, this is not universally true. The error on the second term can be bounded probabilistically; since the estimator of the matrix trace is a mean of independent samples, an approximate probability that the error in the estimate is less than .01 can be derived from the standard error:

$$Pr\left(\left|Tr\left(\mathbf{K}^{-1}\frac{d\mathbf{K}}{d\theta_i}\right) - \sum \vec{\beta}_j^\intercal \frac{d\mathbf{K}}{d\theta}\vec{s}_j\right| \leqslant .01\widetilde{\frac{d\mathcal{L}}{d\theta_i}}\right) \geqslant .95 \tag{129}$$

$$2\sqrt{Var\left(\sum \vec{\beta}_j^\intercal \frac{d\mathbf{K}}{d\theta}\vec{s}_j\right)} \leqslant .01\widetilde{\frac{d\mathcal{L}}{d\theta_i}} \tag{130}$$

$$200\sqrt{\frac{1}{M}Var\left(\vec{\beta}_j^\intercal \frac{d\mathbf{K}}{d\theta}\vec{s}_j\right)} \leqslant \widetilde{\frac{d\mathcal{L}}{d\theta_i}} \tag{131}$$

This can be used as a criterion for terminating sampling.

4.5.1 *Experiments*

To assess the efficacy of this estimator, we sample hyper-parameters from:

$$\vec{a} = \mathcal{N}(0, 3\mathbf{I}) \tag{132}$$

$$\vec{\theta} = exp(\vec{a}) \tag{133}$$

and assess the derivatives and associated quantities at those points.

Accuracy of two samples

Figure 19 shows the log relative error in the estimate of the derivative when it is calculated by taking the fixed rule of two samples. The dataset is generated from the function $f(x) = \sin(x) + x$. Approximately one third of the derivatives have a realative error greater than 1, which is far too high for a non-stochastic gradient method to use. This level of error is a strong argument for an adaptive sample size rule.

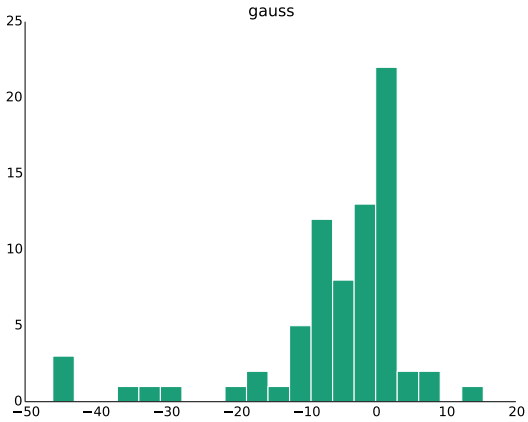


Figure 19: Log relative error of trace estimator for 2 Gaussian samples on a simple 1D dataset

Distribution of iterative convergence across hyper-parameters

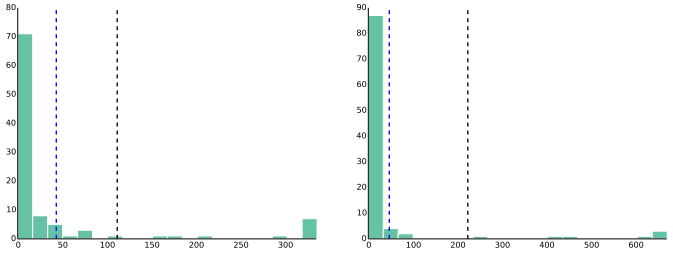
Figure 20 shows the distribution of iterations required to solve the linear systems. The black dotted line represents the cost of Cholesky factorization, and the blue dotted line represents the average cost of solving a linear system for this distribution of hyper-parameters. We can see from the samples to the right of the black lines that the iterative method is not universally faster than the Cholesky decomposition.

Comparison of trace estimators

Figure 21 shows the number of the samples required to achieve a given relative accuracy of .01 on the Boston housing dataset using the different trace estimators, with the average shown as the dotted black line. It shows that for this dataset all three proposed trace estimators outperform the Gaussian estimator.

Overall performance

Table 7 shows the reduction in operations required to calculate derivatives to .01 relative error using the stochastic estimator, rather than Cholesky factorization.



(a) Sea level

(b) Daily min temp

(c) Beer production in Australia

(d) Births in Quebec

Figure 20: Distribution of required iterations to convergence for RBF on different datasets

This shows that across these datasets the stochastic estimators are universally faster, with the Hutchinson performing the best overall.

Dataset	Gaussian	Rayleigh	Hutchinson	Unit
MPG	5.2x	7.9x	**8.3x**	5.0x
Births in Quebec	**26.4x**	**26.4x**	**26.4x**	**26.4x**
Bodyfat	4.0x	7.8x	**8.0x**	5.6x
Boston housing	5.6x	6.7x	**7.1x**	4.4x
Australian beer	**17.0x**	**17.0x**	**17.0x**	**17.0x**
Community crime	530.0x	**843.2x**	**843.2x**	**843.2x**
Sea levels	18.9x	**19.1x**	**19.1x**	**19.1x**
Daily min temp	5.8x	**6.2x**	**6.2x**	5.3x
Prod of H_2SO_4	**19.2x**	**19.2x**	**19.2x**	**19.2x**

Table 7: Average reduction in computation required to calculate derivatives using different trace estimators

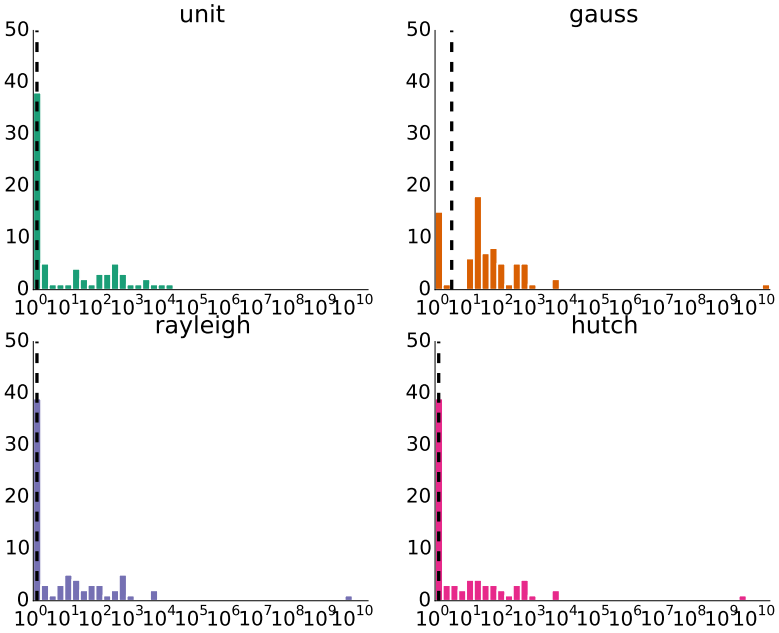


Figure 21: Number of samples required to achieve .01 relative error when calculating derivatives on the Boston housing dataset

4.6 LOG-LIKELIHOOD

The log-likelihood has the following form:

$$\mathcal{L} = -\frac{1}{2}\vec{y}^{\intercal}\mathbf{K}^{-1}\vec{y} - \frac{1}{2}\log(\det(\mathbf{K})) - \frac{N}{2}\log(2\pi) \tag{134}$$

This quantity is used in some cases in gradient descent algorithms, and also in cases where models need to be compared against each other. As this quantity is not necessary for some methods of optimization[**?**], discussion of this was omitted from Mackay and Gibbs[Gibbs and MacKay, 1997]. However, there are many applications where model scoring is important.

We already know how to bound the first term with conjugate gradient; the approximation of the log-determinant is somewhat more difficult. The efficient calculation of this quantity has been investigated in a number of different contexts[Reusken, 2000, Bai and Golub, 1996, Hale et al., 2008, Aune et al., 2014] and specifically in GPs[Stein et al., 2013, Zhang and Leithead, 2007]. The approach we take is to use an expansion of the log-determinant. The work of [McCourt, 2008] extends [Barry and Kelley, 1999] to demonstrate how to create such an approximation to the log-determinant for an arbitrary PSD matrix using the largest

singular value and the Rayleigh trace estimator. We extend this by proposing the use of two different bounds on the largest singular value, as well as the alternative trace estimators shown in Section 4.2.2.

We use the following relationship, where log is the matrix logarithm:

$$\log\det \mathbf{X} = Tr(\log \mathbf{X}) \tag{135}$$

as well as the Mercator series expansion for the matrix logarithm:

$$\log(\mathbf{I} + \mathbf{X}) = \sum_{1=1}^{\infty} \frac{(-1)^i}{i} \mathbf{X}^i \qquad \text{for } \mathbf{X} \text{ with } -1 < \lambda_{max}(\mathbf{X}) < 1 \tag{136}$$

Which combine to give:

$$\log\det(\mathbf{I} + \mathbf{X}) = \sum_{1=1}^{\infty} \frac{(-1)^i}{i} Tr\left(\mathbf{X}^i\right) \tag{137}$$

For $\mathbf{X}$ with $-1 < \lambda_{max}(\mathbf{X}) < 1$.

To obtain a matrix whose largest singular value is bounded between -1 and 1, it is sufficient to divide the matrix by its largest singular value. However, it is worth noting that the calculation of the largest singular value can itself be quite expensive, and crucially its convergence depends on a different quantity[2] than the trace estimators. We therefore propose a more general formulation using an upper bound on the largest singular value $\widehat{\lambda}_{max} \geqslant \lambda_{max}$. Two bounds that we will use in experiments are the trace bound and Gershgorin bound.

$$\lambda_{max}(\mathbf{K}) \leqslant Tr(\mathbf{K}_{-}) + \sigma^2 \qquad \text{Trace bound} \tag{138}$$

$$\lambda_{max}(\mathbf{K}) \leqslant \arg\max_i \sum_{j=1}^{N} \|\mathbf{K}_{ij}\| \qquad \text{Gershgorin bound} \tag{139}$$

Using a bound on $\lambda_{max}(\mathbf{K})$ we can rewrite the log-determinant of a kernel matrix as follows:

$$\log\det(\mathbf{K}) = \log\det\left(\widehat{\lambda}_{max}(\mathbf{K}) \frac{\mathbf{K}}{\widehat{\lambda}_{max}(\mathbf{K})}\right) \tag{140}$$

$$= n\log\widehat{\lambda}_{max}(\mathbf{K}) + \log\det\left(\mathbf{I} + \left(\frac{\mathbf{K}}{\widehat{\lambda}_{max}(\mathbf{K})} - \mathbf{I}\right)\right) \tag{141}$$

$$:= n\log\widehat{\lambda}_{max}(\mathbf{K}) + \log\det\left(\mathbf{I} + \mathbf{D}\right) \tag{142}$$

$$= n\log\widehat{\lambda}_{max}(\mathbf{K}) + \sum_{1=1}^{\infty} \frac{(-1)^i}{i} Tr\left(\mathbf{D}^i\right) \tag{143}$$

2 the convergence rate of iterative methods for finding the largest singular value is proportional to $\frac{\lambda_1(\mathbf{K})}{\lambda_2(\mathbf{K})}$

The choice of $\mathbf{D}$ has been made in such a way that its eigenvalues are bounded between -1 and 1. We can now approximate this quantity by truncating the series at a fixed length m and using one of the stochastic trace estimators from the previous section.

$$\log\det(\mathbf{K}) \approx n\log\widehat{\lambda}_{max}(\mathbf{K}) + \sum_{1=1}^{m}\frac{(-1)^i}{i}\widetilde{Tr}\left(\mathbf{D}^i\right) \tag{144}$$

Each term in the expansion can be bounded as follows:

$$\|Tr(\mathbf{D}^i)\| < N\lambda_{max}(\mathbf{D}^i) \tag{145}$$

$$< N\left(1 - \frac{\lambda_{min}(\mathbf{K})}{\widehat{\lambda}_{max}(\mathbf{K})}\right)^i \tag{146}$$

While the error for truncating at m can be bounded with this term, it is not generally tight enough to be useful in practice. However, this shows us that, like linear systems, the convergence rate depends largely upon the condition number of $\mathbf{K}$. As such, it is likely that pre-conditioning will be as important for approximating the log-determinant. Pre-conditioning for this estimator can be achieved with the following identity:

$$\log\det\mathbf{K} = \log\det\left(\mathbf{P}\mathbf{P}^{-1}\mathbf{K}\right) = \log\det\mathbf{P} + \log\det\left(\mathbf{P}^{-1}\mathbf{K}\right) \tag{147}$$

The effects of this will be reviewed in Section 4.7.

The total number of operations required is based on the truncation of the series, m, and the number of samples required for the trace estimates, p, leading to a total computational cost of $O(mpN^2)$ operations. p can be chosen based on the sample variance, as with the calculation of the derivative, and the series can be truncated at a point where the terms fall below a given tolerance.

4.6.1 *Experiments*

To assess this approximation, we evaluate it at a set of hyper-parameters sampled from the same distribution as Section 4.5.

Trace estimator comparison

We first look at the distribution of required p for different estimators, using the same distribution of hyper-parameters as Section 4.5.1, illustrated in Figure 22. Once again, the three proposed estimators outperform the Gaussian.

Series truncation

Figure 23 shows the number of terms of the series that are needed to achieve a .01 relative error tolerance on the bodyfat dataset using different bounds on the largest eigenvalue. The values of m are bounded at 50, as this is the point at which the iterative method is no more efficient than Cholesky decomposition. The trace bound is clearly inferior to both the Gershgorin and SVD, a trend which is consistent across all datasets.

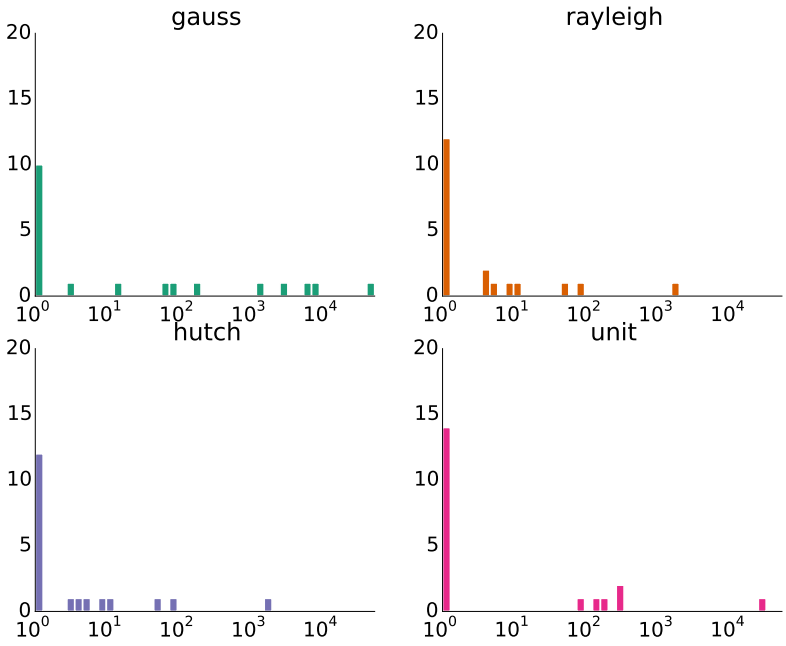


Figure 22: Histogram of required samples to achieve .01 relative accuracy on the Boston housing dataset

Overall performance

Table 8 shows the average computational savings by using the iterative method to an accuracy of .01 over Cholesky factorization. For this calculation, the Rayleigh and Hutchinson provide a substantial speed-up over the Unit and Gaussian sampling schemes. While the performance increase compared to factorization is not as large as for the previous quantities, there are still order of magnitude speed-ups for some datasets.

4.7 PRE-CONDITIONERS

Due to the importance of pre-conditioning linear systems, there is a literature dedicated to the choice of pre-conditioners, and to "general purpose" pre-conditioners[Benzi, 2002, Srinivasan et al., Barrett et al., 1994]. However, by definition a general purpose pre-conditioner can only ever be so useful — the entire point of a pre-conditioner is to integrate prior knowledge about the problem structure in order to convert it to a related problem that is easer to solve iteratively. For GPs there are many pre-conditioners that use detailed knowledge of the problem, and knowledge about individual kernels, that can be easily obtained from standard

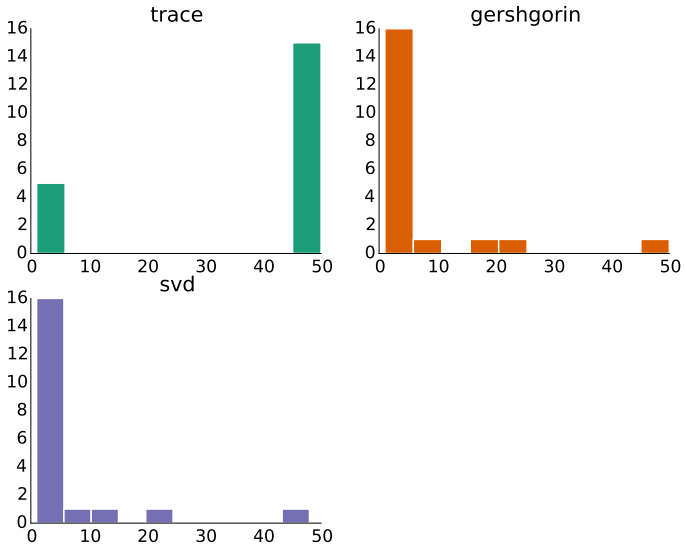


Figure 23: Histogram of required value of m for different bounds on the bodyfat dataset

Dataset	Gaussian	Rayleigh	Hutchinson	Unit
MPG	9.5x	62.7x	62.7x	18.9x
Births in Quebec	1.2x	3.4x	3.4x	3.6x
Boston housing	2.7x	4.3x	4.4x	4.5x
Australian beer	1.4x	2.9x	2.9x	5.2x
Sea levels	2.0x	2.7x	2.7x	3.3x
Daily min temp	1.1x	3.5x	3.5x	3.7x
Bodyfat	5.2x	15.6x	15.8x	12.6x
Prod of H_2SO_4	1.4x	2.6x	2.6x	3.9x

Table 8: Reduction in computation required to calculate the log-likelihood using different trace estimators

methods for approximate GPs.

At a very general level approximate GP methods[Snelson and Ghahramani, 2006, Williams and Seeger, 2001] tend to share a common base: they approximate the full kernel matrix $\mathbf{K}$ with a matrix $\tilde{\mathbf{K}}$ which can be inverted efficiently and *use that matrix in place of the full kernel matrix*. Returning to the definition of pre-conditioners, the pre-conditioning matrix $\mathbf{P}$ shoudl be chosen such that $\mathbf{P}^{-1}\mathbf{K} \approx \mathbf{I}$ (which can be satisified by $\mathbf{P} \approx \mathbf{K}$) and such that $\mathbf{P}$ can be efficiently inverted. So, rather than using $\tilde{\mathbf{K}}$ in place of $\mathbf{K}$, it can be used as an iterative pre-conditioner $\mathbf{P} = \tilde{\mathbf{K}}$. When the aim of a Gaussian Process approximation is to approximate

the full GP (as opposed to provide a different prior[Quiñonero Candela and Rasmussen, 2005]), then using the framework of pre-conditioning offers substantial advantages. Foremost, it avoids the problem of catastrophic failure. Most GP approximations are parameterized by an integer M, representing the number of data points or basis functions used in the approximation. As M becomes too small to approximate the kernel matrix well, directly using the approximate kernel matrix as a replacement for the true one will result in arbitrarily poor predictions. Preconditioning the linear system, on the other hand, will still return bounds on the solution error and will simply converge more slowly.

By setting the initial point of optimization, $v_0 = \tilde{\mathbf{K}}^{-1}\vec{y}$, the direct approximation is a special case where the number of iterations are 0 and the bounds are not checked. As an iteration or bound check requires $O(N^2)$ operations, both may be too expensive in some circumstances. In this case however, an alternative is to consider restricted classes of kernels (Chapter 5), rather than to use a direct approximation and hope for the best.

4.7.1 *Example pre-conditioners*

This section will look at the effect of a number of different pre-conditioners on the convergence rate of the iterative solution to an SE Gram matrix. All the pre-conditioners can be computed and inverted in $O(NM^2)$ operations, with the exception of the partial SVD, which requires $O(N^2 I)$.

Local GPs (Block Jacobi)

The Block Jacobi pre-conditioner is one of the simplest general pre-conditioners and is equivalent to the local GPs model[Snelson and Ghahramani, 2007]. In this context it is constructed by choosing subset of points of size M and forming a block diagonal matrix $\mathbf{P}$ where $\mathbf{P}_{ij} = k(\vec{x}_i, \vec{x}_j)$ if the points are in the same subset, and 0 if not. Since the resulting matrix is block diagonal, it can be inverted in $O(NM^2)$ operations.

$$\widetilde{\mathbf{P}} = \mathbf{K} \circ (\mathbf{I} \otimes \mathbf{1})$$

where $\circ$ is element-wise multiply and $\otimes$ is the Kroneckor product.

Nystrom approximation

The Nystrom approximation[Williams and Seeger, 2001, Drineas and Mahoney, 2005] for a Gram matrix is a low-rank approximation constructed from a subset of the training points m.

$$\widetilde{\mathbf{P}} = \mathbf{K}_{xm}\mathbf{K}_m\mathbf{K}_{mx} + \sigma^2\mathbf{I}$$

PITC

PITC[Snelson and Ghahramani, 2007] is an approximate GP model that exactly models the full GP relationship within clusters of the data and approximates the remaining covariances with a low-rank matrix (The model of PITC is described more fully in Section 5.2.3). The resulting approximation is:

$$\widetilde{\mathbf{P}} = \mathbf{K}_{xm}\mathbf{K}_m^{-1}\mathbf{K}_{mx} + \left(\mathbf{K} - \mathbf{K}_{xm}\mathbf{K}_m^{-1}\mathbf{K}_{mx}\right) \circ (\mathbf{I} \otimes \mathbf{1})$$

From the form of the approximation, we can see that PITC can be though of as a combination of the local GPs and Nystrom approximation.

Partial SVD

Another general preconditioner is to take the M-partial SVD of the noise-free kernel matrix:

$$\widetilde{\mathbf{P}} = \mathbf{U}_{1:m}\mathbf{\Sigma}_{1:m}\mathbf{U}_{1:m}^{\mathsf{T}} + \sigma^2\mathbf{I}$$

This will be impractical in most circumstances, due to the high cost of the partial SVD, but is included in the experiments as a useful benchmark.

4.7.2 *Results*

We assess the effect of the different pre-conditioners on the convergence of solving linear systems, the speed-up of which affects the mean calculation, posterior variance refinement and derivative calculation, and the effect of the pre-conditioners on the approximation to the log-determinant.

Pre-conditioning linear systems

Figure 24 and Figure 25 show the improvement that can be made by pre-conditioning linear systems. Figure 24 shows the distribution of the ratio of the iterations required by pre-conditioned systems compared to the standard solver and Figure 25 shows how the average improvement changes as the approximation quality M is increased. For this dataset PITC is the best pre-conditioner, although this is not true for all the datasets. Those that have more local structure tend to be better served with a PITC pre-conditioner, while those without can be solved faster with an SVD pre-conditioner. In all cases these two were superior to the Block Jacobi and Nystrom approximations.

The overall results for decrease in computation of derivatives is listed in Table 10. The savings are shown for pre-conditioners where $M = \sqrt{N}$, which is the point at which the cost of inverting the pre-conditioner is equal to the number operations for one iteration of conjugate gradient. While this point is not necessarily the optimal in terms of computational efficiency, it is a sensible default in that it asymptotically adds no computational complexity and in practice the cost of one extra iteration is small regardless of problem size. It shows that for these datasets, a PITC pre-conditioner achieves a modest improvement of between 1-3x over standard conjugate gradient.

Pre-conditioning the log-likelihood

Figure 26 shows the reduction in required terms to effectively approximate the log-likelihood for the sea level dataset. The PITC pre-conditioner brings the stochastic estimator to be more efficient in all but one out of twenty instances. The effect of the pre-conditioning is also shown in Table 11, by comparison with

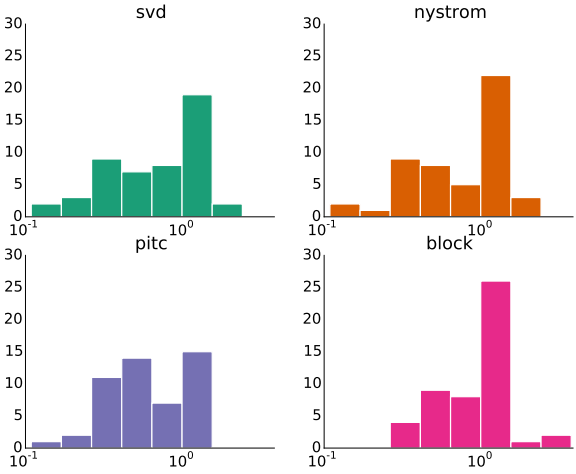


Figure 24: Ratio of required iterations between standard and pre-conditioned systems on the simple 1D dataset. Values of greater than 1 indicate a pre-conditioner that slowed down convergence.

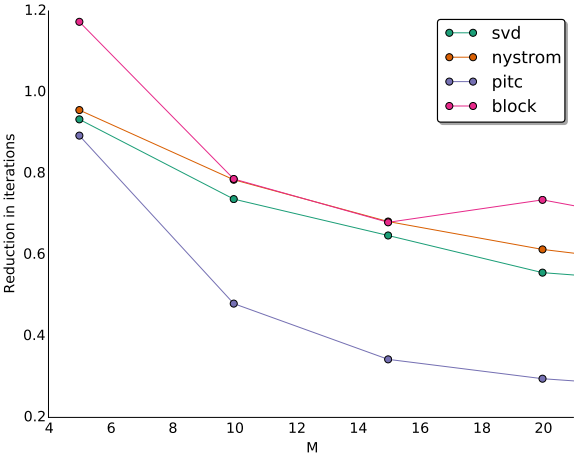


Figure 25: Reduction in required operations due to different pre-conditioners on the simple 1D dataset

Table 8, the number of operations required is reduced by between 4x-40x. It is interesting to see that for two datasets the unit estimator performs better on the pre-conditioned system, but the Hutchinson estimator remains the best general purpose choice.

	SVD	Nystrom	PITC	Block
MPG	**4.7x**	3.5x	3.5x	0.5x
Births in Quebec	**3.0x**	**3.0x**	**3.0x**	1.5x
Bodyfat	**1.0x**	**1.0x**	**1.0x**	**1.0x**
Boston housing	**2.4x**	1.8x	1.4x	0.4x
Australian beer	**7.0x**	**7.0x**	**7.0x**	0.7x
Sea levels	0.8x	0.8x	**1.9x**	0.8x
Daily min temp	**2.9x**	2.3x	2.3x	0.7x
Prod of H_2SO_4	2.1x	1.7x	**2.5x**	0.9x

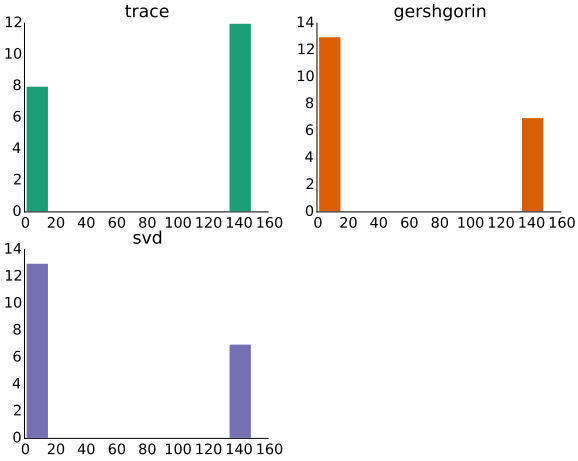
Table 9: Decrease in number of iterations to solve the linear system at optimal hyper-parameters over standard conjugate gradient by using various pre-conditioners with $M = \sqrt{N}$

	Gaussian	Rayleigh	Hutchinson	Unit
MPG	0.8x	**1.5x**	**1.5x**	**1.5x**
Births in Quebec	**2.8x**	**2.8x**	**2.8x**	**2.8x**
Bodyfat	1.2x	1.8x	1.8x	**2.8x**
Boston housing	1.1x	**1.4x**	**1.4x**	0.9x
Australian beer	**1.6x**	**1.6x**	**1.6x**	**1.4x**
Sea levels	2.3x	2.5x	2.5x	**2.6x**
Daily min temp	1.5x	1.6x	1.6x	**2.3x**
Prod of H_2SO_4	**2.3x**	**2.3x**	**2.3x**	**2.3x**

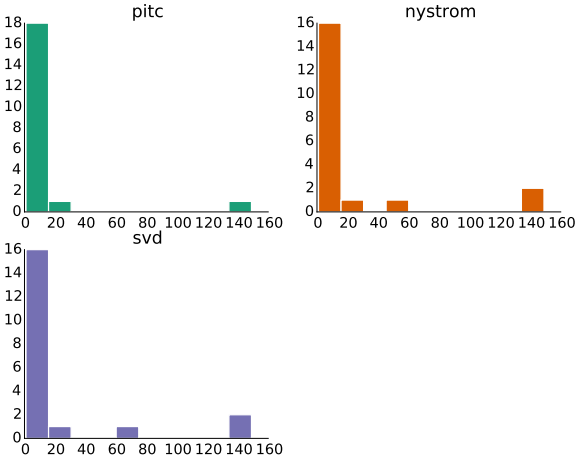
Table 10: Decrease in operations required to calculate derivatives over standard conjugate gradient by using the PITC pre-conditioner with $M = \sqrt{N}$

	Gaussian	Rayleigh	Hutch	Unit
MPG	**180.8x**	**180.8x**	**180.8x**	**180.8x**
Births in Quebec	9.6x	98.9x	**101.7x**	15.4x
Bodyfat	39.7x	**43.1x**	**43.1x**	**43.1x**
Boston housing	5.9x	19.4x	**19.5x**	14.5x
Australian beer	10.1x	37.1x	38.2x	**83.1x**
Sea levels	5.0x	**12.5x**	**12.5x**	12.0x
Daily min temp	8.8x	102.9x	**104.7x**	26.4x
Prod of H_2SO_4	17.6x	122.1x	122.1x	**166.0x**

Table 11: Decrease in operations required to calculate the log-determinant by using the iterative estimator with a PITC pre-conditioner with $M = \sqrt{N}$ compared with Cholesky factorization



(a) Without pre-conditioning



(b) Using the Gershgorin bound with pre-conditioning

Figure 26: Distribution on the number of required terms m to achieve .01 relative error in the log-determinant for the sea level dataset.

5

EFFICIENT KERNELS: $O(N^2 I) \rightarrow O(NMI)$

To them, I said, the truth would be literally nothing but the shadows of the images.

— Plato, The Republic

A key property of the framework of Chapter 4 is that the methods never require direct access to the elements of the Gram matrix, as they only interact with it through the evaluation of the dot product of a vector with the matrix[1]. The N^2 term from the $O(N^2 I)$ operations required to solve a GP in the framework comes from the N^2 operations required to perform this matrix-vector product. This means that the framework can be substantially faster than $O(N^2 I)$ when using a kernel for which the Gram matrix can be multiplied by a vector in less than N^2 operations. There are many such kernels, and we will classify them by defining the idea of an M-efficient kernel. An M-efficient kernel is one for which the matrix-vector product with a Gram matrix of size N that it generates can be performed in $O(NM)$ operations. While there have been investigations to construct routines for approximate matrix-vector products[Greengard and Strain, 1991, Lang et al., 2005, Yang et al., 2003] that are very effective under certain circumstances, we restrict our focus here to classes of kernels for which the matrix-vector product can be calculated exactly.

Feature engineering is often cited as the *"the most important factor"* [Domingos, 2012] for the success of learning algorithms. For kernel methods, this directly translates to the kernel being the most important factor for success. There will never be a single best kernel for all tasks [**?**], so the utility of kernels methods is increased by having a large array of kernels available.

In this chapter we propose three classes of kernels with controllable complexity M. The first is based on graphical models and includes commonly used sparse kernels such as FITC and PITC. The second is a new style of kernel based on random partitions. The final class is designed to model global and local behaviour in high dimensions and is based on compactly supported kernels.

5.1 BACKGROUND

In the inference framework of Chapter 4, the number of operations required to solve a GP is proportional to the time taken to perform the dot product of the

1 With the exception of the trace bounds in Section 4.6

Gram matrix with a vector, which we will hereafter refer to as a matrix-vector product. Additionally, the space requirement is proportional to the space required to store the Gram matrix. To construct a framework for M-efficient kernels we review class of matrices that can be stored in $O(NM)$ and where the matrix-vector product can be performed in $O(NM)$ operations

There are a number of classes of structured matrices that obey this property, however most are not useful in the context of GPs as it is difficult to design general kernels to satisfy the structural assumptions in these matrices. Toeplitz structured kernel matrices are encountered in practice, but only when a stationary kernel is evaluated on a grid[Saatci]. In this section we restrict our focus to kernels that can be applied to arbitrary datasets. There are two classes of matrices that, along with the $+$ operator, will form the basis for all the efficient kernels we discuss. These are low-rank matrices and sparse matrices.

5.1.1 *Low-rank matrices*

Low-rank symmetric matrices can be expressed as $\mathbf{K} = \mathbf{U}\mathbf{U}^\intercal$, where $\mathbf{U} \in \mathbb{R}^{N \times M}$, which is illustrated in Figure 27.

Figure 27: Illustration of a low-rank matrix

Matrix-vector products can be calculated in $O(NM)$ operations by performing the calculation in the following order:

$$\mathbf{K}\vec{v} = \mathbf{U}\left(\mathbf{U}^\intercal \vec{v}\right) \tag{148}$$

If we have the matrix in the above form, then we also have direct access to all its eigenvalues.

Lemma 5.1.1. $\mathbf{U}\mathbf{U}^\intercal$ *has* M *non-zero eigenvalues, and the corresponding eigenvalues and eigenvectors are* $\left\{ \frac{1}{\|\mathbf{U}_i\|}\mathbf{U}_i, \|\mathbf{U}_i\|^2 \right\}$.

Proof. Let $\mathbf{\Sigma}$ be the diagonal matrix of eigenvalues i.e. $\mathbf{\Sigma}_{ii} = \|\mathbf{U}_i\|^2$. The singular value decomposition can be written as

$$\mathbf{K} = \left(\mathbf{U}\mathbf{\Sigma}^{-\frac{1}{2}}\right)\mathbf{\Sigma}\left(\mathbf{U}\mathbf{\Sigma}^{-\frac{1}{2}}\right)^\intercal \tag{149}$$

$$= \mathbf{U}\mathbf{U}^\intercal \tag{150}$$

□

From which it follows that the largest eigenvalue is:

$$\lambda_{max}\left(\mathbf{U}\mathbf{U}^{\intercal}\right) = \arg\max_{i} \|\mathbf{U}_i\|^2 \tag{151}$$

5.1.2 *Sparse matrices*

Sparse matrices are matrices that have a small number of non-zero entries compared to their size. We will use the terminology M-sparse matrices to refer to sparse matrices which have $O(M)$ non-zero entries per row/column resulting in $O(NM)$ non-zero entries.

Individual

Sparse matrices are implemented natively in most programming environments, and its matrix-vector product can be calculated in $O(NM)$ operations. The Gershgorin eigenvalue bound (Eq 139) can be calculated directly in $O(NM)$ operations.

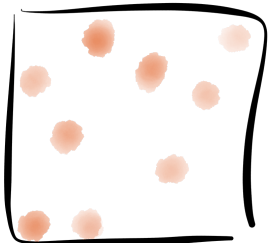


Figure 28: Illustration of a sparse matrix

Product

The matrix-vector product of a product of sparse matrices can also be efficiently evaluted in $O(NM)$:

$$\mathbf{S}_1\left(\mathbf{S}_2\vec{v}\right)$$

and a bound on the eigenvalues can be obtained via the product of the bounds on $\mathbf{S}_1$ and $\mathbf{S}_2$.

$$\lambda_{max}(\mathbf{S}_1\mathbf{S}_2) \leqslant \lambda_{max}(\mathbf{S}_1)\lambda_{max}(\mathbf{S}_2) \tag{152}$$

5.1.3 *Addition*

A GP with a sum of kernels $k = k_1 + k_2$ is equivalent to modelling the function as the sum of two independent functions, one modelled with k_1 and the other with k_2. This is illustrated in Figure 29. For two efficient kernels, the computational

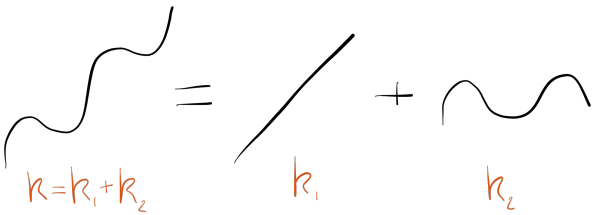


Figure 29: Additive kernel

cost of the matrix-vector product of their sum is the sum of the cost of the individual matrix-vector products, since vector multiplication distributes over matrix addition (as in Equation 153).

$$(\mathbf{K}_1 + \mathbf{K}_2)\,\vec{v} = \mathbf{K}_1\vec{v} + \mathbf{K}_2\vec{v} \tag{153}$$

A bound on the largest eigenvalue can be obtained by the following basic identity:

$$\lambda_{max}(\mathbf{K}_1 + \mathbf{K}_2) \leqslant \lambda_{max}(\mathbf{K}_1) + \lambda_{max}(\mathbf{K}_2) \tag{154}$$

5.2 GRAPHICAL MODEL KERNELS

Taking advantage of conditional independence structure in statistical models is one of the cornerstones of efficient inference. Gaussian Processes are no exception — kernels with graphical model structure are by far the most common type of kernels defined specifically for efficient inference[Csató and Opper, 2001, Sudderth et al., 2004, Weiss and Freeman, 2001]. Many of these were originally proposed as approximations to full Gram matrices, but it was shown in [Quiñonero Candela and Rasmussen, 2005] that it can be equally useful to consider these not as approximations, but an alternative choice of computationally efficient kernel. In this section we will review the meaning of graphical models in the context of Gaussian Processes, demonstrate a number of common fast kernels that can be described with graphical models and show how the M-efficiency of a kernel is defined by its graphical model.

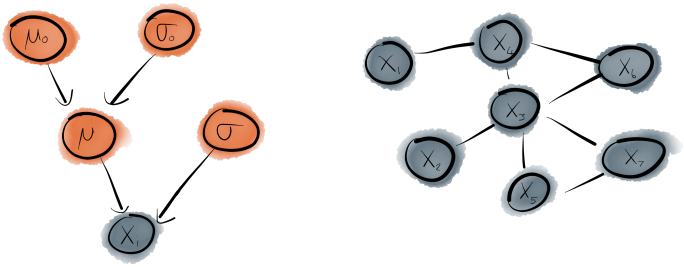
5.2.1 *Background*

Graphical models are used to encode the conditional independence structure of a problem and are critical for efficient inference in many models. The reader is assumed to be familiar with the notation for directed and undirected graphical models. See [Koller and Friedman, 2009] for a comprehensive review.

In parametric models, it is usually easy to understand what the graphical structure of a problem means. It illustrates that the datapoints are drawn from distributions that depend only on a subset of the total parameters and the parameters themselves may be drawn in a similarly structured fashion. A simple example is the model:

$$\mu \sim \mathcal{N}(\mu_0, \sigma_0) \tag{155}$$
$$x_i \sim \mathcal{N}(\mu, \sigma) \tag{156}$$

which is illustrated in Figure 30a. We can clearly see in the graph that x depends only on μ and σ (the dependence on μ_0 and σ_0 is mediated through μ).



(a) Parametric graphical model (b) Non-parametric graphical model

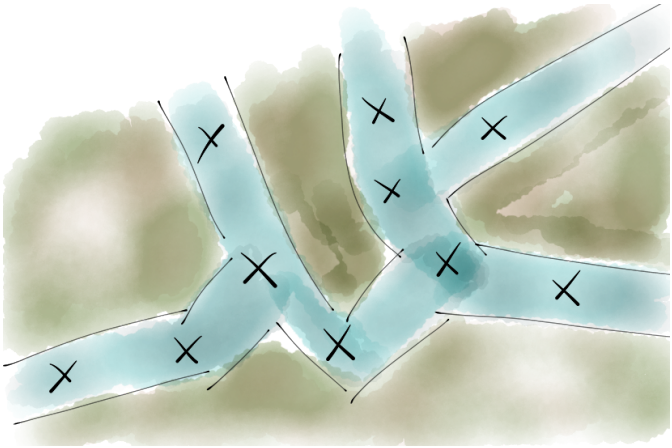
Figure 30: Example graphical models

However, in non-parametric models there are no parameters in the traditional sense. The only elements in the graphical model are the data points, as illustrated in Figure 30b. The generative interpretation is still correct — we can generate the datapoints by sampling from conditional distributions — but it's now far less intuitive. An alternative, more enlightening definition of what graphical models mean in the context of Gaussian Processes is based on the relationship of two points x_a and x_b that the graph defines as being conditionally independent of each other given a set of points $\mathcal{C}$.

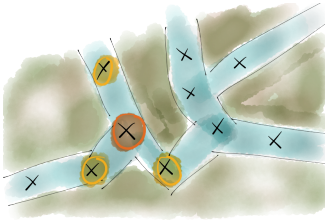
Definition 2. If $x_a \perp\!\!\!\perp x_b \mid \mathcal{C}$ then knowledge of the value of x_b would not change our prediction of the value of x_a, given the value of the points in $\mathcal{C}$. Equivalently, x_b contains no additional predictive information about x_a, above what is already contained in $\mathcal{C}$.

River buoy example

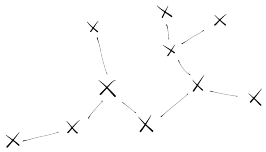
An example of a problem with conditional independence of data points is the river sensor network, which is illustrated in Figure 31. Deployed in the river network are a number of buoys that measure the flow rate of the river. The flow rate varies smoothly over the distance of the river but can be affected by local conditions. To predict the flow rate at a given buoy, we would only required the value of the buoys that are directly upstream and downstream from it (Figure 31b). This knowledge entails the graphical model overlayed in Figure 31c.



(a) A river network, with flow rate monitoring buoys at the X's.



(b) The prediction of the buoy highlighted in orange is conditionally independent of all other buoy measurements, given the measurements at the buoys in yellow.



(c) It is clear that the underlying graphical model is represented by the following graph.

Figure 31: River network example.

Inducing points

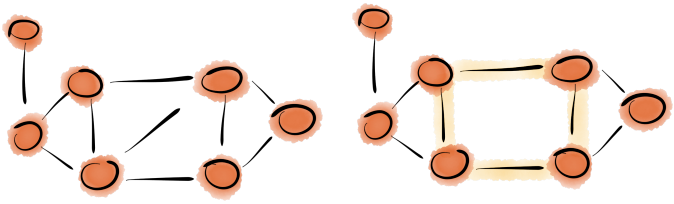
If some buoys in the network were broken, this wouldn't change the underlying structure of the problem, but we would no longer be able to observe the values at these points. When defining these graphs, all training and test datapoints must be part of the graph, however we can also include points that are neither in the training nor test set. These points are referred to as either inducing or latent points. In this case, these broken buoys would be inducing points, as we can not observe them and may not want to predict at them, yet they are necessary to describe the structure of the problem.

If the training and test points are independent given the latent points, then this graphical model bears a strong similarity to traditional parametric models.

Chordal graphs

An important set of undirected graphical models are *chordal graphs*, as these are the set of graphs for which efficient closed form inference has been proved to be possible[Koller and Friedman, 2009]. Chordal undirected graphical models can be translated into factor graphs without loops, and hence can be solved via belief propagation and similar algorithms.

Definition 3. A graph is chordal if every cycle of length > 3 has a chord (an edge between two points on the cycle that is not itself part of the cycle).



(a) Chordal graphical model

(b) Non-chordal graphical model with offending cycle highlighted

Figure 32: Example graphical models

During this section we will only be considering chordal graphs. The efficiency of methods on undirected graphical models are related to the size of the maximal clique in the graph, which is the largest fully connected component of the graph.

Gaussian Processes

A graphical model does not entirely define a Gaussian Process, it merely describes properties of it. There are always many (infinite in fact) Gaussian Processes that are consistent with a given graphical model. An undirected graphical model $\mathcal{G}$ can be represented by a symmetric binary matrix $\mathbf{G}$, where:

$$\mathbf{G}_{ij} = \begin{cases} 1 \text{ if } \{i,j\} \in \mathcal{G} \\ 0 \text{ if } \{i,j\} \notin \mathcal{G} \end{cases} \tag{157}$$

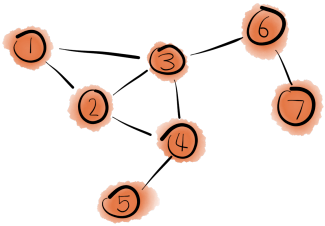
For a Gaussian distribution to obey the conditional independence relationships of a graphical model $\mathcal{G}$, it is equivalent to say:

$$\mathbf{K}^{-1} \circ \mathbf{G} = \mathbf{0} \tag{158}$$

ie the sparsity pattern of the inverse of the Gram matrix is given by $\mathbf{G}$. In order to fully specify the distribution for chordal graphs, it is sufficient to specify the covarinces for the corresponding non-zero elements of $\mathbf{G}$[Shental et al., 2008].

Using this property we can show that when $x_a \perp\!\!\!\perp x_b \parallel \mathcal{C}$,

$$\mathbf{K}_{ac} = \mathbf{K}_{ab}\mathbf{K}_b^{-1}\mathbf{K}_{bc} \tag{159}$$



(a) Graphical model $\mathcal{G}$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

(b) Matrix representation $\mathbf{G}$

Figure 33: A graphical model and its equivalent adjacency matrix

Proof.

$$\begin{bmatrix} K_a & K_{ab} & K_{aC} \\ K_{ba} & K_b & K_{bC} \\ K_{Ca} & K_{Cb} & K_C \end{bmatrix}^{-1} = \begin{bmatrix} \cdot & 0 & \cdot \\ 0 & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

Applying the block matrix inversion lemma to the above:

$$\begin{aligned} \begin{bmatrix} \cdot & 0 \\ 0 & \cdot \end{bmatrix} &= \begin{bmatrix} K_a & K_{ab} \\ K_{ba} & K_b \end{bmatrix} - \begin{bmatrix} K_{aC} & K_{bC} \end{bmatrix} K_C^{-1} \begin{bmatrix} K_{Ca} \\ K_{Cb} \end{bmatrix} \\ 0 &= K_{ab} - K_{aC} K_C^{-1} K_{Cb} \\ K_{ab} &= K_{aC} K_C^{-1} K_{Cb} \end{aligned}$$

□

5.2.2 *Definition*

A graphical model kernel is any kernel that has a graphical model that is not fully connected. To perform a matrix-vector multiplication we start by picking a set of separating variables such that $a \perp\!\!\!\perp c \mid b$.

$$\mathbf{K}\vec{v} = \begin{bmatrix} \mathbf{K}_a & \mathbf{K}_{ab} & \mathbf{K}_{ab}\mathbf{K}_b^{-1}\mathbf{K}_{bc} \\ \mathbf{K}_{ba} & \mathbf{K}_b & \mathbf{K}_{bc} \\ \mathbf{K}_{cb}\mathbf{K}_b^{-1}\mathbf{K}_{ba} & \mathbf{K}_{cb} & \mathbf{K}_c \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vec{v}_3 \end{bmatrix} \tag{160}$$

We perform:

$$\vec{k}_0 = \vec{v}_2 + \mathbf{K}_b^{-1}\mathbf{K}_{ba}\vec{v}_1 \tag{161}$$

$$\begin{bmatrix} \vec{k}_1 \\ \vec{k}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{K}_b & \mathbf{K}_{bc} \\ \mathbf{K}_{cb} & \mathbf{K}_c \end{bmatrix} \begin{bmatrix} \vec{k}_0 \\ \vec{v}_3 \end{bmatrix} \tag{162}$$

$$\mathbf{K}\vec{v} = \begin{bmatrix} \mathbf{K}_a\vec{v}_1 + \mathbf{K}_{ab}\mathbf{K}_b^{-1}\left(\vec{k}_1 - \mathbf{K}_{ba}\vec{v}_1\right) \\ \vec{k}_1 \\ \vec{k}_2 \end{bmatrix} \tag{163}$$

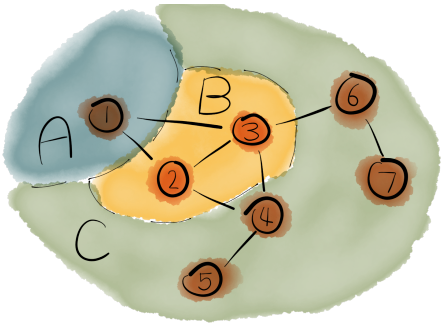


Figure 34: Separation of a graphical model into three sets of points: a, b and c.

Equations 161 and 163 can be calculated in $O(M_b^3) + O(M_a M_b)$ operations. Equation 162 is the same style matrix vector product on the remainder of the graph when the points in a are removed. This leads the overall computational requirements to be $O(NM_{max}^2)$, where M_{max} is the size of the maximal clique in the graph. This is not necessarily a tight upper bound, and some methods are faster when $\mathbf{K}_b^{-1}$ is correctly cached. For example, by selecting the cliques in a sensible order, a multiplication with a FITC Gram matrix can be performed in $O(NM)$ operations. If the graphical model contains inducing points, this same formula can be used by setting the $\vec{v}$ values for the inducing points to 0 and only evaluating the elements of $\mathbf{K}\vec{v}$ that correspond to non-inducing points.

Therefore a graphical model kernel with a graph of max clique size M is at least M^2-efficient.

5.2.3 *Example graphical models*

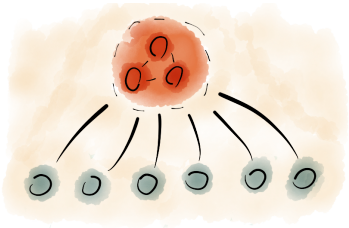
In this section we review a set of existing kernels that are members of this class of graphical model kernels.

Known feature space

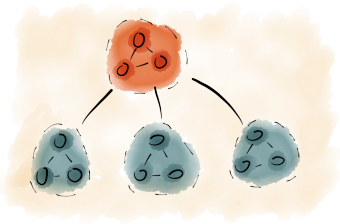
Any kernel with a known feature space whose dimensionality D is much smaller than N, can be implemented efficiently by observing that any D linearly independent observations fully specify the function. This means that all points are conditionally independent given any D linearly independent points. This includes, among many examples, the linear kernel as well as fixed basis function approximations such as Random Kitchen Sinks.

FITC

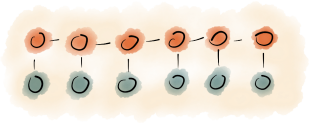
The Fully Independent Training Conditionals (FITC) approximation, introduced as Sparse Pseudo-input Gaussian Processes in [Snelson and Ghahramani, 2006], is likely the most common graphical model structured GP approximation currently in use. The graphical model for FITC shows that the training and test points are



(a) Graphical model for FITC and fixed rank methods



(b) Graphical model for PITC



(c) Graphical model for a 1st-order Markov chain

Figure 35: Illustrations of graphical models. Training points are shaded in blue and inducing points in red. Thick lines between groups of fully connected points indicate connections between all elements.

conditionally independent given an inducing set (Illustrated in Figure 35a).

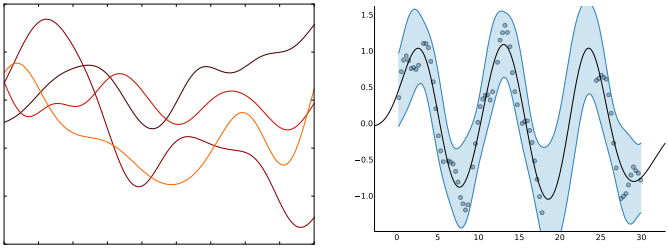
The distribution of a FITC GP is:

$$\mathbf{K}_{FITC} = \mathbf{K}_{xm}\mathbf{K}_m^{-1}\mathbf{K}_{mx} + \left(\mathbf{K} - \mathbf{K}_{xm}\mathbf{K}_m^{-1}\mathbf{K}_{mx}\right) \circ \mathbf{I}$$

Where $\circ$ is the element-wise matrix product. Figure 36 shows an illustration of this model applied to the Squared Exponential kernel.

PITC

The Partially Independent Training Conditionals (PITC) method was introduced in [Snelson and Ghahramani, 2007] as an extension to FITC that considered not just the global structure of the problem, but also the local structure. In this model, all points belong to one of D clusters $\mathcal{C}_1, ..., \mathcal{C}_D$. The conditional independence structure assumption for PITC is that a point x_i in cluster $\mathcal{C}_j$ is independent of all points $x_k \notin \mathcal{C}_j$ given $\mathcal{X}_m$. Instead of a point being independent of all other points given the inducing set (FITC), it is independent of all points that are not in its

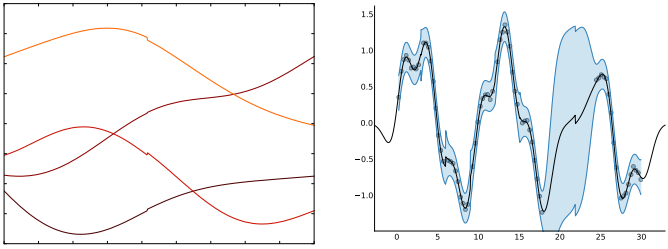


(a) Prior samples (b) Posterior distribution

Figure 36: Prior samples and posterior distribution for the RBF-PITC kernel

cluster, given the inducing set. This is illustrated in Figure 37.

$$\mathbf{K}_{PITC} = \mathbf{K}_{xm}\mathbf{K}_m^{-1}\mathbf{K}_{mx} + \left(\mathbf{K} - \mathbf{K}_{xm}\mathbf{K}_m^{-1}\mathbf{K}_{mx}\right) \circ (\mathbf{I} \otimes \mathbf{1}) \tag{164}$$



(a) Prior samples (b) Posterior distribution

Figure 37: Prior samples and posterior distribution for the SE-PITC kernel

Markov

A common assumption in time series models is that the data is Markovian — that the prediction at a given point is independent of all observations given the M observations either side of that point. A 1st-order Markov graphical model is illustrated in Figure 35c.

5.3 RANDOM PARTITION KERNELS

In this section we show how to derive a kernel from a random partition, and a method to create an M-efficient approximation to the kernel. Random partitions can be thought of as randomized clusterings of a dataset. Clustering can be a useful means to specify a kernel as it is a task which is usually quite intuitive, and for which there are many efficient algorithms. There has recently been interest in using K-means and other clustering algorithms to generate features for difficult

classification tasks[Coates and Ng, 2011, 2012] which bears similarity to this work, but approached from a non-kernelized, non-probabilistic perspective.

5.3.1 *Background*

Clustering

Clustering has a long history in machine learning and statistics precisely because it as a very simple and intuitive task. Originally, clustering methods were designed to find "the best" partition of a datset. More recent probabilistic methods allow uncertainty and instead learn a distribution on what the clusterings of the data might be. These distributions are known as partition distributions, and samples from them are random partitions.

A cluster C in a dataset $\mathcal{D}$ is a non-empty subset of $\mathcal{D}$. A partition of $\mathcal{D}$ is the segmentation of $\mathcal{D}$ into one or more non-overlapping clusters $\varrho = \{C_1, ..., C_j\}$. ie:

$$C_i \cap C_j = \varnothing \tag{165}$$

$$\bigcup_i C_i = \mathcal{D} \tag{166}$$

The following is an example dataset and an example partition of that dataset:

$$\mathcal{D} = \{a, b, c, d, e, f\} \tag{167}$$

$$\varrho = \{\{a, e\}, \{c\}, \{d, b, f\}\} \tag{168}$$

A *random partition* of $\mathcal{D}$ is a sample from a partition distribution $\mathcal{P}$. This distribution is a discrete pdf that represents how likely a given clustering is. We will use the notation $\varrho(a)$ to indicate the cluster that the partition ϱ assigns to point a.

Random partitions have been studied extensively in the field of non-parametric Bayesian statistics. Well-known Bayesian models on random partitions are the Chinese Restaurant Process [Aldous, 1985], the Pitman-Yor Process [Pitman and Yor, 1997] and the Mondrian Process [Roy and Teh, 2009], while many other combinatorial models can be used to define random partitions.

Defining distributions

There are many ways of defining a probability distribution: for example, by specifying the probability density function, the cumulative density function or the moment generating function. An alternative way to define a probability distribution is by defining a random program.

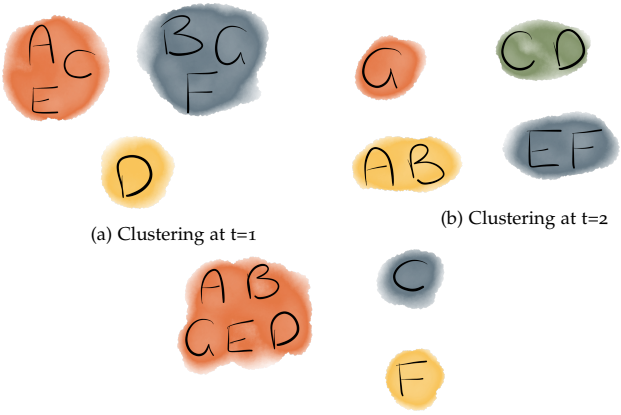
Definition 4. Random Program.
A random program is a program that takes a source of randomness $U_1, U_2, \cdots$ as input and returns an output in a set $\mathcal{S}$.

Any program of this form can be viewed as a program that generates samples from some distribution on $\mathcal{S}$. This should not be unfamiliar: any time a program is written to sample from a specific distribution, it is a random program. However, the converse is also true: any random program is a sampler for some distribution, and thus the random program defines a distribution. This correspondence is not 1-to-1: two programs may define the same distribution, but importantly every computable random program defines a distribution on $\mathcal{S}$. Therefore, in order to define a random partition of size N, it is sufficient to define a random program where $\mathcal{S}$ is the set of partitions of size N.

5.3.2 *Definition*

The Random Partition Kernel can be described succintly using a metaphor in the tradition of the CRP [Aldous, 1985] and IBP[Griffiths and Ghahramani, 2011].

We consider The Colonel, who is the host of a cocktail party. He needs to determine the strength of the affinity between two of his guests, Alice and Bob. Neither Alice and Bob, nor the other guests, must suspect the Colonel's intentions, so he is only able to do so through surreptuous observation. At the beginning of the evening, his N guests naturally form into different groups to have conversations. As the evening progresses, these groups evolve as people leave and join different conversations. At m opportunities during the course of the evening, our host notes whether Alice and Bob are together in the same conversation. Figure 38 illustrates the clusterings at different observation times.



(a) Clustering at t=1

(b) Clustering at t=2

(c) Clustering at t=3

Figure 38: Illustration of the cocktail party at different times throughout the evening

As the Colonel farewells his guests, he has a very good idea of Alice and Bob's affinity for one another, based on the number of conversations they participated in together.

A partition distribution naturally leads to a kernel in the following way:

Definition 5. Given a partition distribution $\mathcal{P}$, we define the kernel

$$k_{\mathcal{P}}(a,b) = \mathbb{E}\left[I\left[\varrho(a) = \varrho(b)\right]\right]_{\varrho\sim\mathcal{P}}$$

to be the random partition kernel induced by $\mathcal{P}$, where I is the indicator function.

That is, the kernel is defined to be the fraction of the time that two points are assigned to the same cluster.

Lemma 5.3.1. *$k_{\mathcal{P}}(a,b)$ constitutes a valid PSD kernel.*

Proof. First we define:

$$k_{\varrho}(a,b) = I\left[\varrho(a) = \varrho(b)\right]$$

To prove that $k_{\mathcal{P}}$ is PSD, we decompose the expectation into the limit of a summation and show that the individual terms are PSD.

$$k_{\mathcal{P}}(a,b) = \mathbb{E}\left[I\left[\varrho(a) = \varrho(b)\right]\right]_{\varrho\sim\mathcal{P}} \tag{169}$$

$$= \lim_{n\to\infty} \frac{1}{n} \sum_{\varrho\sim\mathcal{P}}^{n} I\left[\varrho(a) = \varrho(b)\right] \tag{170}$$

$$= \lim_{n\to\infty} \frac{1}{n} \sum_{\varrho\sim\mathcal{P}}^{n} k_{\varrho}(a,b) \tag{171}$$

For any dataset of size N, the kernel matrix for k_{ϱ} will be an $N \times N$ matrix that can be permuted into a block diagonal matrix of the following form:

$$\mathbf{Z}\mathbf{K}_{\varrho}\mathbf{Z}^{\intercal} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

where $\mathbf{1}$ is a matrix with all entries equal to 1, $\mathbf{0}$ is a matrix will all entries 0 and $\mathbf{Z}$ is a permutation matrix. Each $\mathbf{1}$ matrix represents a single cluster.

From this we can conclude that $\mathbf{Z}\mathbf{K}_{\varrho}\mathbf{Z}^{\intercal}$ is PSD, as it is a block matrix of PSD matrices. Further, since a permutation does not affect the eigenvalues of a matrix, $\mathbf{K}_{\varrho}$ is PSD. Since $\mathbf{K}_{\varrho}$ is PSD for any dataset, k_{ϱ} must be a valid kernel. Finally, since a linear combination of kernels with positive coefficients is also a valid kernel, $k_{\mathcal{P}}$ is a valid kernel.

□

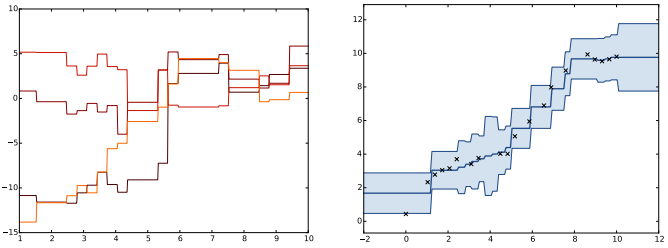
m-approximate kernel

This structure allows for a simple approximation scheme that only requires the ability to sample from the distribution $\mathcal{P}$.

Definition 6. The m-approximate Random Partition Kernel is the fraction of times that ϱ assigns a and b to the same cluster over m samples.

$$k_{\mathcal{P}}(a,b) \approx \tilde{k}_{\mathcal{P}}(a,b) = \frac{1}{m} \sum_{\varrho \sim \mathcal{P}}^{m} k_{\varrho}(a,b)$$

This m-approximate partition kernel is m-efficient, and defines a distribution over piecewise constant functions. Samples from the prior and the posterior distribution of a random partition kernel are illustrated in Figure 39 (the particular random partition is the Fast Cluster partition, which is described in Section 5.3.3). We can clearly see here that it defines a distribution over *piece-wise constant functions*. It can be seen in the samples from the prior (Figure 41b) that a given random partition kernel is not a distribution over *all* piece-wise constant functions, but rather there are a finite number of jump locations that are determined by the clustering algorithm.



(a) Prior samples

(b) Posterior distribution

Figure 39: Prior samples and posterior distribution for FastCluster kernel

Convergence of m-approximate kernel

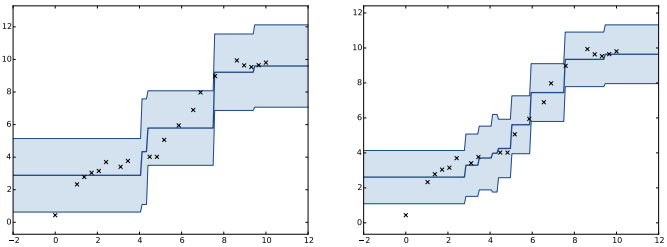
Lemma 5.3.2. *If the samples from $\mathcal{P}$ are independent then the bound on the variance of the approximation to $k_{\mathcal{P}}(a,b)$ is $O\left(\frac{1}{m}\right)$.*

Proof. If ϱ are independent samples from $\mathcal{P}$, then

$$k_{\varrho}(a,b) \sim \textit{Bernoulli}\left(k_{\mathcal{P}}(a,b)\right)$$

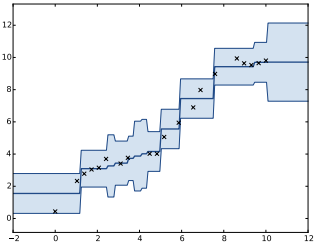
and $\tilde{k}_{\mathcal{P}}(a,b)$ is the maximum likelihood estimator for $k_{\mathcal{P}}(a,b)$. The variance of the ML estimator for a Bernoulli is bounded by $\frac{1}{4m}$. □

In Figure 40 we see how the posterior converges as we increase the number of samples M.

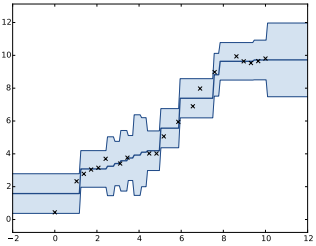


(a) M=3

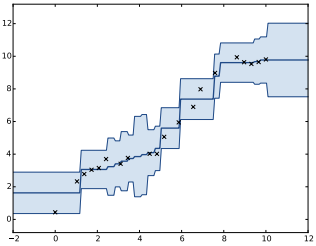
(b) M=8



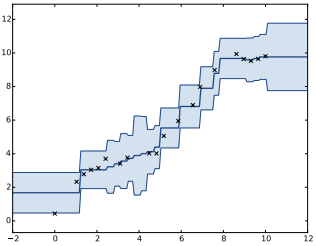
(c) M=13



(d) M=18



(e) M=23



(f) M=28

Figure 40: Posterior converging with increasing M

Implementation

A partition matrix can be implemented as a product of sparse matrices $\mathbf{S}\mathbf{S}^\intercal$ where

$$\mathbf{S}_{ij} = \begin{cases} \frac{1}{\sqrt{\|\varrho(j)\|}} & \text{if } \varrho(j) = i \\ 0 & \text{otherwise} \end{cases} \tag{172}$$

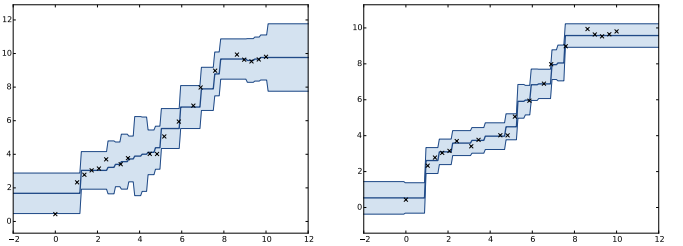
Where $\|\varrho(j)\|$ is the size of cluster $\varrho(j)$.

Optimizing sample weights

A further extension to this kernel is to relax our interpretation of the random partition kernel and learn cluster specific weights $w_1, \ldots, w_m$ to be set when optimizing the GP. This more general formulation gives us the kernel form:

$$\tilde{k}_{\mathcal{P}}^{opt}(a,b) = \sum_{i=1}^{m} w_i k_{Q_i}(a,b) \tag{173}$$

In Figure 41 we see the results of a standard and optimized weight random partition kernel side by side.



(a) FastCluster without weights (b) FastCluster with optimized weights

Figure 41: Effect of cluster specific weight optimization

5.3.3 *Random partitions*

To demonstrate the potential of this method, we first show how to obtain random partitions from a large class of existing algorithms. Then we focus on a particular example, the *Fast Cluster Kernel*, to illustrated properties of kernels derived from this method.

Stochastic clustering algorithms

Any standard clustering algorithm (K-means [MacKay, 2003a] and DBSCAN [Ester et al., 1996] being just a couple of examples) generates a partition for a given dataset. Adding any element of randomness to the algorithms (if it does not already exist) results in a stochastic clustering algorithm. Elements of the clustering algorithm that can be randomized include:

1. Initializations
2. Number of clusters
3. Subsets of the data (Bagging)
4. Subset of the features
5. Projections of the features

The output for a stochastic clustering algorithm is a random partition. As a simple concrete example, the output of K-means with random restarts returns a random partition.

Ensembled tree methods

A large class of random partitions can be derived from *ensembled tree methods*. We use this term to refer to any method whose output is a collection of random trees. By far the most well known of these are Random Forest [Breiman, 2001] and Boosted Decision Trees [Freund and Schapire, 1997], though it also includes Bayesian Additive Regression Trees [Chipman et al., 2010], Gradient Boosting Machines with decision trees [Friedman, 2000] and many others.

As a tree defines a hierarchical partition, it can be converted to a partition by randomly or deterministically choosing a height of the tree and taking the partition entailed by the tree at that height.

Example: Fast cluster kernel

To illustrate some of the properties of the Random Partition Kernel, we propose the following random partition, which we call "FastCluster". The algorithm is as follows: First, sample a subset of the dimensions which will be used to measure distances for the partition. Then, select a random number of cluster centres M uniformly from $\{2, 4, ..., N\}$ and randomly sample M unique cluster centers from $\mathbf{X}$. Assign every point to the cluster associated with its nearest centre, measured only on the subsampled dimensions. The pseudocode is shown in Figure 42.

$\vec{d} \sim \text{Bernoulli}(.5, D)$
$M \sim \text{Sample}([2, 4, ..., N], 1)$
$\mathcal{C} \sim \text{Sample}([1, ..., N], s)$
for $a \in \mathcal{D}$ **do**
    $\varrho(a) = argmin_{c \in \mathcal{C}} \left(\|(\mathbf{X}_c - \mathbf{X}_a) \odot \vec{d}\| \right)$

Figure 42: Algorithm for Fast Cluster

5.4 SPARSE AND LOW-RANK KERNELS

The final class of efficient kernels we propose are sparse and low-rank kernels. These are kernels for which the Gram matrix is the sum of a sparse and a low-rank matrix. This class is particularly suited to the framework of Chapter 4. While random partition kernels have sparse known features and thus can be solved efficiently with iterative methods in the primal and graphical model kernels can be solved efficiently in the dual with factorization via belief propagation, sparse and low-rank kernels are uniquely suited to iterative solvers in the dual.

5.4.1 *Background*

Sparse and low-rank matrices are tied very much to the idea of the global and local modelling of functions.

Local clusters

There have been a number of approximations and kernels that are based around the idea that there is distinct global and local structure in functions. That is, that a function can be modelled well by splitting interactions into local effects, that only occur between points that are close, and global effects, that can be summarized by a number of basis functions much smaller than the total number of points. PITC, which we have already seen in Section 5.2.3, was one of the first models to be based on this assumption. The definition of its Gram matrix in Equation 164 reveals it to be a low-rank and sparse matrix, which shows there is some overlap between this class and graphical model kernels. PITC suffers from some limitations in modelling short term interactions, as it requires the data to be segmented into clusters for the local function modelling. This is especially problematic in cases where there are no obvious clusters of the data, such as time-series modelling.

Compactly supported kernels

Compactly supported kernels[Zhu, 2012] are those such that for a given a, $k(a, b)$ is only non-zero for b in a finite region of space. For the stationary kernels that we consider, this means that k is only non-zero for points that are within a certain distance C of each other. This property means that the resulting Gram matrix is sparse. The idea of constructing an efficient low-rank and sparse kernel from a compact kernel was investigated in [Vanhatalo and Vehtari, 2012], which proposes a sum of a FITC kernel and a compactly support kernel. One of the limitations of this work is the reliance on a sparse factorization method for calculations, rather than an iterative framework.

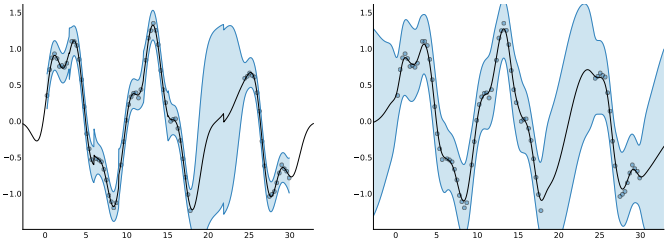
Sparse factorization methods — such as the sparse Cholesky — are factorization methods that take advantage of the structure of a sparse matrix by calculating only the non-zero elements of the resulting Cholesky factors. This means that the method is often substantially faster than a full factorization for sparse matrices. However, the resulting sparsity pattern of the Cholesky matrix is less sparse than that of the original matrix. In some instances, a sparse matrix may lead to an entirely dense Cholesky factor, in which case the required number of operations is $O(N^3)$. While we have already made the case that the framework of Chapter 4 is generally preferable to factorization due to the incorporation of error tolerances, in this specific example the worst case bound of the iterative method $O(N^2M)$, where the full $I = N$ iterations are required, can actually be substantially faster than the worst case bound of $O(N^3)$ of sparse factorization.

5.4.2 *Definition*

The kernel is defined as the sum of a low-rank kernel and a sparse kernel.

$$k(a,b) = k_{low-rank}(a,b) + k_{sparse}(a,b) \tag{174}$$

Two examples are shown in Figure 43



(a) PITC kernel (b) FITC + compact support kernel

Figure 43: Posterior distributions for different low-rank and sparse kernels

There are two practical requirements for using compact support kernels in an efficient scheme such as this. Firstly, the compactly supported kernels we will use have a cut-off parameter, C, that defines the distance past which points are uncorrelated. To construct an M-efficient kernel with tuneable efficiency a value of C must be chosen such that the number of non-zero entries in $\mathbf{K}$ is $O(MN)$. Secondly, we need to identify the non-zero entries of $\mathbf{K}$ efficiently — it is unacceptable to simply generate the full matrix to find which values exceed the threshold, as this would require N^2 operations.

The first problem is a *quantile estimation* problem; we need to identify the $\frac{M}{N}$th percentile of the elements of the distance matrix. This quantity does not need to be identified to a high degree of accuracy, only sufficiently so that it doesn't select so few points as to be a poor approximation, nor so many that the problem becomes computationally intractable. This can be achieved by sampling MN entries of the distance matrix at random and selecting the empirical percentile. The second problem, finding points within a certain distance of a given point, is a common issue in information retrieval and there are a number of techniques to solve this problem: k-d trees[**?**] and ball trees[**?**], for example, can be constructed in $O(DN \log N)$ and require $O(N)$ storage.

5.4.3 *Compactly supported kernels*

There are a number of different classes of sparse kernels [Zhu, 2012, Fasshauer], so in this section we will review one of the most commonly used classes, a previously proposed extension, and present a new efficient class based on this.

Wendland compact support

The Wendland family[Wendland, 1995] of compactly supported kernels are a family of kernels that define a prior over piece-wise polynomial functions. The family can be generated by the repeated blurring of a delta spike kernel[MacKay, 1998, Morse et al., 2005]. The family is parameterized by a polynomial order p, a decay term η and a cut-off distance C. The kernel is defined as follows:

$$k_{\eta,p}(a,b) = \left(1 - \frac{\|a-b\|}{C}\right)_+^{\eta+d} \psi_{\eta,p}\left(\frac{\|a-b\|}{C}\right) \tag{175}$$

Where:

$$\psi_{\eta,p}(r) = \sum_{i=0}^{p} \beta_{i,p} r^i \psi_{\eta+2k-i}(r) \tag{176}$$

$$\beta_{0,0} = 1 \tag{177}$$

$$\beta_{j,p+1} = \sum_{i=j-1}^{p} \beta_{i,p} \frac{[i+1]_{i-j+1}}{(\eta+2p-i+1)_{i-j+2}} \tag{178}$$

and $C > 0$ and $\eta \geqslant \frac{d+1}{2}$.

This kernel is compactly supported since the value is 0 for any points for which $\|a-b\| > C$. Increasing the polynomial order of the family results in increasingly smooth functions: samples from the prior of the Wendland order 0 kernel are shown in Figure 44a and the posterior distribution for a simple dataset is shown in Figure 44b. The smoother order 3 kernel is similarly illustrated in Figure 44c and Figure 44d.

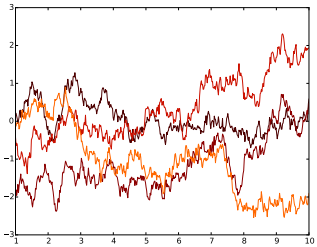
From the figures we can see that while the higher order kernel results in a smoother function, the piecewise polynomial nature of the function leads to a strong chance of overfitting.

Compactly support RBF

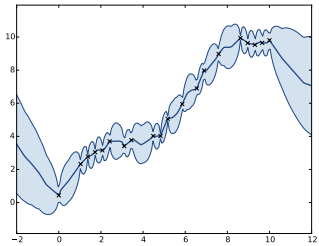
A number of authors use a compactly supported radial basis function (RBF) kernel (CS-RBF) by multiplying an RBF kernel with the 0th order Wendland kernel[Hamers et al., 2002, Genton, 2002]. The RBF used in these cases is a Squared Exponential.

$$k_{CS-RBF}(a,b) = k_{RBF}(a,b) k_{Wendland}(a,b) \tag{179}$$

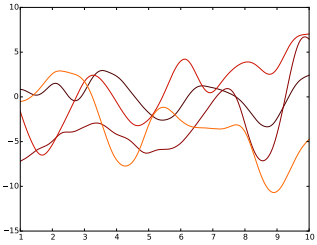
Figure 45 shows a compactly supported RBF using both the 0th and 3rd order Wendlands. While they look very similar to the Wendland diagrams in Figure 45, there are subtle differences, especially noticable in the variance.



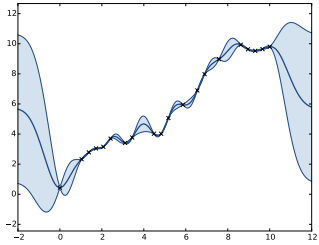
(a) Prior samples (oth order)



(b) Posterior distribution (oth order)



(c) Prior samples (3rd order)



(d) Posterior distribution (3rd order)

Figure 44: Prior samples and posterior distribution of the oth order Wendland kernel

Exponentiated Wendland

Based on the Wendland family, we propose a new class of kernels — the exponentiated Wendland family.

$$k(a,b) = e^{k_{wendland}(a,b)} \tag{180}$$

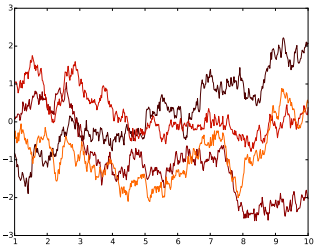
The Gram matrix that is generated by this kernel can be expressed as:

$$\mathbf{K} = \vec{1}\vec{1}^{\intercal} + \mathbf{S} \tag{181}$$

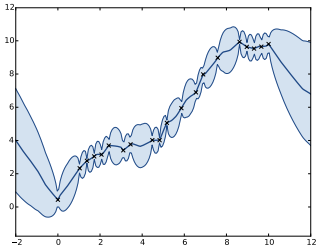
where:

$$\mathbf{S}_{ij} = \begin{cases} 0 & \text{if } k(x_i, x_j) = 0 \\ e^{k(x_i,x_j)} - 1 & \text{otherwise} \end{cases} \tag{182}$$

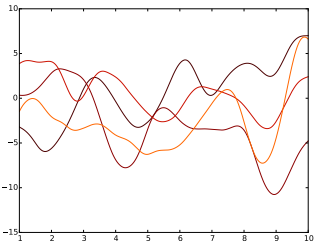
While this is no longer compact, it can be expressed as the sum of a sparse and low rank matrix, which means that it is still efficient. The oth and 3rd members of the family are shown in Figure 46



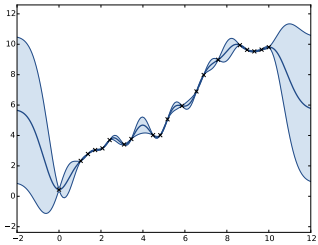
(a) Prior samples (0th order)



(b) Posterior distribution (0th order)

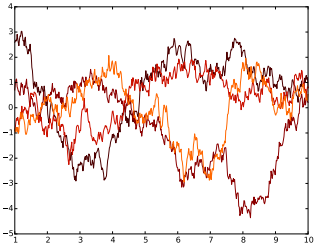


(c) Prior samples (3rd order)

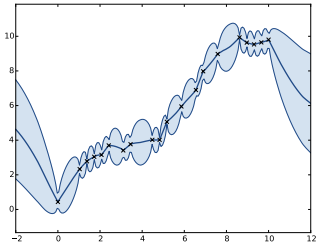


(d) Posterior distribution (3rd order)

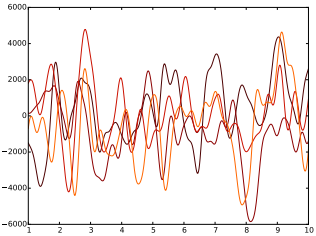
Figure 45: Prior samples and posterior distribution of the 0th order Wendland kernel



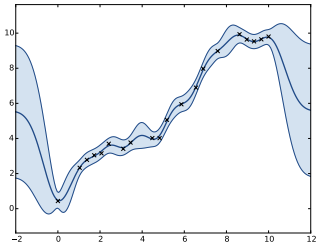
(a) Exponentiated compact (order 0)



(b) Exponentiated compact (order 0)



(c) Exponentiated compact (order 3)



(d) Exponentiated compact (order 3)

Figure 46: Prior samples from exponentiated compact kernel

5.5 EXPERIMENTS

In this section we demonstrate the performance of some example kernels from the proposed classes, and show that posterior variances in these kernels can be refined faster than $O(NMI)$ by using conjugate directions with a specific set of basis vectors.

5.5.1 *A return to conjugate directions*

A remaining limitation of the framework of Chapter 4 is that refining variance estimates uses conjugate gradient, at a cost of $O(MN)$ per iteration for a test data point. If the computational cost is affordable it is likely the best choice, however at test time it would not be unusual for $O(MN)$ to be a prohibitive cost. In this case, we can still refine variances at a cost cheaper than a single iteration of conjugate gradient. Recall that conjugate directions allows us to select any set of linearly independent basis vectors (Section 4.2.1). The rows of $\mathbf{K}$ form such a basis and in the case of these fast kernels the matrix-vector product between $\mathbf{K}$ and a row of $\mathbf{K}$ can be calculated in $O(M^2)$.

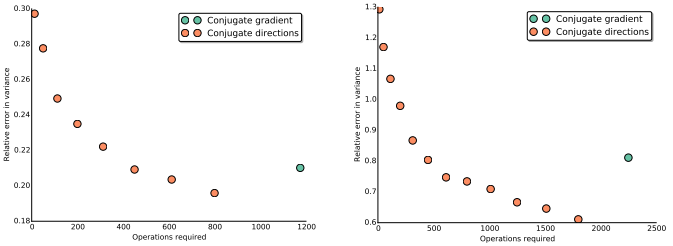
This can be done in the following way: at each iteration i of conjugate directions, the search direction is a linear combination of the previous i basis vectors. If the unit vectors are m-efficient, then the ith search direction will be im-efficient. This means at each iteration, the cost is bounded by $O(M^2 i)$, and the total cost up to the ith iteration bounded by $O(M^2 i^2)$. While the bounds under conjugate directions do not necessarily decrease monotonically, as in conjugate gradient, we can take a running lower bound.

In this case, for small number of iterations (which is likely all that we can afford), conjugate directions is actually substantially more efficient. We illustrated the difference between the two methods in Figure 47. The basis vectors for conjugate directions were selected for each test point by locating the 50 nearest training points and sampling uniformly from within that set. In nearly all of the datasets tested, conjugate directions was able to achieve a tighter variance bound in fewer operations than a single iteration of conjugate gradient.

While these experiments prove that conjugate directions is a viable alternative for varaince refinement for fast kernels, it is likely that further gains can be acheived by judicious selection of the basis vectors used.

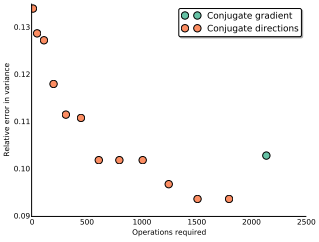
5.5.2 *Kernel comparison*

To support the validity of the classes of kernels suggested in this chapter, we present the performance of a selection of example kernels as speed-accuracy trade-off plots (Figure 48 - 55), as suggested in [Chalupka, Krzysztof and Williams, Christopher KI and Iain Murray, 2013].The axis on the plots measure computation in number of operations and predictive performance as log predictive likelihood, the log probability of independently predicting the points in a test set. In these plots the top left corner represents the ideal location, representing high accuracy and low computational requirements. Plotted for reference are the SE-RBF kernel

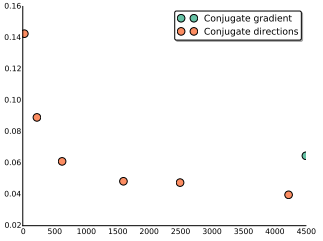


(a) MPG dataset

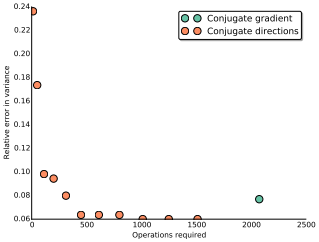
(b) Boston housing dataset



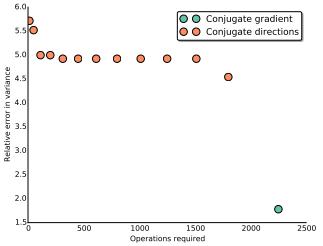
(c) Australian beer dataset



(d) Daily min temp dataset



(e) Prod of H_2SO_4 dataset



(f) Sea levels

Figure 47: Average relative error in variance estimate vs number of computations used to refine variance

solved with the framework of Chapter 3 and Cholesky decomposition. For each kernel we show the results for a 10-efficient and 25-efficient version of the kernel.

These plots show two baselines: the squared exponential RBF kernel solved with the Cholesky method and the iterative method. They have the same predictive likelihood, but in almost all cases, the iterative method is an order of magnitude faster than the Cholesky. Further, we can see that the M-efficient kernels generally achieve at least another order of magnitude speed-up over the iterative squared exponential kernel, in many cases with little loss in predictive accuracy. In more than half the cases, at least one of the fast kernels performance is an order of magnitude better, in terms of predictive log-likelihood. In half of the datasets (Figures 48,49,53,55) the Exponentiated Wendland kernel achieved the

highest predictive accuracy, which suggests it is worthy of further investigation. Thus in almost all datasets shown, judicious choice of an M-efficient kernel and an iterative solver allows for many orders of magnitude speed improvements over a Cholesky solver and a squared exponential kernel, with no less in predictive performance and in some cases, substantial improvement in predictive performance.

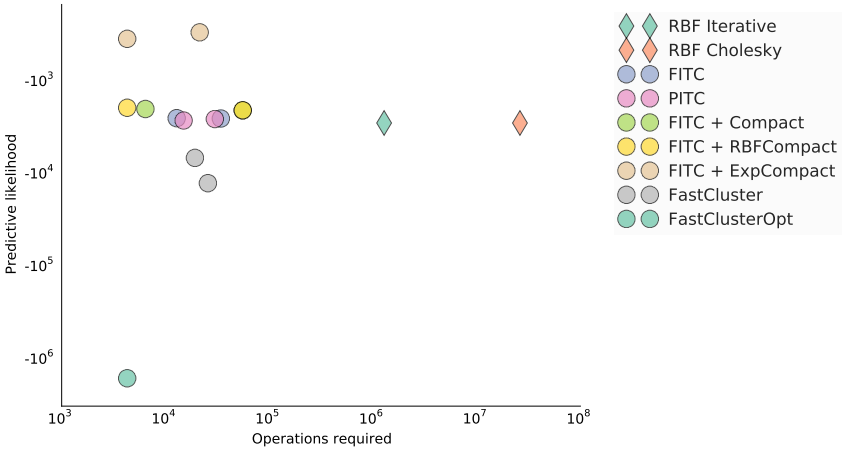


Figure 48: Australian beer dataset

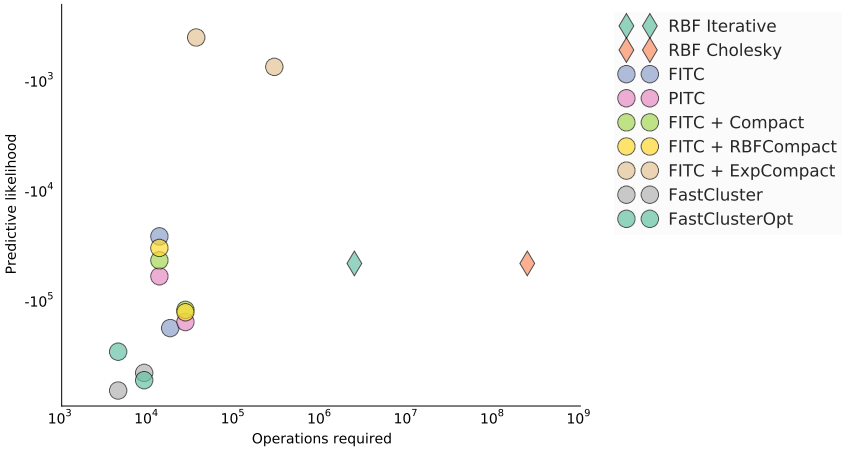


Figure 49: Quebec births dataset

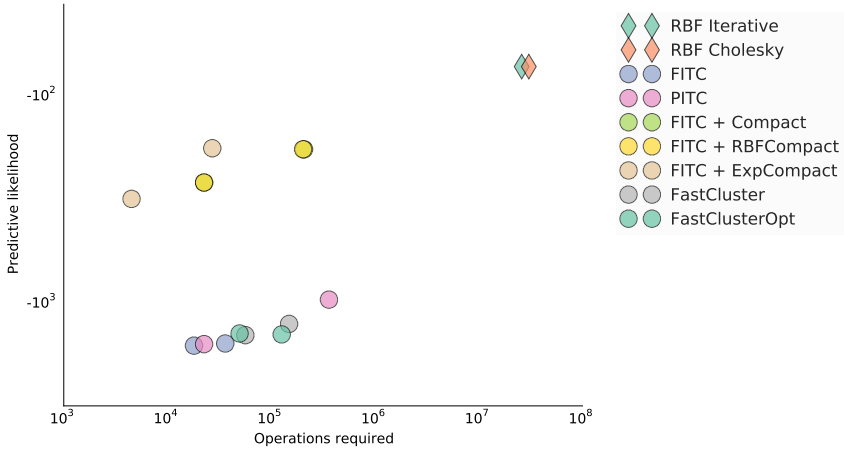


Figure 50: Sea levels dataset

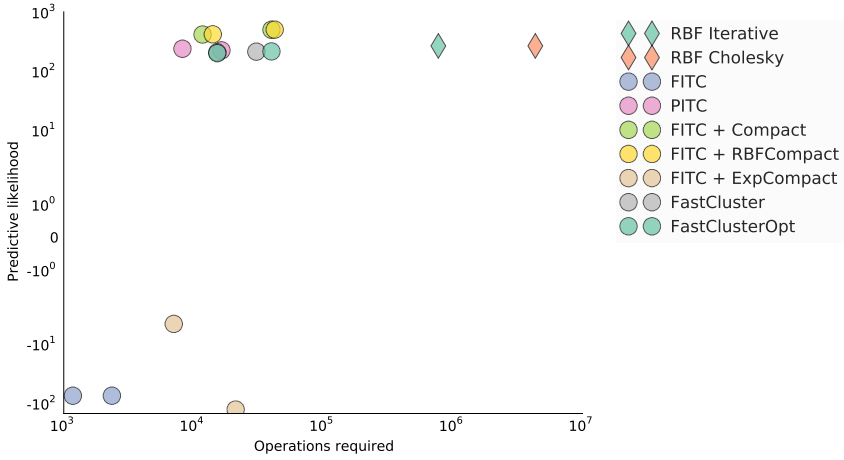


Figure 51: MPG dataset

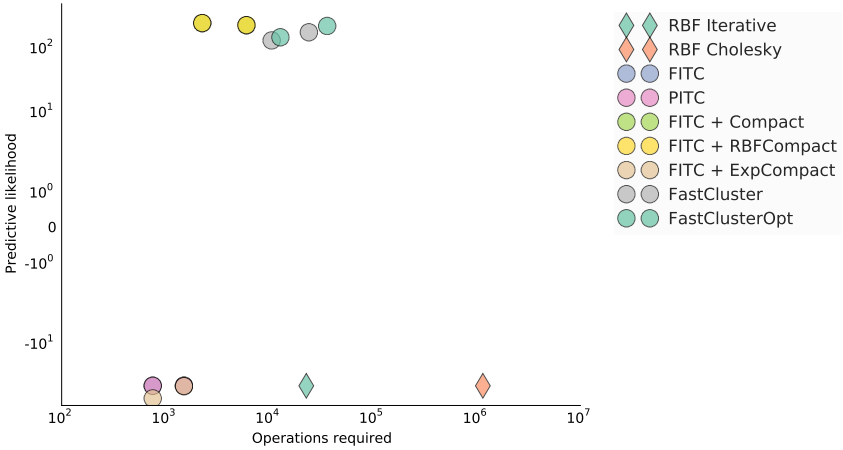


Figure 52: Bodyfat dataset

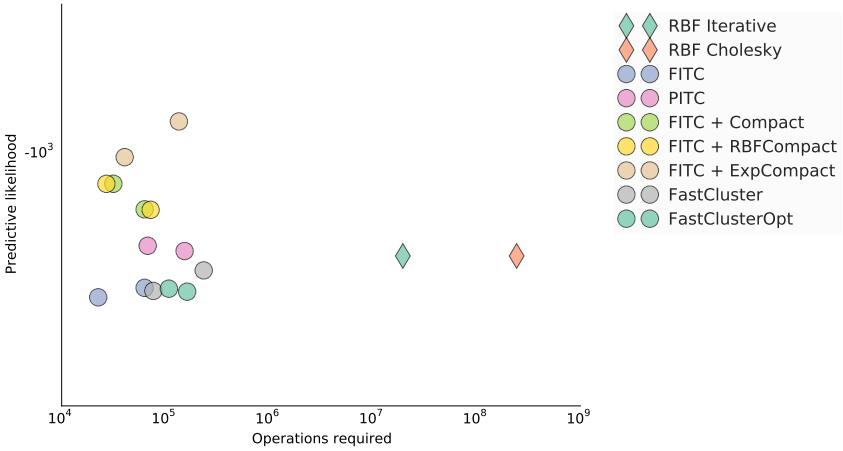


Figure 53: Daily min temp dataset

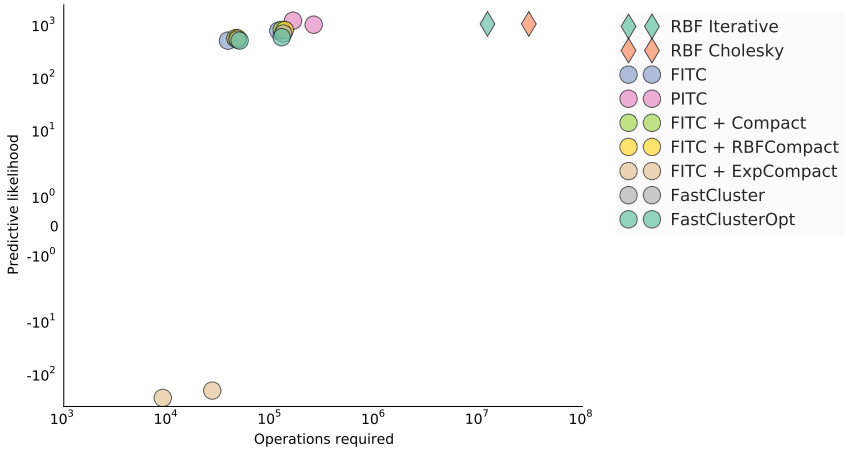


Figure 54: Boston housing

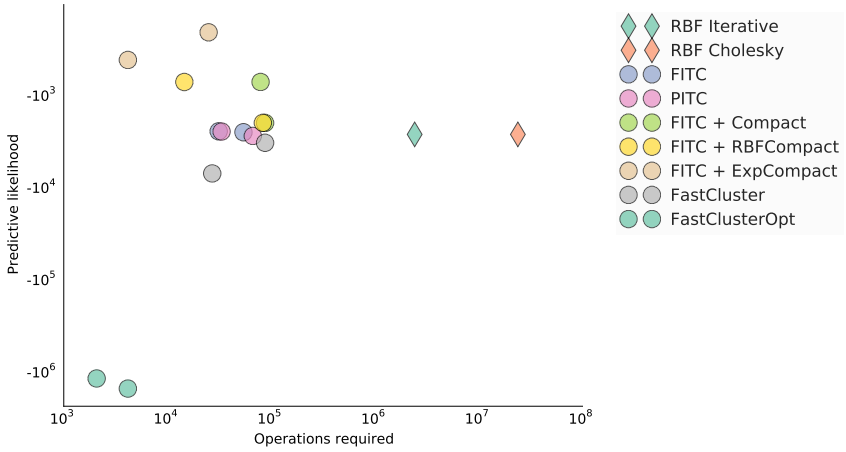


Figure 55: Production of H_2SO_4

6

EFFECTIVE OPTIMIZATION OF GPS: CONSTANT FACTOR IMPROVEMENTS

Climb if you will, but remember that courage and strength are naught without prudence, and that a momentary negligence may destroy the happiness of a lifetime. Do nothing in haste, look well to each step, and from the beginning think what may be the end.

— Edward Whymper, Scrambles amongst the Alps

Up until this chapter, we have been largely concerned with GPs that have a given kernel and hyper-parameters. The last element of a framework of Gaussian Processes for machine learning is a method for kernel and hyper-parameter selection. For a fixed set of kernel hyper-parameters Gaussian Process regression can be viewed as a kernel smoothing technique, or linear regression with a fixed set of basis functions, which does not fully capture what many would consider to be the full scope of "learning"[MacKay, 2003b]. Kernel and hyper-parameter selection affords us the chance to learn deeper structural properties of the function that we are modelling. This can occur within different frameworks: there are examples of learning kernels through exploring compositional search trees [Lloyd et al., 2014, Duvenaud et al., 2013], others that learn the spectral density of a kernel[Wilson and Adams, 2013] and a framework to optimize kernel hyper-parameters through a stochastic programming formulation[Anitescu et al., 2012]. In the overwhelming majority of cases though, the kernel is learnt through the gradient-based optimization of a set of kernel hyper-parameters for a fixed kernel form that is chosen by a human expert. This is the framework for which we explore the effective implementation.

When optimising kernel hyper-parameters via gradient-based optimisation, there are two choices that have an impact on the overall computational requirements. The first is the optimisation objective, the quantity with respect to which the hyper-parameters are being optimized. Traditionally this is the log-marginal likelihood of the data, but we propose several alternatives in Section that can be evaluated much more efficiently. The second is the use of effective pre-conditioning. As the optimization space for kernel hyper-parameters is not quadratic, as was the optimization space in Section 4.7, we require a more general idea of pre-conditioning, that we put forward in Section . We then go on to propose a number of efficient pre-conditioners for Gaussian Processes in Section 6.2.2.

6.1 THE OPTIMIZATION OBJECTIVE

GPs have traditionally been optimized by calculating the derivatives of the log-likelihood with respect to the kernel hyper-parameters $\vec{\theta}$ and then performing a gradient optimization method. In practice this is usually either non-linear conjugate gradient, L-BFGS or a truncated Newton method[Authors, 2014][Rasmussen and Nickisch, 2010].

Alongside the traditional method of GP optimization we propose two objective functions — one derived from a variational framework and another from validation error. Like having a choice of kernels, it is beneficial to having a choice of objective functions for optimization, due to the different goals of regression. For example, in some cases the only relevant goal may be to return the best predictions on a given test set, while in others the goal may be to model the function for different purposes.

6.1.1 *Validation*

The first objective we suggest is a lower bound on the validation predictive loss. As the objective of GP regression is often to achieve the best possible predictive distribution, we can estimate this by evaluating the predictive distribution against a held-out validation set. This has the benefit that it directly penalizes the quantity that we are most interested in which, unlike the classical loss, does not depend on predictive covariance between points. This does mean that it is an inappropriate loss function in circumstances where posterior covariances are important. On the downside, unlike the classical loss, we must remove some datapoints from our training set in order to construct a validation set.

The formula for the validation loss is:

$$\mathcal{L}_{val} = -\frac{1}{2}\sum_i \left(\frac{(y_i - \mu_i)^2}{\sigma_i^2} + \log \sigma_i^2 + \log 2\pi \right) \tag{183}$$

We can use the upper and lower bounds on posterior variances from Section **??** to lower bound this objective function, and this lower bound is the objective function that we will use for optimization:

$$\mathcal{L}_{val} \geqslant \check{\mathcal{L}}_{val} \tag{184}$$

$$= -\frac{1}{2}\sum_i \left(\frac{(y_i - \mu_i)^2}{\check{\sigma}_i^2} + \log \hat{\sigma}_i^2 + \log 2\pi \right) \tag{185}$$

The derivatives for this objective are as follows:

$$\frac{d\breve{\mathcal{L}}_{val}}{d\vec{\theta}_j} = \sum_i -\frac{(y_i - \mu_i)}{\breve{\sigma}_i^2}\frac{d\mu_i}{d\vec{\theta}_j} - \frac{(y_i - \mu_i)^2}{\breve{\sigma}_i^4}\frac{d\breve{\sigma}_i^2}{d\vec{\theta}_j} - \frac{1}{\hat{\sigma}_i^2}\frac{d\hat{\sigma}_i^2}{d\vec{\theta}_j} \tag{186}$$

$$= \sum_i -\frac{(y_i - \mu_i)}{\breve{\sigma}_i^2}\left(\left(\frac{d\mathbf{K}_-}{d\vec{\theta}_i} - \mathbf{K}_-\mathbf{K}^{-1}\frac{d\mathbf{K}}{d\vec{\theta}_i}\right)\mathbf{K}^{-1}\vec{y}\right) - \frac{(y_i - \mu_i)^2}{\breve{\sigma}_i^4}\frac{d\breve{\sigma}_i^2}{d\vec{\theta}_j} - \frac{1}{\hat{\sigma}_i^2}\frac{d\hat{\sigma}_i^2}{d\vec{\theta}_j} \tag{187}$$

The total cost of calculating the derivatives is that of calculating the variance bounds and their derivatives, as well as two linear system solves. In the case of the subset variance bounds, without refinement, the computational cost is $O(NM(I + M))$ operations.

6.1.2 *Variational*

The second objective function that we derive is based on a variational approximation. Whenever a variational approximation is made to a distribution there is an associated lower bound on the log-likelihood, sometimes refered to as the evidence lower bound, or ELBO[Jordan et al., 1999]. This type of lower bound has been derived for low-rank GPs in [Titsias, 2009b] and extended in [Titsias, 2009a, Hensman et al., 2013].

We take this form of bound further by extending it beyond low-rank GPs to those that have a low-rank posterior covariance and arbitrary posterior mean. This results in an approximation that is at least as good as a fully low-rank approximation and better in all cases except where the true posterior mean is low-rank. In cases where an M-efficient kernel (as described in Chapter 5) is used, this can be calculated with no increase in computational complexity over the low-rank variational approximation.

A variational approximation to a full distribution P is made by finding the distribution in a restricted class Q that is the closest to P in terms of KL-divergence. The resulting bound on the full likelihood is:

$$\mathcal{L}_{var} = \log P - D_{KL}\left(Q \parallel P\right) \tag{188}$$

Since the KL divergence is always positive, this is a lower bound on $\log P$ that is tight when the exact and approximate posteriors are the same. To form our bound, we approximate the posterior distribution of the GP with an approximate posterior $Q(\vec{\theta}, \vec{\alpha})$ that is defined as follows:

$$\vec{\mu}_* = \mathbf{K}_{*x}\vec{\alpha} \tag{189}$$

$$\mathbf{\Sigma}_* = \mathbf{K}_* - \mathbf{K}_{*m}\mathbf{K}_m^{-1}\mathbf{K}_{m*} \tag{190}$$

For this form of Q, the lower bound is:

$$\begin{aligned}\mathcal{L}_{var} = &-\frac{1}{2}\left(\vec{y}^\intercal \mathbf{K}^{-1}\vec{y} + \log\det\mathbf{K} + N\log 2\pi\right)\\ &-\frac{1}{2}\left(Tr(\mathbf{\Sigma}_P^{-1}\mathbf{\Sigma}_Q) + \left(\mathbf{K}_{-}\mathbf{K}^{-1}\vec{y} - \mathbf{K}_{-}\vec{\alpha}\right)^\intercal \mathbf{\Sigma}_P^{-1}\left(\mathbf{K}_{-}\mathbf{K}^{-1}\vec{y} - \mathbf{K}_{-}\vec{\alpha}\right)\right)\\ &-\frac{1}{2}\left(N - \log\left(\frac{\det\mathbf{\Sigma}_Q}{\det\mathbf{\Sigma}_P}\right)\right)\end{aligned} \tag{191}$$

Where $\mathbf{\Sigma}_P$ and $\mathbf{\Sigma}_Q$ are the posterior variances at the training points under each distribution.

$$\mathbf{\Sigma}_P = \mathbf{K} - \mathbf{K}_{-}\mathbf{K}^{-1}\mathbf{K}_{-} \tag{192}$$

$$\mathbf{\Sigma}_Q = \mathbf{K} - \mathbf{K}_{xm}\mathbf{K}_m^{-1}\mathbf{K}_{mx} \tag{193}$$

We will not be able to efficiently calculate this bound, so we will derive a further lower bound that can be efficiently calculated. To begin with, we separate the terms into "fit", "penalizers" and "constants":

$$\mathcal{L}_{var} = \mathcal{L}_{var}^{fit} + \mathcal{L}_{var}^{penalier} + \mathcal{L}_{var}^{const} \tag{194}$$

$$\begin{aligned}= &-\frac{1}{2}\vec{y}^\intercal\mathbf{K}^{-1}\vec{y} - \frac{1}{2}\left(\mathbf{K}_{-}\mathbf{K}^{-1}\vec{y} - \mathbf{K}_{-}\vec{\alpha}\right)^\intercal\mathbf{\Sigma}_P^{-1}\left(\mathbf{K}_{-}\mathbf{K}^{-1}\vec{y} - \mathbf{K}_{-}\vec{\alpha}\right) \quad \text{(Fit)}\\ &-\frac{1}{2}Tr(\mathbf{\Sigma}_P^{-1}\mathbf{\Sigma}_Q) - \frac{1}{2}\log\det\mathbf{K} + \frac{1}{2}\log\left(\frac{\det\mathbf{\Sigma}_Q}{\det\mathbf{\Sigma}_P}\right) \quad \text{(Penalizer)}\\ &+\frac{N}{2} + \frac{N}{2}\log 2\pi \quad \text{(Constants)}\end{aligned} \tag{195}$$

Starting with our attention on the fit term, we can bound the second element by observing that the smallest eigenvalue of $\mathbf{\Sigma}_P$ is bounded below by σ^2, and thus the largest is of $\mathbf{\Sigma}_P^{-1}$ is bounded above by $\frac{1}{\sigma^2}$.

$$\mathcal{L}_{var}^{fit} = -\frac{1}{2}\vec{y}^\intercal\mathbf{K}^{-1}\vec{y} - \frac{1}{2}\left(\mathbf{K}_{-}\mathbf{K}^{-1}\vec{y} - \mathbf{K}_{-}\vec{\alpha}\right)^\intercal\mathbf{\Sigma}_P^{-1}\left(\mathbf{K}_{-}\mathbf{K}^{-1}\vec{y} - \mathbf{K}_{-}\vec{\alpha}\right) \tag{196}$$

$$\geqslant -\frac{1}{2}\vec{y}^\intercal\mathbf{K}^{-1}\vec{y} - \frac{1}{2\sigma^2}\left(\mathbf{K}_{-}\mathbf{K}^{-1}\vec{y} - \mathbf{K}_{-}\vec{\alpha}\right)^\intercal\left(\mathbf{K}_{-}\mathbf{K}^{-1}\vec{y} - \mathbf{K}_{-}\vec{\alpha}\right) \tag{197}$$

$$= -\frac{1}{2}\vec{y}^\intercal\mathbf{K}^{-1}\vec{y} - \frac{1}{2\sigma^2}(\vec{y} - \mathbf{K}\vec{\alpha})^\intercal\left(\mathbf{I} - \sigma^2\mathbf{K}^{-\mathbf{1}}\right)^2(\vec{y} - \mathbf{K}\vec{\alpha}) \tag{198}$$

The term can be further bounded by observing that $\left(\mathbf{I} - \sigma^2\mathbf{K}^{-\mathbf{1}}\right)$ is PSD with all eigenvalues between 0 and 1. This implies that $\vec{v}^\intercal\left(\mathbf{I} - \sigma^2\mathbf{K}^{-\mathbf{1}}\right)\vec{v} \geqslant \vec{v}^\intercal\left(\mathbf{I} - \sigma^2\mathbf{K}^{-\mathbf{1}}\right)^2\vec{v}$

$$\geqslant -\frac{1}{2}\vec{y}^\intercal\mathbf{K}^{-1}\vec{y} - \frac{1}{2\sigma^2}(\vec{y} - \mathbf{K}\vec{\alpha})^\intercal\left(\mathbf{I} - \sigma^2\mathbf{K}^{-\mathbf{1}}\right)(\vec{y} - \mathbf{K}\vec{\alpha}) \tag{199}$$

$$= -\frac{1}{2\sigma^2}\left(\vec{y}^\intercal\vec{y} - 2\vec{y}^\intercal\mathbf{K}_{-}\vec{\alpha} + \vec{\alpha}^\intercal\mathbf{K}_{-}\mathbf{K}\vec{\alpha}\right) \tag{200}$$

This is the same as the conjugate gradient lower bound from Equation 64.

Secondly, we turn our attention to the penalizer term:

$$\mathcal{L}_{var}^{penalier} = -\frac{1}{2} Tr(\mathbf{\Sigma}_P^{-1}\mathbf{\Sigma}_Q) - \frac{1}{2}\log\det\mathbf{K} + \frac{1}{2}\log\left(\frac{\det\mathbf{\Sigma}_Q}{\det\mathbf{\Sigma}_P}\right) \tag{201}$$

We can simplifiy the log determinants by using the block log-determinant formula:

$$\log\det\mathbf{K} = \log\det\mathbf{K}_m + \log\det\left(\mathbf{K}_{\neg m} - \mathbf{K}_{\neg m m}\mathbf{K}_m^{-1}\mathbf{K}_{m\neg m}\right) \tag{202}$$

and since $\mathbf{K}_{\neg m} - \mathbf{K}_{\neg m m}\mathbf{K}_m^{-1}\mathbf{K}_{m\neg m}$ is a submatrix of $\mathbf{\Sigma}_Q$ and the remaining block can be bounded

$$\log\det\mathbf{\Sigma}_Q \geqslant \log\det\left(\mathbf{K}_{\neg m} - \mathbf{K}_{\neg m m}\mathbf{K}_m^{-1}\mathbf{K}_{m\neg m}\right) + m\log\sigma^2 \tag{203}$$

Which gives us the first bound on the penalizer term:

$$\mathcal{L}_{var}^{penalier} \geqslant -\frac{1}{2} Tr(\mathbf{\Sigma}_P^{-1}\mathbf{\Sigma}_Q) - \frac{1}{2}\log\det\mathbf{K}_m - \frac{1}{2}\log\det\mathbf{\Sigma}_P + \frac{m}{2}\log\sigma^2 \tag{204}$$

Next we can re-arrange $\mathbf{\Sigma}_P$:

$$\begin{aligned}
\mathbf{\Sigma}_P &= \mathbf{K} - \mathbf{K}_- \mathbf{K}^{-1}\mathbf{K}_- && (205)\\
&= \mathbf{K} - \left(\mathbf{K} - \sigma^2\mathbf{I}\right)\mathbf{K}^{-1}\left(\mathbf{K} - \sigma^2\mathbf{I}\right) && (206)\\
&= 2\sigma^2\mathbf{I} - \sigma^4\mathbf{K}^{-1} && (207)\\
&= \sigma^2\left(2\mathbf{I} - \sigma^2\mathbf{K}^{-1}\right) && (208)
\end{aligned}$$

Which allows us to bound the log-determinant:

$$\begin{aligned}
\log\det\mathbf{\Sigma}_P &= \log\det\left(\sigma^2\left(2\mathbf{I} - \sigma^2\mathbf{K}^{-1}\right)\right) && (209)\\
&= N\log\sigma^2 + \log\det\left(2\mathbf{I} - \sigma^2\mathbf{K}^{-1}\right) && (210)
\end{aligned}$$

and since the eigenvalues of $\sigma^2\mathbf{K}^{-1}$ are bounded between 0 and 1

$$\begin{aligned}
&\leqslant N\log\sigma^2 + \log\det\left(2\mathbf{I}\right) && (211)\\
&= N\log\sigma^2 + N\log 2 && (212)
\end{aligned}$$

Which gives us

$$\mathcal{L}_{var}^{penalier} \geqslant -\frac{1}{2} Tr(\mathbf{\Sigma}_P^{-1}\mathbf{\Sigma}_Q) - \frac{1}{2}\log\det\mathbf{K}_m - \frac{N-m}{2}\log\sigma^2 \tag{213}$$

And finally, von Neumann's trace inequality gives us that $Tr(\mathbf{AB}) \leqslant \sum \lambda_i(\mathbf{A})\lambda_i(\mathbf{B})$

$$\geqslant -\frac{1}{2}\sum \lambda_i(\mathbf{\Sigma}_P^{-1})\lambda_i(\mathbf{\Sigma}_Q) - \frac{1}{2}\log\det \mathbf{K}_m - \frac{N-m}{2}\log\sigma^2 \tag{214}$$

$$\geqslant -\frac{1}{2}\sum \lambda_{max}(\mathbf{\Sigma}_P^{-1})\lambda_i(\mathbf{\Sigma}_Q) - \frac{1}{2}\log\det \mathbf{K}_m - \frac{N-m}{2}\log\sigma^2 \tag{215}$$

$$\geqslant -\frac{1}{2\sigma^2}\sum \lambda_i(\mathbf{\Sigma}_Q) - \frac{1}{2}\log\det \mathbf{K}_m - \frac{N-m}{2}\log\sigma^2 \tag{216}$$

$$= -\frac{1}{2\sigma^2}Tr(\mathbf{\Sigma}_Q) - \frac{1}{2}\log\det \mathbf{K}_m - \frac{N-m}{2}\log\sigma^2 \tag{217}$$

This gives us the final lower bound on the objective:

$$\mathcal{L}_{var} \geqslant \breve{\mathcal{L}}_{var} \tag{218}$$

$$\begin{aligned} &= -\frac{1}{2\sigma^2}\left(\vec{y}^\intercal\vec{y} - 2\vec{y}^\intercal\mathbf{K}_-\vec{\alpha} + \vec{\alpha}^\intercal\mathbf{K}_-\mathbf{K}\vec{\alpha}\right) \\ &\quad - \frac{1}{2\sigma^2}Tr(\mathbf{\Sigma}_Q) - \frac{1}{2}\log\det\mathbf{K}_m - \frac{(n-m)}{2}\log\sigma^2 \\ &\quad + \frac{N}{2} + \frac{N}{2}\log 2\pi \end{aligned} \tag{219}$$

Which results in the following derivatives:

$$\begin{aligned} \frac{d\breve{\mathcal{L}}_{var}}{d\vec{\theta}} &= -\frac{1}{2\sigma^2}\left(-2\vec{y}^\intercal\frac{d\mathbf{K}_-}{d\vec{\theta}}\vec{\alpha} + \vec{\alpha}^\intercal\frac{d\mathbf{K}_-}{d\vec{\theta}}\mathbf{K}\vec{\alpha} + \vec{\alpha}^\intercal\mathbf{K}_-\frac{d\mathbf{K}}{d\vec{\theta}}\vec{\alpha}\right) \\ &\quad - \frac{1}{2\sigma^2}Tr\left(\frac{d\mathbf{\Sigma}_Q}{d\vec{\theta}}\right) - \frac{1}{2}Tr\left(\mathbf{K}_m^{-1}\frac{d\mathbf{K}_m}{d\vec{\theta}}\right) \end{aligned} \tag{220}$$

Which can be evaluated in $O(NM^2)$ operations.

Derivative wrt $\vec{\alpha}$

If we also evaluate the derivative with respect to $\vec{\alpha}$:

$$\frac{d\breve{\mathcal{L}}_{var}}{d\vec{\alpha}} = \frac{1}{\sigma^2}\mathbf{K}_-\left(\mathbf{K}\vec{\alpha} - \vec{y}\right) \tag{221}$$

We see that for a given setting of $\vec{\theta}$, this is optimized at $\vec{\alpha} = \mathbf{K}^{-1}\vec{y}$, recovering the exact solution for the GP. However, despite having the same optimal point, this gradient is not the same as gradient used to optimize $\vec{\alpha}$ in Section 4.3. It is a multiplicative factor of $\frac{1}{\sigma^2}\mathbf{K}_-$ different. This factor is not desirable as we know that conjugate gradient converges as $O(\sqrt{\kappa})$ and the condition number:

$$\kappa\left(\left(\mathbf{K}_- + \sigma\mathbf{I}\right)\mathbf{K}_-\right) \geqslant \kappa\left(\mathbf{K}_-^2\right) = \kappa(\mathbf{K}_-)^2 \geqslant \kappa(\mathbf{K})^2$$

We already have the tools to recover the original gradients, since we know that we can pre-condition the system with $\sigma^2\mathbf{K}_-^{-1}$. However, we don't actually

need to calculate the preconditioner at all in this case, since we know that the gradient after applying the pre-conditioner is $\vec{y} - \mathbf{K}\vec{\alpha}$. This is known as *implicit pre-conditioning*[Ratliff and Bagnell, 2007].

This property means that when optimizing with this objective, we can choose to optimize $\vec{\alpha}$ and $\vec{\theta}$ jointly.

6.1.3 *Experiments*

The results of optimizing a GP with an isotropic SE kernel against the three different objectives are summarized in Table 12. The optimizations are run to convergence using a standard non-linear CG solver. The "speed" is calculated as the relative number of operations required compared to the standard objective. The number of operations is affected by both the number of iterations taken to converge, as well as the different costs of calculating the objective and its derivatives. The effectiveness of the optimization is measured in terms of log probability per data point, where higher is better. In terms of predictive accuracy, the results are split between the full log-likelihood and the validation error, though the validation error in most cases is between 2x-10x faster.

	Standard		Variational		Validation	
	Speed	Log prob	Speed	Log prob	Speed	Log prob
MPG	1.0x	**1.11**	1.7x	-0.40	0.9x	0.41
Births in Quebec	1.0x	-7.4	21.4x	-7.5	2.4x	**-1.5**
Boston housing	1.0x	**2.5**	0.9x	1.8	0.6x	0.29
Australian beer	1.0x	-5.90	7.9x	-6.00	4.9x	**-1.22**
Sea levels	1.0x	**-2.15**	11.8x	-3.13	3.5x	-3.92
Daily min temp	1.0x	-2.03	8.8x	-2.20	4.1x	**-1.39**
Bodyfat	1.0x	**-0.09**	202.7x	-0.45	175x	-0.45
Prod of H_2SO_4	1.0x	-5.33	9.6x	-6.05	1.9x	**-4.32**

Table 12: Results for different optimization objectives over different datasets using the SE kernel

6.2 PRE-CONDITIONING

In Section 4.7, we showed that sensible pre-conditioning is an important step to the optimization procedure in our framework. We will show that when optimizing the kernel parameters $\vec{\theta}$, pre-conditioning is no less important. The convergence rate of optimizers are still sensitive to a generalized notion of the condition number of the problem, and as the hyper-parameters are non-convex it can also affect the quality of the final solution.

In the following sections we give an intuition about the meaning of pre-conditioners in non-convex optimization, showing how methods such as natural gradients and Newton's method can be viewed as a generalization of pre-conditioning, and then derive 5 different pre-conditioners for hyper-parameters optimization in GPs.

6.2.1 *Introduction to multivariate gradient optimization*

In this section we will derive multivariate gradient descent, highlighting the important choice that results in the difference between many gradient optimization methods.

Gradient descent

Gradient descent, the simplest of gradient optimization schemes, is often introduced simply as the following equation:

$$x_{t+1} = x_t + \eta \frac{df(x_t)}{dx} \tag{222}$$

At each iteration, the algorithm takes a step of "size" η in the direction of the gradient. In 1D the intuition is clearly visible in Figure 56. At the point indicated by the arrow, the gradient is shown by the red line. We would expect that taking a reasonable step[1] in the direction that the gradient points downward will bring us closer to a (possibly local) minima.

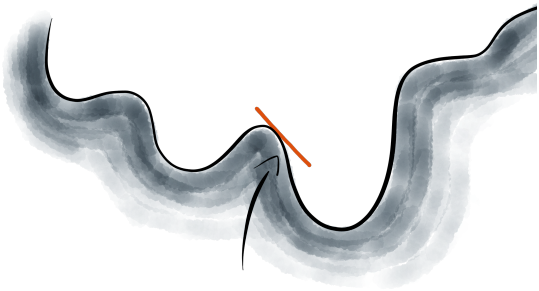


Figure 56: A function and its gradient at a point

It seems reasonable that this formulation would extend directly to higher dimensions; after all, we just want to take steps that reduce the function value, regardless of the dimensionality of the problem. This seems to imply a simple extension to the multivariate case:

$$\vec{x}_{t+1} = \vec{x}_t + \eta \frac{df(\vec{x}_t)}{d\vec{x}} \tag{223}$$

which is multivariate gradient descent. While this seems sensible — and is a well established optimization method — it transgresses a cardinal sin of first-year

1 This can be calculated using a line search[Boyd and Vandenberghe, 2009]

undergraduate physics. That is, as it is normally presented it is *dimensionally inconsistent*, a point that is rarely mentioned.

As an example, let's consider the heat shield on a space probe. As it re-enters the Earth's atmosphere it begins to heat up and the temperature increases, but not evenly over the entire shield. We are interested in finding the point on a 1D section of the probe that reachest the highest temperature, and at what time. We have a function $f(x,t)$ that calculates the temperature at a point on this section given by a distance from the end x in meters, t seconds after it has entered the atmosphere. We want to find a local maximum of temperature across these two variables. In this case, the function we are maximizing returns a temperature (units K) and takes as inputs a time measurement (units s) and position (units m). Thus the dimensions of Equation 223 are:

$$\begin{bmatrix} s \\ m \end{bmatrix} = \begin{bmatrix} s \\ m \end{bmatrix} + \alpha \begin{bmatrix} Ks^{-1} \\ Km^{-1} \end{bmatrix} \tag{224}$$

α are the units of η and there is clearly no choice for them that will make this set of equations consistent. This is not to say that gradient descent in the multivariate setting is incorrect — it seems unlikely, given the prevalence and effectiveness of first order gradient methods. However, this dimensional inconsistency should serve as a red flag that we have made a mathematical omission somewhere. We will introduce the more general idea of multivariate steepest descent, which will resolve this inconsistency.

Direction of steepest descent

In order to properly derive gradient optimization in the multivariate setting, we start by more precisely defining what we are trying to achieve. One way to define what we want is that at each iteration of optimization our step be in the direction of steepest descent. This is exactly what happens when using the gradient in the 1D case as there are only two directions, left and right, and the direction descending will clearly be a steeper descent than the one that is ascending.

We can define this direction of steepest descent as the direction of the step of infinitesimal size that results in the largest decrease in the function value.

$$\begin{aligned} &\arg\min_{\delta\theta} && L(\theta + \delta\theta) \\ &\text{subject to} && |\delta\theta|_X < \epsilon \end{aligned} \tag{225}$$

In the limit of $\epsilon \to 0$.

This gives us the direction which has the most negative directional derivative; the direction which is sloped most strongly downward, exactly what our intuition expected. Crucially though, this answer is with respect to our choice of norm X.

In brief, this norm is necessary because we often have a meaningful notion of distance in our problems that is different than simply the Euclidean norm on the optimization space. This choice of norm characterizes the different forms of steepest descent and is the key factor for the rate of convergence of the algorithm. It should be noted that while the norm is critical for the convergence rate of the algorithm, steepest descent has the same convergence assurances for any sensible

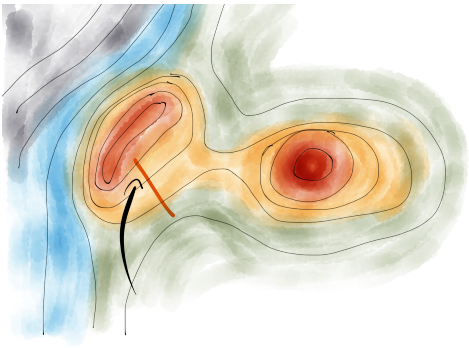
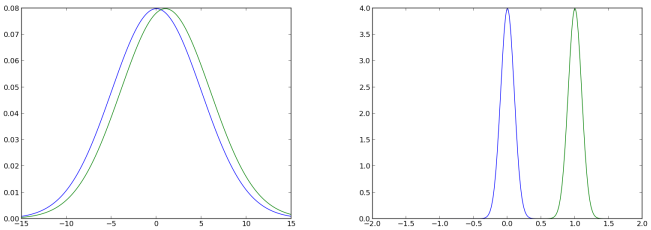


Figure 57: A contour illustration of a 2D function and the direction of steepest descent at a point

choice of norm. After all, as long as the algorithm is always heading downhill by some measure, it will converge eventually.

To understand the importance of the choice of norm, consider the following example: we are optimizing a function f that takes as an argument a 1D Gaussian distribution. We represent this distribution by its parameters: mean μ and variance σ^2. But do we believe that the Euclidean norm on this parameter space describes our notion of difference between these distributions well? Using this norm, the distributions in Figure 58a are as similar to each other as the two distributions in Figure 58b. While in some cases this may well be the correct choice of norm, in most cases it probably will not.



(a) Gaussian with $\mu = 0, \sigma = 5$ and $\mu = 1, \sigma = 5$

(b) Gaussian with $\mu = 0, \sigma = .1$ and $\mu = 1, \sigma = .1$

Figure 58: Two sets of Gaussian distributions with the same distance under the Euclidean norm

Solving Equation 225 results in a steepest descent direction of the following form:

$$\mathbf{P}_X(\vec{\theta}_t)^{-1}\frac{df(\vec{\theta}_t)}{d\vec{\theta}} \tag{226}$$

Where $\mathbf{P}$ is recognisable as a pre-conditioning matrix[Boyd and Vandenberghe, 2009]. The difference between this generalized pre-conditioner, and those we used in Section 4.7 is that this pre-conditioner can vary with $\vec{\theta}$.

Existing methods

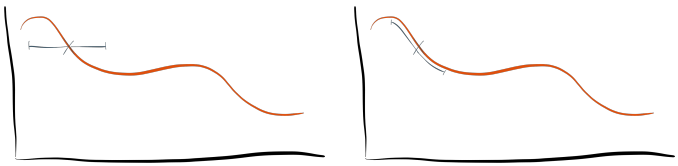
To make this connection more concrete, we look at three different choices of norms that result in the methods of methods of gradient descent, Newton's method and natural gradients.

The simplest choice of norm is the Euclidean norm on the parameter vector. This results in a transformation matrix of $\mathbf{I}$. This provides a resolution to the dimensional inconsistency of Equation 222: we forgot to explicitly write down the pre-conditioner. We can check the dimensions are consistent by ensuring that the units of $\mathbf{I}\vec{\theta} = \frac{df(\vec{\theta})}{d\vec{\theta}}$ are consistent:

$$\begin{bmatrix} Ks^{-2} & Ks^{-1}m^{-1} \\ Ks^{-1}cm^{-1} & Km^{-2} \end{bmatrix}\begin{bmatrix} s \\ m \end{bmatrix} = \begin{bmatrix} Ks^{-1} \\ Km^{-1} \end{bmatrix} \tag{227}$$

This should serve to emphasize the point that whenever performing a multivariate gradient method we have made a choice of norm, whether we make it explicit or not.

Newton's method[Evtushenko, 1985] is an example of a general purpose gradient method that can be derived through a particular choice of norm. It is generally given as the gold standard for optimization in terms of convergence, though often it is too computationally expensive in practice for large numbers of variables. The norm resulting in Newton's method is the Hessian norm, which measures the distance along the optimization manifold. This is illustrated in Figure 59b.



(a) Epsilon ball for Euclidean norm (b) Epsilon ball for Hessian norm

Figure 59: Comparison of Euclidean and Hessian norm

When optimizing probability distributions, the choice of the symmetrized KL-divergence as a distance metric results in natural gradients[Amari and Douglas, 1998]. This is a sensible choice of distance between probability distributions; if

the loss function we are optimizing depends on the distribution and not directly on the parameters, this is likely going to be a better norm.

How to choose a norm

As with pre-conditioning in the quadratic context, the choice of pre-conditioner, in this case the norm, is an important factor in the convergence of gradient methods.

Since the norm is such a critical factor in the convergence of steepest descent, the question of how to select it for a given problem is a very important one. Intuitively, the choice of norm should give us a "sensible" notion of the distance between two points in the optimization space. For a more detailed understanding of what this means, we need to understand the concept of sublevel sets, their condition number, and their effect on the convergence of steepest descent.

For a given function f, a sublevel set of a point $\vec{x}$ is the set of points

$$C_{\vec{x}} = \{\vec{x}_i \mid f(\vec{x}_i) \leqslant f(\vec{x})\} \tag{228}$$

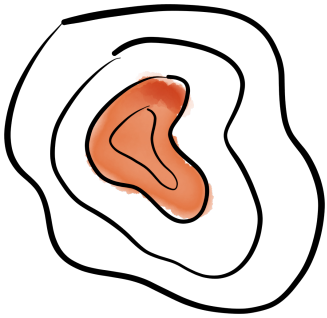


Figure 60: Illustration of a sublevel set

Simply put, it is the set of points that are at the same level, or lower, than a given point. For a convex function, such as the one illustrated in Figure 60, these points will form a closed region. The width of a sublevel set in a certain direction is the width of the set projected onto that direction, as shown in Figure 61.

The *condition number* of a sublevel set is the square of the ratio between the maximum and minimum width of the sublevel set. A width of a sublevel set in a direction $\vec{v}$ is the length of the projection of the sublevel set onto that vector $\vec{v}$.

$$\kappa = \left(\frac{W_{max}}{W_{min}}\right)^2 \tag{229}$$

If this number is small then the sublevel set has a similar width in all directions, if the number is large then it is much wider in some directions than others. This is exactly analoguous to the condition number of a matrix; if the eigenvalues are all similar then the condition number is small and if there are some eigenvalues that are much larger than others, the condition number will be large. For a

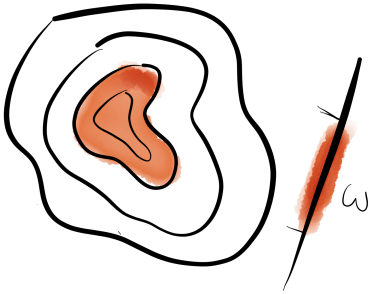


Figure 61: Illustration of a sublevel set

quadratic objective function, the condition number of the sublevel sets and the matrix are the same. The convergence of gradient methods depend on this condition number [Boyd and Vandenberghe, 2009] — the smaller the condition number, the faster the convergence.

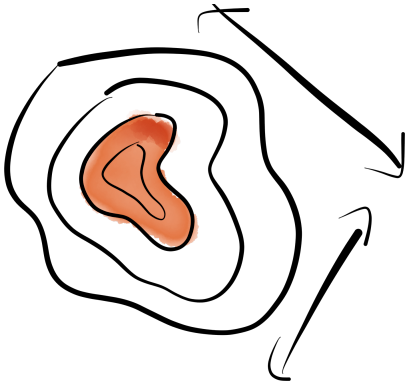


Figure 62: Illustration of a sublevel set

Because of this, we want to choose a norm such that the resulting condition number is as small as possible. This is not a particularly intuitive statement, but we can consider the problem in the following manner. Performing steepest descent with a norm other than the standard Euclidean norm is equivalent to performing steepest descent with the Euclidean norm in a transformed space. As an example, if the norm is of the form $\langle x_a, x_b \rangle_{\mathbf{P}} = x_a \mathbf{P} x_b$, this is equivalent to performing descent in the transformed space $x' = \mathbf{P}^{\frac{1}{2}} x$ with the standard Euclidean

norm. For different norms, the transformation may be non-linear.

Now our previous statement on condition numbers can be interpreted for arbitrary norm: the smaller the condition number of the sublevel sets in the *transformed space*, the faster the algorithm will converge. Rephrasing with the definition of sublevel sets: we ideally want points that are the same distance from the optimum in the transformed space to have the same function value. ie:

$$f(\vec{x}_a) \approx f(\vec{x}_b) \tag{230}$$

for

$$\left\langle \vec{x}_a - \vec{x}_{opt}, \vec{x}_a - \vec{x}_{opt} \right\rangle_P = \left\langle \vec{x}_b - \vec{x}_{opt}, \vec{x}_b - \vec{x}_{opt} \right\rangle_P \tag{231}$$

For a quadratic norm, Equation 231 defines an ellipse around $\vec{x}_{opt}$, shown in Figure 63.

Figure 63: Illustration of a sublevel set

This definition allows us to specify examples of aspirational norms — ones that would be ideal if they could be used in practice. One such norm would be where $d(\vec{x}_a, \vec{x}_{opt}) = \|f(\vec{x}_a) - f(\vec{x}_{opt})\|$. In such a space, the condition number of the sublevel sets would be 1, as all points of the same function value would be the same distance from the optima.

As an example we consider a quadratic objective function $f(\vec{x}) = \frac{1}{2}\vec{x}^\intercal \mathbf{A}\vec{x} - \vec{x}^\intercal \vec{b}$, which will have a minimum $\vec{x}_{opt} = \mathbf{A}^{-1}\vec{b}$. We now show that using the above norm in this case recovers the ideal quadratic pre-conditioner from Section 4.7.

$$d(\vec{x}_a, \vec{x}_{opt}) = \|f(\vec{x}_a) - f(\vec{x}_{opt})\| \tag{232}$$
$$= \frac{1}{2}\vec{x}_a^\intercal \mathbf{A}\vec{x}_a - \vec{x}_a^\intercal \vec{b} + \frac{1}{2}\vec{x}_{opt}^\intercal \mathbf{A}\vec{x}_{opt} \tag{233}$$
$$= \frac{1}{2}\vec{x}_a^\intercal \mathbf{A}\vec{x}_a - \vec{x}_a^\intercal \mathbf{A}\vec{x}_{opt} + \frac{1}{2}\vec{x}_{opt}^\intercal \mathbf{A}\vec{x}_{opt} \tag{234}$$
$$= \left(\vec{x}_a - \vec{x}_{opt}\right)^\intercal \mathbf{A}\left(\vec{x}_a - \vec{x}_{opt}\right) \tag{235}$$
$$= \|\vec{x}_a - \vec{x}_{opt}\|_{\mathbf{A}} \tag{236}$$

This is the $\mathbf{A}$-norm distance metric, $d(\vec{x}_a, \vec{x}_b) = \|\vec{x}_a - \vec{x}_b\|_{\mathbf{A}}$, which results in a pre-conditioner of $\mathbf{A}^{-1}$.

While we can evaluate a norm of this form, we can't expect to do closed form transformations with it most of the time. Instead we should use this as a guide to the selection of norms. Boiled down to a rough heuristic, this implies that we select norms such that the distance between two points is proportional to how much they likely differ in function value.

Further, while we haven't discussed multi-modal settings, this is not assured to help in finding an optimal minima, but pre-conditioning will speed convergence to *some* minima.

6.2.2 $\vec{\theta}$ *pre-conditioners*

While natural gradients are arguably the best general purpose distance metric for Gaussian Processes, they are also very expensive to compute. It is therefore worthwhile investigating alternative norms that attempt to capture some of the structure of the optimization space, but are not as computationally intensive. In this section, we use the intuition from Section 6.2.1 to construct a set of efficient pre-conditioners for optimization, which are summarized in Table 13.

Gradient	Distance metric	Cost
Standard	$\left\|\vec{\theta}_a - \vec{\theta}_b\right\|^2$	$O(D)$
Gram	$\|\mathbf{K}_a - \mathbf{K}_b\|_{Fr}^2$	$O(DNMI)$
Alpha	$\left\|\mathbf{K}_a^{-1}\vec{y} - \mathbf{K}_b^{-1}\vec{y}\right\|^2$	$O(DNMI)$
Predictive	$\|\vec{\mu}_a - \vec{\mu}_b\|^2$	$O(DNMI)$
Natural predictive	$KL\left(p\left(\vec{y} \mid \vec{\mu}_a, \vec{\sigma}_a^2\right), p\left(\vec{y} \mid \vec{\mu}_b, \vec{\sigma}_b^2\right)\right)$	$O(DNM(I+M))$

Table 13: $\vec{\theta}$ gradients and distance measures

Gram gradients

The Gram matrix is a more direct representation of a kernel to a GP than the hyper-parameters of that kernel.

A norm we can use to define the distance between two matrices is the Frobenius norm, used here for its convenient differentiability. The Gram gradient metric is defined as:

$$d(\vec{\theta}_a, \vec{\theta}_b) = \left|\mathbf{K}(\vec{\theta}_a) - \mathbf{K}(\vec{\theta}_b)\right|_{Fr}^2 \tag{237}$$

Returning to the definition of steepest descent (Section **??**), we wish to solve:

$$\begin{aligned} \arg\max_{\delta\vec{\theta}} \quad & L(\vec{\theta} + \delta\vec{\theta}) \\ \text{subject to} \quad & \left|\mathbf{K}(\vec{\theta}) - \mathbf{K}(\vec{\theta} + \delta\vec{\theta})\right|_{Fr}^2 < \epsilon \end{aligned} \tag{238}$$

In the limit of $\epsilon \to 0$.

As this is a constrained optimization problem, it can be solved by the use of a Lagrangian.

$$\Lambda = L(\vec{\theta} + \delta\vec{\theta}) + \lambda\left(\left|\mathbf{K}(\vec{\theta}) - \mathbf{K}(\vec{\theta} + \delta\vec{\theta})\right|_{Fr}^2 - \epsilon\right) \tag{239}$$

As we are allowing $\epsilon \to 0$, we can make first-order approximations to a number of terms in the equation:

$$L(\vec{\theta} + \delta\vec{\theta}) = L(\vec{\theta}) + \frac{\partial L}{\partial \vec{\theta}}\delta\vec{\theta} \tag{240}$$

$$\left|\mathbf{K}(\vec{\theta}) - \mathbf{K}(\vec{\theta} + \delta\vec{\theta})\right|_{Fr}^2 = \left|\sum_i \delta\vec{\theta}_i \frac{\partial \mathbf{K}}{\partial \vec{\theta}_i}\right|_{Fr}^2 \tag{241}$$

To avoid tensor notation later on, we take the derivative of the equation above with respect to an individual hyper-parameter:

$$\frac{\partial}{\partial \vec{\theta}_i}\left|\sum_j \delta\vec{\theta}_j \frac{\partial \mathbf{K}}{\partial \vec{\theta}_j}\right|_{Fr}^2 = 2Tr\left(\left(\sum_j \delta\vec{\theta}_j \frac{\partial \mathbf{K}}{\partial \vec{\theta}_j}\right)\frac{\partial}{\partial \vec{\theta}_i}\left(\sum_j \delta\vec{\theta}_j \frac{\partial \mathbf{K}}{\partial \vec{\theta}_j}\right)\right) \tag{242}$$

$$= 2Tr\left(\left(\sum_j \delta\vec{\theta}_j \frac{\partial \mathbf{K}}{\partial \vec{\theta}_j}\right)\frac{\partial \mathbf{K}}{\partial \vec{\theta}_i}\right) \tag{243}$$

$$= 2\sum_j \delta\vec{\theta}_j Tr\left(\frac{\partial \mathbf{K}}{\partial \vec{\theta}_j}\frac{\partial \mathbf{K}}{\partial \vec{\theta}_i}\right) \tag{244}$$

$$:= 2\sum_j \delta\vec{\theta}_j \mathbf{S}_{ij} \tag{245}$$

$$= \mathbf{S}_i^\intercal \delta\vec{\theta}_j \tag{246}$$

For notational clarity, we define a $D \times D$ matrix $\mathbf{S}$ such that $\mathbf{S}_{ij} = Tr\left(\frac{\partial \mathbf{K}}{\partial \vec{\theta}_i}\frac{\partial \mathbf{K}}{\partial \vec{\theta}_j}\right)$. Using these results, we find the optimal point of Λ by setting the gradient to 0.

$$\frac{\partial \Lambda}{\partial \vec{\theta}} = 2\frac{\partial L}{\partial \vec{\theta}} + \lambda \frac{\partial}{\partial \vec{\theta}} \left| \sum_i \delta\vec{\theta}_i \frac{\partial \mathbf{K}}{\partial \vec{\theta}_i} \right|^2_{Fr} \tag{247}$$

$$\frac{\partial \Lambda}{\partial \vec{\theta}} = 2\frac{\partial L}{\partial \vec{\theta}} + 2\lambda \mathbf{S}\delta\vec{\theta} \tag{248}$$

$$0 = 2\frac{\partial L}{\partial \vec{\theta}} + 2\lambda \mathbf{S}\delta\vec{\theta} \tag{249}$$

$$\delta\vec{\theta} \propto \mathbf{S}^{-1}\frac{\partial L}{\partial \vec{\theta}} \tag{250}$$

With the use of a stochastic trace estimator, the matrix $\mathbf{S}$ can be computed at a cost of $O(DNMI)$.

Alpha gradients

The second metric we propose is the normed distance of $\vec{\alpha}$. While this does not capture anything about posterior variances, it is more directly related to final quantities of the GP than the Gram matrix. The derivation follows almost exactly that of Section **??**, but with the distance metric replaced by:

$$d(\vec{\theta}_a, \vec{\theta}_b) = \left| \mathbf{K}(\theta_a)^{-1}\vec{y} - \mathbf{K}(\theta_b)^{-1}\vec{y} \right|^2 \tag{251}$$

$$\begin{aligned} \arg\max_{\delta\theta} \quad & L(\theta + \delta\theta) \\ \text{subject to} \quad & \left| \mathbf{K}(\theta)^{-1}\vec{y} - \mathbf{K}(\theta + \delta\theta)^{-1}\vec{y} \right|^2 < \epsilon \end{aligned} \tag{252}$$

Which is solved by the matrix:

$$\mathbf{T}_{ij} := \vec{y}^\intercal \left(\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta_j}\mathbf{K}^{-1}\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta_i}\mathbf{K}^{-1} \right) \vec{y} \tag{253}$$

$$\delta\theta \propto \mathbf{T}^{-1}\frac{\partial L}{\partial \theta} \tag{254}$$

Construction of $\mathbf{T}$ requires $O(DNMI)$ operations. While this is notationally the same as the complexity for Gram gradients, the I here refers to the number of iterations required for a linear system solve, rather than a trace estimate, which is likely to be larger.

Predictive gradients

Another meaningful metric of distance between two GPs is the difference in the posterior mean at the training points.

$$d\left(\vec{\theta}_a, \vec{\theta}_b\right) = \left| \mathbf{K}_-(\vec{\theta}_a)\mathbf{K}(\vec{\theta}_a)^{-1}\vec{y} - \mathbf{K}_-(\vec{\theta}_b)\mathbf{K}(\vec{\theta}_b)^{-1}\vec{y} \right|^2 \tag{255}$$

Using the same style of derivation again, this results in the pre-conditioner $\mathbf{M}$ where:

$$\mathbf{M}_{ij} = \sigma^4 \vec{y}^\intercal \mathbf{K}^{-1} \frac{d\mathbf{K}}{d\vec{\theta}_i} \mathbf{K}^{-1} \mathbf{K}^{-1} \frac{d\mathbf{K}}{d\vec{\theta}_j} \mathbf{K}^{-1} \vec{y} \tag{256}$$

This is almost exactly the same pre-conditioner as that derived from $\vec{\alpha}$, with the addition of a σ^4 term. This forces the step-size to be proportional to the square of the noise level. As before, this derivative can be calculated in $O(DNMI)$.

Natural predictive gradients

The final distance metric we suggest is natural predictive gradients. This has the benefit of incorporating information about the variances, without the excessive cost of natural gradients on the posterior. The metric is:

$$d\left(\vec{\theta}_a, \vec{\theta}_b\right) = KL\left(p\left(\vec{y} \mid \vec{\mu}_a, \vec{\sigma}_a^2\right), p\left(\vec{y} \mid \vec{\mu}_b, \vec{\sigma}_b^2\right)\right) \tag{257}$$

Where $\vec{\sigma}_a^2$ and $\vec{\sigma}_b^2$ are the predictive variances for parameters $\vec{\theta}_a$ and $\vec{\theta}_b$. Using a KL-divergence as a distance metric results in the pre-conditioner being the Fisher Information matrix, which has the following form for multivariate Gaussians:

$$\mathcal{I}_{ij} = \frac{\partial \vec{\mu}}{\partial \theta_i}^\intercal \boldsymbol{\Sigma}^{-1} \frac{\partial \vec{\mu}}{\partial \theta_j} + \frac{1}{2} Tr\left(\boldsymbol{\Sigma}^{-1} \frac{\partial \boldsymbol{\Sigma}}{\partial \theta_i} \boldsymbol{\Sigma}^{-1} \frac{\partial \boldsymbol{\Sigma}}{\partial \theta_j}\right) \tag{258}$$

Since it is for the predictive distribution, the posterior variance matrix $\boldsymbol{\Sigma}$ is a diagonal matrix where $\boldsymbol{\Sigma}_{ii}$ is the posterior variance for point i. We will derive this for a subset of variance approximation, which is the approximate posterior from variational derivation, but the iterative bounds from Section 4 can also be easily used instead. This allows the matrix to be calculated in $O(DNM(I+M))$.

6.2.3 *Comparing pre-conditioners*

Here we perform a preliminary comparison between the different pre-conditioners, showing that they are worthy of further investigation. To compare pre-conditioners we perform steepest descent with a line search on the standard objective function to convergence and record the number of required iterations, averaged over 5 random initializations, that are required using different gradients. From Seciton 6.2.1 we know that the condition number is proportional to the number of iterations required for steepest descent and that the condition number dictates the convergence rate for other methods, such as conjugate gradient.

We can see from Table 14 and Table 15 that the number of iterations is improved by different pre-conditioners. In all cases the optimal parameters found across the 5 runs were the same, except for Gram gradients in Table 14, which did not converge due to numerical issues. These preliminary results show that there is potential benefit for using such pre-conditioners and we conjecture that for kernels with more complicated structure and parameterization, that this will be substantially more important.

	Standard	Gram	Alpha	Predictive	Natural pred.
MPG	677	565	414	474	481
Boston housing	471	564	537	355	276
Australian beer	841	205	846	202	683
Sea levels	709	445	631	245	563
Bodyfat	705	649	342	373	501
Daily min temp	750	491	656	352	463
Prod of H_2SO_4	903	122	845	203	635

Table 14: Average number of iterations required for SE-ARD kernel

	Standard	Gram	Alpha	Predictive	Natural pred.
MPG	425	NA	265	143	231
Boston housing	309	NA	269	258	81
Australian beer	187	NA	186	180	177
Sea levels	184	NA	191	185	175
Bodyfat	646	NA	220	135	124
Daily min temp	396	NA	168	248	198
Prod of H_2SO_4	187	NA	188	176	170

Table 15: Average number of iterations required for FITC + Exponentiated Wendland 3 kernel

7

CONCLUSION

This thesis has made the argument that for most applications, and certainly for those involving larger datasets, the most effective framework for implementing Gaussian processes is based on iterative procedures.

We have also reframed the standard usage of Gaussian processes for regression from:

> Optimize the hyper-parameters of a kernel using a gradient descent method and then calculate the predictive distribution on test points.

To the following:

> Optimize hyper-parameters of a kernel *of controllable complexity* using a *well pre-conditioned* gradient method and then calculate the predictive distribution on test points *to the required precision*.

7.1 SUMMARY OF CONTRIBUTION

The main contribution of this thesis was the specification of a fully iterative framework for GP regression, building on the work of MacKay in Chapter 2. This included the introduction of termination criteria, introduction of new stochastic estimators and the understanding of approximate GP methods as pre-conditioners to optimization methods.

Secondly, the outlining of efficient kernels, including the specification of the class of graphical model kernels, to easily incorporate existing efficient kernel definitions, the introduction of random partition kernels and the introduction of global and local kernels, which includes the construction of the exponentiated Wendland family.

Finally, the assessment of different optimizing objective functions for efficient optimization of GPs and a generalized definition of pre-conditioning, emphasizing its importance in a joint optimization framework for GPs.

7.2 RELATIONSHIP TO EXISTING WORK

This framework is an extension and expansion of the GP framework of [Gibbs, 1997]. As referenced in the appropriate sections, there are similarities with other,

less general frameworks such as [Freytag et al., 2013]. Much of the extended literature on sparse or computationally efficient GPs can be used directly within this framework, as either a particular M-efficient kernel (Chapter 5) or as a pre-conditioner (Section 4.7). This includes spectral techniques, state space models, inducing-point approximations, approximate matrix decompositions, random projections and many others.

7.3 FUTURE WORK

While this thesis has laid a solid foundation for the implementation of effective solvers for Gaussian process regression, there are several further avenues that would result in more effective solvers:

1. **Improved pre-conditioners**. This thesis has argued strongly that the use of pre-conditioners is important for optimization of GPs — both in finding solutions in small number of iterations, and with further work, hopefully finding good solutions in the highly non-convex optimization space of hyper-parameters. While we have proposed some sensible initial choices, the true value of pre-conditioning comes from those that are tailored to specific kernels and applications.

2. **Joint optimization and joint pre-conditioners**. The variational objective opens the way for joint optimization of $\vec{\theta}$ and $\vec{\alpha}$ jointly. We believe this could have potential benefits of simplifying and streamlining optimization, and that in this regime, joint pre-conditioners will be important to ensure that steps in $\vec{\theta}$ space are balanced with appropriate steps in $\vec{\alpha}$ space

3. **Cost sensitive optimization methods**. The experiments of Chapter 2 showed that there is a great variation in the cost to calculate derivatives at various points of the optimization space. Furthermore, in our experiments they tend to be very expensive in regions of low probability, which are not close to the final parameter values. Optimization methods that can avoid these regions may provide an alternative method to stochastic gradients to optimizing difficult functions.

BIBLIOGRAPHY

David J. Aldous. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII - 1983*, Lecture Notes in Mathematics, pages 1–198. Springer Berlin Heidelberg, 1 January 1985.

S Amari and S C Douglas. Why natural gradient? In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 2, pages 1213–1216 vol.2, May 1998.

M Anitescu, J Chen, and L Wang. A matrix-free approach for solving the parametric Gaussian process maximum likelihood problem. *SIAM J. Sci. Comput.*, 34(1):A240–A262, 2012.

Erlend Aune, Daniel P Simpson, and Jo Eidsvik. Parameter estimation in high dimensional Gaussian distributions. *Stat. Comput.*, 24(2):247–263, 1 March 2014.

The GPy Authors. GPy2014. `https://github.com/SheffieldML/GPy`, 2014.

Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *J. ACM*, 58(2):8, 1 April 2011.

Z Bai and G H Golub. Bounds for the trace of the inverse and the determinant of symmetric positive definite matrices. *Annals of Numerical Mathematics*, 1996.

A Banerjee, DB Dunson, and ST Tokdar. Efficient Gaussian process regression for large datasets. *Biometrika*, 2008.

R Barrett, M W Berry, T F Chan, J Demmel, J Donato, J Dongarra, V Eijkhout, R Pozo, C Romine, and H van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Society for Industrial and Applied Mathematics, 1994.

R P Barry and R Kelley. Monte Carlo estimates of the log determinant of large sparse matrices. *Linear Algebra Appl.*, 1999.

Michele Benzi. Preconditioning techniques for large linear systems: A survey. *J. Comput. Phys.*, 182(2):418–477, November 2002.

M Botsch and J A Nossek. Construction of interpretable radial basis function classifiers based on the random forest kernel. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 220–227, June 2008.

S Boyd and L Vandenberghe. *Convex optimization*. Cambridge University Press, 2009.

Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 1 October 2001.

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

Hugh A. Chipman, Edward I. George, and Robert E. McCulloch. BART: Bayesian additive regression trees. *Ann. Appl. Stat.*, 4(1):266–298, March 2010.

Arindam Choudhury, Prasanth B Nair, and Andy J Keane. A data parallel approach for Large-Scale Gaussian process modeling. In *Proc. the Second SIAM International Conference on Data Mining.*, 2002.

Adam Coates and Andrew Y Ng. The importance of encoding versus training with sparse coding and vector quantization. *Proceedings of the 28th International*, 2011.

Adam Coates and Andrew Y Ng. Learning feature representations with K-Means. In *Neural Networks: Tricks of the Trade*, Lecture Notes in Computer Science, pages 561–580. Springer Berlin Heidelberg, 1 January 2012.

Planck Collaboration. Planck 2013 results. i. overview of products and scientific results. 03 2013.

William Cook. World travelling salesman problem. `http://www.math.uwaterloo.ca/tsp/world/`. Accessed: 2025-9-14.

Lehel Csató. *Gaussian Processes - Iterative Sparse Approximations*. PhD thesis, 2002.

Lehel Csató and Manfred Opper. Sparse representation for Gaussian process models. In *Advances in Neural Information Processing Systems*, 2001.

N De Freitas and Yang Wang. Fast Krylov methods for n-body learning. *Adv. Neural Inf. Process. Syst.*, 18:251, 2006.

Pedro Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, October 2012.

Petros Drineas and MW Mahoney. On the nystr$\ddot{o}$m method for approximating a gram matrix for improved kernel-based learning. *J. Mach. Learn. Res.*, 6:2153–2175, 2005.

David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, 02 2013.

M Ester, HP Kriegel, J Sander, and X Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD*, 1996.

Yurij G Evtushenko. *Numerical Optimization Techniques*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

Gregory E Fasshauer. Compactly supported radial basis functions. In *Meshfree Approximation Methods with Matlab*.

Leslie Foster, Alex Waagen, Nabeela Aijaz, Michael Hurley, Apolonio Luis, Joel Rinsky, Chandrika Satyavolu, Michael J Way, Paul Gazis, and Ashok Srivastava. Stable and efficient Gaussian process calculations. *J. Mach. Learn. Res.*, 10:857–882, June 2009.

Yoav Freund and Robert E Schapire. A Decision-Theoretic generalization of On-Line learning and an application to boosting. *J. Comput. System Sci.*, 55(1): 119–139, August 1997.

Alexander Freytag, Erik Rodner, Paul Bodesheim, and Joachim Denzler. Rapid uncertainty computation with Gaussian processes and histogram intersection kernels. In *Computer Vision - ACCV 2012*, Lecture Notes in Computer Science, pages 511–524. Springer Berlin Heidelberg, 1 January 2013.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. In *Annals of Statistics*, 2000.

Marc G Genton. Classes of kernels for machine learning: A statistics perspective. *J. Mach. Learn. Res.*, 2:299–312, March 2002.

Mark Gibbs. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, University of Cambridge, 1997.

Mark Gibbs and David J C MacKay. Efficient implementation of Gaussian processes. Technical report, 1997.

Elad Gilboa, Yunus Saatçi, and John P Cunningham. Scaling multidimensional inference for structured Gaussian processes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30 September 2013.

L Greengard and J Strain. The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.

TL Griffiths and Z Ghahramani. The indian buffet process: An introduction and review. *J. Mach. Learn. Res.*, 2011.

N Hale, N Higham, and L Trefethen. Computing $a^{\alpha}, \log(a)$, and related matrix functions by contour integrals. *SIAM J. Numer. Anal.*, 46(5):2505–2523, 2008.

Bart Hamers, Johan A K Suykens., and Bart De Moor. Compactly supported RBF kernels for sparsifying the gram matrix in LS-SVM regression models. In *Artificial Neural Networks - ICANN 2002*, pages 720–726. Springer Berlin Heidelberg, 1 January 2002.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. 26 September 2013.

R Herbrich. *Learning Kernel Classifiers: Theory and Algorithms*. Adaptive computation and machine learning. MIT Press, 2002.

Matt Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. 29 June 2012.

Antti Honkela, Matti Tornio, Tapani Raiko, and Juha Karhunen. Natural conjugate gradient in variational inference. In *Neural Information Processing*, Lecture Notes in Computer Science, pages 305–314. Springer Berlin Heidelberg, 1 January 2008.

Alexander Ihler. An overview of fast multipole methods. Technical report, 2004.

MI Jordan and Y Weiss. Probabilistic inference in graphical models. *Handbook of neural networks and brain theory*, (510), 2002.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Mach. Learn.*, 37 (2):183–233, 1 November 1999.

D Koller and N Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive computation and machine learning. MIT Press, 2009.

Sanjiv Kumar, M Mohri, and A Talwalkar. Sampling techniques for the nystr$^{\circ}$om method. In *Conference on Artificial Intelligence and Statistics*, volume 5, 2009.

D Lang, M Klaas, and N de Freitas. Empirical testing of fast kernel density estimation algorithms. *UBC Technical repor*, 2005.

Pierre Latouche. *Distributed Machine Learning*. PhD thesis, 2007.

Neil Lawrence and Ralf Herbrich. A sparse bayesian compression scheme - the informative vector machine. In *Presented at NIPS 2001 Workshop on Kernel Methods*, 2001.

Miguel Lázaro-Gredilla, Joaquin Quiñonero Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal. Sparse spectrum Gaussian process regression. *J. Mach. Learn. Res.*, 11:1865–1881, August 2010.

Q Le, T Sarlos, and A Smola. Fastfood-computing hilbert space expansions in loglinear time. *Proceedings of the 30th*, 2013.

F Lindgren, H Rue, and J Lindström. An explicit link between Gaussian fields and Gaussian markov random fields: the stochastic partial differential equation approach. *J. R. Stat. Soc.*, pages 423–498, 2011.

James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua B Tenenbaum, and Zoubin Ghahramani. Automatic construction and Natural-Language description of nonparametric regression models. 18 February 2014.

David MacKay. Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 1998.

David MacKay. K-means. In *Information Theory, Inference and Learning Algorithms*, pages 285–287. Cambridge University Press, 2003a.

David MacKay. Gaussian processes. In *Information Theory, Inference and Learning Algorithms*, pages 549–550. Cambridge University Press, 2003b.

Georges Matheron. Principles of geostatistics. *Econ. Geol.*, 58(8):1246–1266, 1 December 1963.

M McCourt. A stochastic simulation for approximating the log-determinant of a symmetric positive definite matrix. 2008.

Bryan S Morse, Terry S Yoo, Penny Rheingans, David T Chen, and K R Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

Iain Murray. Gaussian processes and fast matrix-vector multiplies. 2009.

A Naish-Guzman and S Holden. The generalized FITC approximation. *Adv. Neural Inf. Process. Syst.*, 20:1057–1064, 2007.

Anthony O'Hagan and JFC Kingman. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–42, 1978.

Stephen Malvern Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

Beng Chin Ooi, Ken J McDonell, and Ron Sacks-Davis. Spatial kd-tree: An indexing mechanism for spatial databases. In *IEEE COMPSAC*, volume 87, page 85, 1987.

Jim Pitman and Marc Yor. The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *Ann. Probab.*, 25(2):855–900, April 1997.

Joaquin Quiñonero Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *J. Mach. Learn. Res.*, 6:1939–1959, December 2005.

A Rahimi and B Recht. Random features for large-scale kernel machines. *Adv. Neural Inf. Process. Syst.*, 2007.

Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In D Koller, D Schuurmans, Y Bengio, and L Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1313–1320. Curran Associates, Inc., 2009.

Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (GPML) toolbox. *J. Mach. Learn. Res.*, 11:3011–3015, December 2010.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.

ND Ratliff and JA Bagnell. Kernel conjugate gradient for fast kernel machines. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007.

VC Raykar and Ramani Duraiswami. Fast large scale Gaussian process regression using approximate matrix-vector products. *Yale Workshop Adapt. Learn. Syst.*, 2007.

Arnold Reusken. Approximation of the determinant of large sparse symmetric positive definite matrices. 10 August 2000.

W Ring and B Wirth. Optimization methods on riemannian manifolds and their application to shape space. *SIAM J. Optim.*, 22(2):596–627, 2012.

Farbod Roosta-Khorasani and Uri Ascher. Improved bounds on sample size for implicit matrix trace estimators. 12 August 2013.

N L Roux and A W Fitzgibbon. A fast natural Newton method. *Proceedings of the 27th International Conference in Machine Learning*, 2010.

Daniel M Roy and Yee Whye Teh. The Mondrian process. In *Adv. Neural Information Processing Systems 21 (NIPS)*, 2009.

H Rue and H Tjelmeland. Fitting Gaussian markov random fields to Gaussian fields. *Scand. Stat. Theory Appl.*, 29:31–49, 2002.

Yousef Saad. *Iterative Methods for Sparse Linear Systems, Second Edition*. PWS, 2003.

Yunus Saatci. *Scalable Inference for Structured Gaussian Process Models*. PhD thesis, University of Cambridge.

S Särkkä, A Solin, and J Hartikainen. Spatio-Temporal learning via Infinite-Dimensional bayesian filtering and smoothing. *IEEE Signal Process. Mag.*, 2013.

Hiroyuki Sato and Toshihiro Iwai. A new, globally convergent riemannian conjugate gradient method. 1 February 2013.

M Seeger. Skilling techniques for bayesian analysis. *Technical Notes, University of Edinburgh*, 2000.

M Seeger. *Bayesian Gaussian process models: PAC-Bayesian generalisation error bounds and sparse approximations*. PhD thesis, 2003.

J Shawe-Taylor and N Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

Ori Shental, Paul H Siegel, Jack K Wolf, Danny Bickson, and Danny Dolev. Gaussian belief propagation solver for systems of linear equations. 09 October 2008.

J R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, 1994.

John Skilling. *Physics and Probability: Bayesian Numerical Analysis*. 1993.

Ed Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. *Adv. Neural Inf. Process. Syst.*, 18:1257, 2006.

Edward Snelson and Zoubin Ghahramani. Local and global sparse Gaussian process approximations. *Artificial Intelligence and Statistics*, 2007.

DA Spielman and SH Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004.

B V Srinivasan, Q Hu, N A Gumerov, and R Duraiswami. Preconditioned Krylov solvers for kernel regression. Technical report.

Michael L Stein, Jie Chen, and Mihai Anitescu. Stochastic approximation of score functions for Gaussian processes. *Ann. Appl. Stat.*, 7(2):1162–1191, June 2013.

E B Sudderth, M J Wainwright, and A S Willsky. Embedded trees: estimation of Gaussian processes on graphs with cycles. *Signal Processing, IEEE Transactions on*, 52(11):3136–3150, November 2004.

Michalis Titsias. Variational learning of inducing variables in sparse Gaussian processes. *Journal of Machine Learning Research*, April 2009a.

Michalis Titsias. Variational model selection for sparse Gaussian process regression. Technical report, 2009b.

Top500.org. Top 500 supercomputer sites, 3 October 2014. Accessed: 2014-10-3.

L N Trefethen and D Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

Volker Tresp. A bayesian committee machine. *Neural Comput.*, 12(11):2719–2741, 2000.

Jarno Vanhatalo and Aki Vehtari. Modelling local and global phenomena with sparse Gaussian processes. 2012.

Christian Walder, Kwang In Kim, and Bernhard Sch$\ddot{}$olkopf. Sparse multiscale Gaussian process regression. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 1112–1119. ACM Press, 2008.

Y Weiss and W T Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Comput.*, 13(10):2173–2200, October 2001.

Holger Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Adv. Comput. Math.*, 4(1):389–396, 1 December 1995.

Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, 2001.

Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian process kernels for pattern discovery and extrapolation. 18 February 2013.

David H. Wolpert. The supervised learning no-free-lunch theorems. In Rajkumar Roy, Mario Köppen, Seppo Ovaska, Takeshi Furuhashi, and Frank Hoffmann, editors, *Soft Computing and Industry*, pages 25–42. Springer London, 2002. ISBN 978-1-4471-1101-6. doi: 10.1007/978-1-4471-0123-9_3. URL `http://dx.doi.org/10.1007/978-1-4471-0123-9_3`.

Changjiang Yang, R Duraiswami, NA Gumerov, and L Davis. Improved fast gauss transform and efficient kernel density estimation. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 664–671 vol.1, 2003.

Y Zhang and W E Leithead. Approximate implementation of the logarithm of the matrix determinant in Gaussian process regression. *J. Stat. Comput. Simul.*, 77 (4):329–348, 2007.

S Zhu. Compactly supported radial basis functions: how and why? *SIAM Rev.*, 2012.