Preconditioning Kernel Matrices

Kurt Cutajar KURT.CUTAJAR@EURECOM.FR
EURECOM, Department of Data Science

Michael A. Osborne MOSB@ROBOTS.OX.AC.UK
University of Oxford, Department of Engineering Science

John P. Cunningham JPC2181@COLUMBIA.EDU
Columbia University, Department of Statistics

Maurizio Filippone MAURIZIO.FILIPPONE@EURECOM.FR
EURECOM, Department of Data Science

Abstract

The computational and storage complexity of kernel machines presents the primary barrier to their scaling to large, modern, datasets. A common way to tackle the scalability issue is to use the conjugate gradient algorithm, which relieves the constraints on both storage (the kernel matrix need not be stored) and computation (both stochastic gradients and parallelization can be used). Even so, conjugate gradient is not without its own issues: the conditioning of kernel matrices is often such that conjugate gradients will have poor convergence in practice. Preconditioning is a common approach to alleviating this issue. Here we propose preconditioned conjugate gradients for kernel machines, and develop a broad range of preconditioners particularly useful for kernel matrices. We describe a scalable approach to both solving kernel machines and learning their hyperparameters. We show this approach is exact in the limit of iterations and outperforms state-of-the-art approximations for a given computational budget [1].

1. Introduction

Kernel machines, in enabling flexible feature space representations of data, comprise a broad and important class of tools throughout machine learning and statistics; prominent examples include support vector machines (Schölkopf & Smola, 2001) and Gaussian processes (GPs) (Rasmussen & Williams, 2006). At the core of most kernel machines is a need to solve linear systems involving the Gram matrix $K = \{k(\mathbf{x}_i, \mathbf{x}_j \mid \boldsymbol{\theta})\}_{i,j=1,\dots,n}$, where the kernel function k, parameterized by $\boldsymbol{\theta}$, implicitly specifies the feature space representation of data points $\mathbf{x}_i$. Because K grows with the number of data points n, a fundamental computational bottleneck exists: storing K is $\mathcal{O}(n^2)$, and solving a linear system with K is $\mathcal{O}(n^3)$. As the need for large-scale kernel machines grows, much work has been directed towards this scaling issue.

Standard approaches to kernel machines involve a factorization (typically Cholesky) of K, which is efficient and exact but maintains the quadratic storage and cubic runtime costs. This cost is particularly acute when adapting (or learning) hyperparameters $\boldsymbol{\theta}$ of the kernel function, as K must then be factorized afresh for each $\boldsymbol{\theta}$. To alleviate this burden, numerous works have turned to approximate methods (Candela & Rasmussen, 2005; Snelson & Ghahramani, 2007; Rahimi & Recht, 2008) or methods that exploit structure in the kernel (Gilboa et al., 2015). Approximate methods can achieve attractive scaling, often through the use of low-rank approximations to K, but they can incur a potentially severe loss of accuracy. An alternative to factorization is found in the conjugate gradient method (CG), which is used to directly solve linear systems via a sequence of matrix-vector products. The kernel structure is then used to enable fast multiplications, driving similarly attractive runtime improvements, and eliminating the storage burden (neither K nor its factor need be represented in memory). Unfortunately, in the absence of special structure that accelerates multiplications, CG performs no better than $\mathcal{O}(n^3)$ in the worst case, and in practice finite numerical precision often results in a degradation of runtime performance compared to a naïve approach.

[1] Code to replicate all results in this paper is available at http://github.com/mauriziofilippone/preconditioned_GPs

Throughout optimization, the typical approach to the slow convergence of CG is to apply preconditioners to improve the geometry of the linear system being solved (Golub & Van Loan, 1996). While preconditioning is well-known and can converge in drastically fewer iterations than CG, the application of preconditioning to kernel matrices has received surprisingly little attention. Here we design and study preconditioned conjugate gradient methods (PCG) for use in kernel machines, and we provide a full exploration of the use of approximations of K as preconditioners. Our contributions are as follows. (i) Extending the work in (Davies, 2014), we apply a broad range of kernel matrix approximations as preconditioners, including the Nyström method (Williams & Seeger, 2000), partially- and fully-independent training conditional methods (PITC and FITC) (Snelson & Ghahramani, 2005), randomized partial singular value decomposition (SVD) (Halko et al., 2011), inner conjugate gradients with regularization (Srinivasan et al., 2014), random Fourier features (Rahimi & Recht, 2008; Lázaro-Gredilla et al., 2010) and structured kernel interpolation (SKI) (Wilson & Nickisch, 2015). Interestingly, this step allows us to exploit the important developments of *approximate* kernel machines to accelerate the *exact* computation that PCG offers. (ii) As a motivating example used throughout the paper, we analyze and provide a general framework to both learn kernel parameters and make predictions in GPs. (iii) We extend stochastic gradient learning for GPs (Filippone & Engler, 2015; Anitescu et al., 2012) to allow any likelihood that factorizes over the data points by developing an unbiased estimate of the gradient of the approximate log-marginal likelihood. We demonstrate this contribution in making the first use of PCG for GP classification. (iv) We evaluate datasets over a range of problem size and dimensionality. Because PCG is exact in the limit of iterations (unlike approximate techniques), we demonstrate a tradeoff between accuracy and computational effort that improves beyond state-of-the-art approximation and factorization approaches.

In all, we show that PCG, with a thoughtful choice of preconditioner, is a competitive strategy which is possibly even superior than existing approximation and CG-based techniques for solving general kernel machines.

2. Motivating example – Gaussian Processes

Gaussian processes (GPs) are the fundamental building block of many probabilistic kernel machines that can be applied in a large variety of modeling scenarios (Rasmussen & Williams, 2006). Throughout the paper, we will denote by $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ a set of n input vectors and use $\mathbf{y} = (y_1, \dots, y_n)^\top$ for the corresponding labels. GPs are formally defined as collections of random variables characterized by the property that any finite number of them is jointly Gaussian distributed. The specification of a kernel function determines the covariance structure of such random variables

$$\operatorname{cov}\big(f(\mathbf{x}), f(\mathbf{x}')\big) = k(\mathbf{x}, \mathbf{x}' \mid \boldsymbol{\theta}).$$

In this work we focus in particular on the popular Radial Basis Function (RBF) kernel

$$k(\mathbf{x}_i, \mathbf{x}_j \mid \boldsymbol{\theta}) = \sigma^2 \exp\left[-\frac{1}{2}\sum_{r=1}^{d}\frac{(\mathbf{x}_i - \mathbf{x}_j)_r^2}{l_r^2}\right], \qquad (1)$$

where $\boldsymbol{\theta}$ represents the collection of all kernel parameters. Defining $f_i = f(\mathbf{x}_i)$ and $\mathbf{f} = (f_1, \dots, f_n)^\top$, and assuming a zero mean GP for the sake of simplicity, we have

$$\mathbf{f} \sim \mathcal{N}(\mathbf{f} \mid \mathbf{0}, K),$$

where K is the $n \times n$ Gram matrix with elements $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j \mid \boldsymbol{\theta})$. Note that the kernel above and many popular kernels in machine learning give rise to dense kernel matrices. Observations are then modeled through a transformation h of a set of GP-distributed latent variables, specifying the model

$$y_i \sim p\big(y_i \mid h(f_i)\big), \qquad \mathbf{f} \sim \mathcal{N}(\mathbf{f} \mid \mathbf{0}, K).$$

2.1. The need for preconditioning

The success of nonparametric models based on kernels hinges on the adaptation of kernel parameters $\boldsymbol{\theta}$. The motivation for preconditioning begins with an inspection of the log-marginal likelihood of GP models with prior $\mathcal{N}(\mathbf{f} \mid \mathbf{0}, K)$. In Gaussian processes with a Gaussian likelihood $y_i \sim \mathcal{N}(y_i \mid f_i, \lambda)$, we have analytic forms for

$$\log[p(\mathbf{y} \mid \boldsymbol{\theta}, X)] = -\frac{1}{2}\log\left(|K_\mathbf{y}|\right) - \frac{1}{2}\mathbf{y}^\top K_\mathbf{y}^{-1}\mathbf{y} + \text{const},$$

and its derivatives with respect to kernel parameters θ_i,

$$g_i = -\frac{1}{2}\operatorname{Tr}\left(K_\mathbf{y}^{-1}\frac{\partial K_\mathbf{y}}{\partial \theta_i}\right) + \frac{1}{2}\mathbf{y}^\top K_\mathbf{y}^{-1}\frac{\partial K_\mathbf{y}}{\partial \theta_i}K_\mathbf{y}^{-1}\mathbf{y}. \quad (2)$$

where $K_\mathbf{y} = K + \lambda I$. Traditionally, these calculations are carried out exactly. We first factorize the kernel matrix $K_\mathbf{y} = LL^\top$ using the Cholesky algorithm (Golub & Van Loan, 1996) which costs $\mathcal{O}(n^3)$ operations. After that, all other operations cost $\mathcal{O}(n^2)$ except for the trace term in the calculation of g_i that requires again $\mathcal{O}(n^3)$ operations.

This approach is not viable for large n. As such, many approaches have been proposed to approximate these computations, thus leading to approximate optimal values for $\boldsymbol{\theta}$ and approximate predictions. Here we investigate the possibility of avoiding approximations altogether, by arguing that for parameter optimization it is sufficient to obtain an unbiased estimate of the gradient g_i. In particular,

when such an estimate is available, it is possible to employ stochastic gradient optimization that has strong theoretical guarantees (Robbins & Monro, 1951). In the case of GPs, the problematic terms in eq. 2 are the solution of the linear system $K_{\mathbf{y}}^{-1}\mathbf{y}$ and the trace term. In this work we make use of a simple result in stochastic linear algebra that allows for an approximation of the trace term,

$$\mathrm{Tr}\left(K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \theta_i}\right) \approx \frac{1}{N_{\mathbf{r}}}\sum_{i=1}^{N_{\mathbf{r}}} \mathbf{r}^{(i)\top} K_{\mathbf{y}}^{-1}\frac{\partial K_{\mathbf{y}}}{\partial \theta_i}\mathbf{r}^{(i)},$$

where the $N_{\mathbf{r}}$ vectors $\mathbf{r}^{(i)}$ have components drawn from $\{-1, 1\}$ with probability $1/2$. Verifying that this is an unbiased estimate of the trace term is straightforward considering that $\mathrm{E}(\mathbf{r}^{(i)}\mathbf{r}^{(i)\top}) = I$ (Gibbs, 1997).

This result shows that all it takes to calculate stochastic gradients is the ability to solve linear systems. Linear systems can be iteratively solved using *conjugate gradient* (CG) (Golub & Van Loan, 1996). The advantage of this formulation is that we can attempt to optimize kernel parameters using stochastic gradient optimization without having to store $K_{\mathbf{y}}$ and requiring only $\mathcal{O}(n^2)$ computation, given that the most expensive operation is now multiplying the kernel matrix by vectors. However, it is well known that the convergence of the CG algorithm depends on the condition number $\kappa(K_{\mathbf{y}})$ (ratio of largest to smallest eigenvalues), so the suitability of this approach may also be curtailed if $K_{\mathbf{y}}$ is badly conditioned. To this end, a well-known approach for improving the conditioning of a matrix is *preconditioning*. This technique can be incorporated into the CG algorithm by transforming the linear system to be better-conditioned, improving convergence. This necessitates the introduction of a preconditioning matrix, P, which can be chosen so as to simplify solving by having $P^{-1}K_{\mathbf{y}}$ approximate the identity matrix, I. Intuitively, this can be obtained by setting $P = K_{\mathbf{y}}$; however, given that we are required to solve $P^{-1}\mathbf{v}$, this choice would be no easier than solving the original system. Thus we must choose a P which approximates $K_{\mathbf{y}}$ as closely as possible, but which can also be easily inverted. The PCG algorithm is shown in Algorithm 1.

Algorithm 1 The Preconditioned CG Algorithm

Require: data X, vector $\mathbf{v}$, convergence threshold ϵ, initial vector $\mathbf{x}_0$, maximum no. of iterations T

$\mathbf{r}_0 = \mathbf{v} - K_{\mathbf{y}}\mathbf{x}_0$; $\mathbf{z}_0 = P^{-1}\mathbf{r}_0$; $\mathbf{p}_0 = \mathbf{z}_0$

for i = 0 : T **do**

  $\alpha_i = \frac{\mathbf{r}_i^T\mathbf{z}_i}{\mathbf{r}_i^T K_{\mathbf{y}}\mathbf{z}_i}$

  $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i\mathbf{p}_i$

  $\mathbf{r}_{i+1} = \mathbf{r}_i + \alpha_i K_{\mathbf{y}}\mathbf{p}_i$

  if $\|\mathbf{r}_{i+1}\| < \epsilon$ **then**

    return $\mathbf{x} = \mathbf{x}_{i+1}$

  end if

  $\mathbf{z}_{i+1} = P^{-1}\mathbf{r}_{i+1}$

  $\beta_i = \frac{\mathbf{r}_{i+1}^T\mathbf{r}_{i+1}}{\mathbf{r}_i^T\mathbf{r}_i}$

  $\mathbf{p}_{i+1} = \mathbf{p}_{i+1} + \beta_i\mathbf{p}_i$

end for

2.2. Non-Gaussian Likelihoods

We present the first approach to PCG for GPs with non-Gaussian likelihoods. When the likelihood $p(y_i \mid f_i)$ is not Gaussian, it is no longer possible to analytically integrate out latent variables. Several methods for countering this issue have been proposed, from Gaussian approximation (see, e.g., (Kuss & Rasmussen, 2005; Nickisch & Rasmussen, 2008)) to methods that attempt to characterize the full posterior $p(\mathbf{f}, \boldsymbol{\theta} \mid \mathbf{y})$ (Murray et al., 2010; Filippone et al., 2013). Among the various schemes to recover tractability in the case of models with a non-Gaussian likelihood, we choose the Laplace approximation, as we can easily formulate it in a way that only requires the solution of linear systems. The GP models we consider assume that the likelihood factorizes across all data points $p(\mathbf{y} \mid \mathbf{f}) = \prod_{i=1}^{n} p(y_i \mid f_i)$ and that the latent variables $\mathbf{f}$ are given a zero mean GP prior with kernel K.

Defining $W = -\nabla_{\mathbf{f}}\nabla_{\mathbf{f}} \log[p(\mathbf{y} \mid \mathbf{f})]$ (a diagonal matrix), carrying out the Laplace approximation algorithm, computing its derivatives wrt $\boldsymbol{\theta}$, and making predictions, all possess the same computational bottleneck: the solution of linear systems involving the matrix $B = I + W^{\frac{1}{2}}KW^{\frac{1}{2}}$. For a given $\boldsymbol{\theta}$, each iteration of the Laplace approximation algorithm requires solving one linear system involving B and two matrix-vector multiplications involving K; the linear system involving B can be solved using CG or PCG. The Laplace approximation yields the mode $\hat{\mathbf{f}}$ of the posterior over latent variables and offers an approximate log-marginal likelihood in the form:

$$\log[\hat{p}(\mathbf{y} \mid \boldsymbol{\theta}, X)] = -\frac{1}{2}\log|B| - \frac{1}{2}\hat{\mathbf{f}}^\top K^{-1}\hat{\mathbf{f}} + \log[p(\mathbf{y} \mid \hat{\mathbf{f}})]$$

which poses the same computational challenges as the regression case. Once again, we therefore seek an alternative way to learn kernel parameters by stochastic gradient optimization based on computing unbiased estimates of the gradient of the approximate log-marginal likelihood. This is complicated further by the inclusion of an additional "implicit" term accounting for the change in the solution given by the Laplace approximation for a change in $\boldsymbol{\theta}$. The full derivation of the gradient is rather lengthy and is deferred to the supplementary material. Nonetheless, it is worth noting that the calculation of the exact gradient involves trace terms similar to the regression case that cannot be computed for large n, and we unbiasedly approximate these using the stochastic approximation of the trace.

3. Preconditioning Kernel Matrices

Here we consider choices for kernel preconditioners. Unless stated otherwise, we shall consider standard *left* preconditioning, whereby the original problem of solving $K_{\mathbf{y}}\mathbf{x} = \mathbf{v}$ is transformed by applying a preconditioner, P, to both sides of this equation. This formulation may thus be expressed as $P^{-1}K_{\mathbf{y}}\mathbf{x} = P^{-1}\mathbf{v}$.

3.1. Nyström type approximations

The Nyström method was originally proposed to approximate the eigendecomposition of kernel matrices (Williams & Seeger, 2000); as a result, it offers a way to obtain a low rank approximation of K. This method selects a subset of $m \ll n$ data (inducing points) stacked in U which are intended for approximating the spectrum of K. The resulting approximation is $\hat{K} = K_{XU}K_{UU}^{-1}K_{UX}$ where K_{UU} denotes the evaluation of the kernel function over the inducing points, and K_{XU} denotes the evaluation of the kernel function between the input points and the inducing points. The resulting preconditioner $P = K_{XU}K_{UU}^{-1}K_{UX} + \lambda I$ can be inverted using the matrix inversion lemma

$$P^{-1}\mathbf{v} = \lambda^{-1}\left[I - K_{XU}\left(K_{UU} + K_{UX}K_{XU}\right)^{-1}K_{UX}\right]\mathbf{v}$$

which has $\mathcal{O}(m^3)$ complexity.

3.1.1. Fully and Partially Independent Training Conditional

The use of a subset of data for approximating a GP kernel has also been utilized in the fully and partially independent training conditional approaches (FITC and PITC respectively) for approximating GP regression (Candela & Rasmussen, 2005). In the former case, the prior covariance of the approximation can be written as follows:

$$P = K_{XU}K_{UU}^{-1}K_{UX} + \mathbf{diag}\left(K - K_{XU}K_{UU}^{-1}K_{UX}\right) + \lambda I.$$

As the name implies, this formulation enforces that the latent variables associated with U are taken to be completely conditionally independent. On the other hand, the PITC method extends on this approach by enforcing that although inducing points assigned to a designated block are conditionally dependent on each other, there is no dependence between points placed in different blocks:

$$P = K_{XU}K_{UU}^{-1}K_{UX} + \mathbf{bldiag}\left(K - K_{XU}K_{UU}^{-1}K_{UX}\right) + \lambda I.$$

For the FITC preconditioner, the diagonal resulting from the training conditional can be added to the diagonal noise matrix, and the inversion lemma can be invoked as for the Nyström case. Meanwhile, for the PITC preconditioner, the noise diagonal can be added to the block diagonal matrix, which can then be inverted block-by-block. Once again, matrix inversion can then be carried out as before, where the inverted block diagonal matrix takes the place of λI in the original formulation.

3.2. Approximate factorization of kernel matrices

This group of preconditioners relies on approximations to K that factorize as $\hat{K} = \Phi\Phi^\top$. We shall consider different ways of determining Φ such that P can be inverted at a lower cost than the original kernel matrix K. Once again, this enables us to employ the matrix inversion lemma, and express the linear system:

$$P^{-1}\mathbf{v} = (\Phi\Phi^\top + \lambda I)^{-1}\mathbf{v} = \lambda^{-1}[I - \Phi(I + \Phi^\top\Phi)^{-1}\Phi^\top]\mathbf{v}.$$

We now review a few methods to approximate the kernel matrix K in the form $\Phi\Phi^\top$.

3.2.1. Spectral Approximation

The spectral approach is a data independent method which uses random Fourier features for deriving a sparse approximation of a GP. This approach was first introduced in (Lázaro-Gredilla et al., 2010), and relies on the assumption that stationary kernel functions can be represented as the Fourier transforms. As such, the kernel function can be expressed using this representation as follows:

$$\hat{K}_{ij} = \frac{\sigma_0^2}{m}\phi(\mathbf{x}_i)^{\mathrm{T}}\phi(\mathbf{x}_j) = \frac{\sigma_0^2}{m}\sum_{r=1}^{m}\cos\left[2\pi\mathbf{s}_r^\top(\mathbf{x}_i - \mathbf{x}_j)\right]$$

where In the equation above, the vectors $\mathbf{s}_r$ denote the *spectral points* (or *frequencies*), which are sampled from a Gaussian distribution approximating a designated kernel matrix. In the case of the RBF kernel, the frequencies can be sampled from $\mathcal{N}\left(\mathbf{0}, \frac{1}{4\pi^2}\Lambda\right)$, where $\Lambda = \left[1/l_1^2, \ldots, 1/l_n^2\right]$. To the best of our knowledge, this is the first time such an approximation has been considered for the purpose of preconditioning kernel matrices.

3.2.2. Partial SVD Factorization

Another factorization approach that we consider in this work is the partial singular value decomposition (SVD) method (Golub & Van Loan, 1996), which transforms the original kernel matrix K into the factorized form $A\Lambda A^\top$, where A is a unitary matrix and Λ is a diagonal matrix of singular values. The decomposition of K using this formulation yields a low-rank representation with $\Phi = A\Lambda^{1/2}$. In particular, we shall consider a variation of this technique called *randomized truncated* SVD (Halko et al., 2011), which constructs an approximation to the proper factorization of K using random sampling for determining which subspace of a matrix captures most of its variance.

3.2.3. Structured Kernel Interpolation

Some recent work on approximating GPs has exploited the fast computation of Kronecker matrix-vector multiplications when inputs are located on a Cartesian grid (Gilboa et al., 2015). Unfortunately, very few datasets meet this requirement, thus limiting the possibility of applying Kronecker inference in practice. To this end, SKI (Wilson & Nickisch, 2015) is an approximation technique which exploits the benefits of the Kronecker product without imposing any requirements on the structure of the training data. In particular, a grid of inducing points, U, is constructed, and the covariance between the training data and U is then represented as $K_{XU} = WK_{UU}$. In this formulation, W denotes a sparse interpolation matrix for assigning weights to the elements of K_{UU}. In this manner, a preconditioner exploiting Kronecker structure can be constructed as $P = WK_{UU}W^\top + \lambda I$. If we consider $V = W/\sqrt{\lambda}$, we can rewrite the (inverse) preconditioner as $P^{-1} = \lambda^{-1}(VK_{\text{UU}}V^\top + I)^{-1}$. Since this can no longer be solved directly, we solve this (inner-loop) linear system using the CG algorithm (all within one iteration of the outer-loop PCG). The presence of the identity matrix seems promising in making the system well conditioned; however, the conditioning of this matrix is similar to the original kernel matrix (as it should be if the approximation is accurate). As a result, for badly conditioned systems, although the complexity of the required matrix-vector multiplications is now much less than $\mathcal{O}(n^2)$, the number of iterations to solve linear systems involving the preconditioner is potentially very large, and could possibly diminish the benefits of preconditioning.

3.3. Other approaches

3.3.1. Block Jacobi

An alternative to using a single subset of data involves constructing local GPs over segments of the original data (Snelson & Ghahramani, 2007). An example of such an approach is the *Block Jacobi* approximation, whereby the preconditioner is constructed by taking a block diagonal of K and discarding all other elements in the kernel matrix. In this manner, covariance is only expressed for points within the same block, as $P = \textbf{bldiag}\left(K_{\mathbf{y}} + \lambda I\right)$.

The inverse of this block diagonal matrix is computationally cheap (also block diagonal). However, given that a substantial amount of information contained in the original covariance matrix is ignored, this choice is intrinsically a rather crude approach.

3.3.2. Regularization

An appealing feature shared by the aforementioned preconditioners (aside from SKI) is that their structure enables us to efficiently solve $P^{-1}\mathbf{v}$ directly. An alternative technique for constructing a preconditioner involves adding a positive regularization parameter, δI, to the original kernel matrix, such that $P = K_{\mathbf{y}} + \delta I$ (Srinivasan et al., 2014). This follows from the fact that adding noise to the diagonal of $K_{\mathbf{y}}$ makes it better-conditioned, and the condition number is expected to decrease further as δ increases. Nonetheless, for the purpose of preconditioning, this parameter should be tuned in such a way that P remains a sensible approximation of $K_{\mathbf{y}}$. As opposed to the previous preconditioners, this is an instance of *right* preconditioning, which has the following general form $K_{\mathbf{y}}P^{-1}(P\mathbf{x}) = \mathbf{v}$.

Given that it is no longer possible to evaluate $P^{-1}\mathbf{v}$ analytically, this linear system is solved yet again using CG, such that a linear system of equations is solved at every outer iteration of the PCG algorithm. Due to the potential loss of accuracy incurred while solving the inner linear systems, a variation of the standard PCG algorithm presented in algorithm 1, referred to as *flexible* PCG (Notay, 2000), is used instead. Using this approach, a re-orthogonalization step is introduced such that the search directions remain orthogonal even when the inner system is not solved precisely.

4. Comparison of Preconditioners

In this section, we provide an empirical exploration of these preconditioners in a practical setting. We consider three datasets for regression from the UCI repository (Asuncion & Newman, 2007), namely the Concrete dataset ($n = 1030, d = 8$), the Power Plant dataset ($n = 9568, d = 4$), and the Protein dataset ($n = 45730, d = 9$). In particular, we evaluate the convergence in solving $K_{\mathbf{y}}\mathbf{x} = \mathbf{y}$ using iterative methods, where $\mathbf{y}$ denotes the labels of the designated dataset, and $K_{\mathbf{y}}$ is constructed using different configurations of kernel parameters.

With this experiment, we aim to assess the quality of different preconditioners based on how many matrix-vector products they require, which, for most approaches, corresponds to the number of iterations taken by PCG to converge. The convergence threshold is set to $\epsilon^2 = n \cdot 10^{-10}$ so as to roughly accept an average error of 10^{-5} on each element of the solution.

For every variation, we set the parameters of the preconditioners so as to have a complexity lower than the $\mathcal{O}(n^2)$ cost associated with matrix-vector products; by doing so, we can assume that the latter computations are the dominant cost for large n. In particular, for Nyström-type methods, we set $m = \sqrt{n}$ inducing points, so that when we invert the preconditioner using the matrix inversion lemma, the cost is in $\mathcal{O}(m^3) = \mathcal{O}(n^{3/2})$. Similarly, for the Spectral preconditioner, we set $m = \sqrt{n}$ random features. For the Kronecker preconditioner, we take an equal

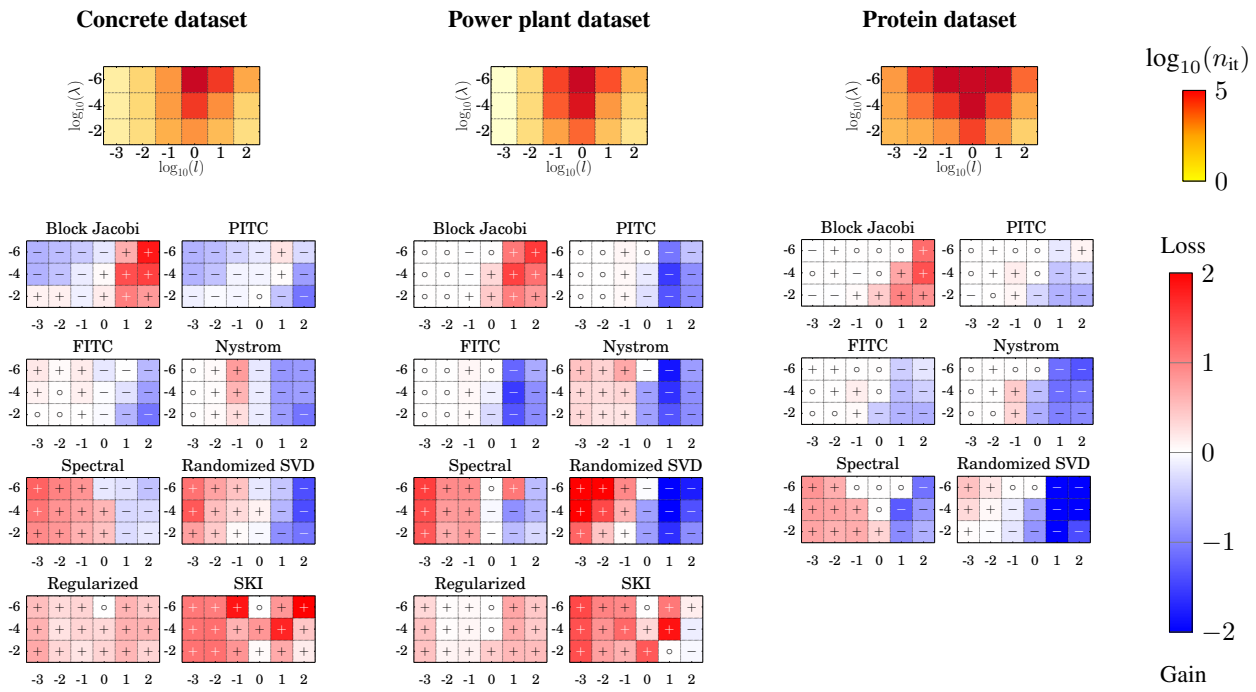

Figure 1. Comparison of preconditioners for different settings of kernel parameters. The lengthscale l and the noise variance λ are shown on the x and y axes respectively. The top figure indicates the number of iterations required to solve the corresponding linear system using CG, whilst the bottom part of the figure shows the rate of improvement (negative - blue) or degradation (positive - red) achieved by using PCG to solve the same linear system. Parameters and results are reported in $\log_{10}$. Symbols added to facilitate reading in B/W print.

number of elements on the grid for each dimension; under this assumption, Kronecker products have $\mathcal{O}(dn^{\frac{d+1}{d}})$ cost (Gilboa et al., 2015), and we set the size of the grid so that the complexity of applying the preconditioner matches $\mathcal{O}(n^{3/2})$, so as to be consistent with the other preconditioners. For the Regularized approach, each iteration needed to apply the preconditioner requires one matrix-vector product, and we add this to the overall count of such computations. For this equation, we add a diagonal offset to the original matrix, equivalent to two orders of magnitude greater than the noise of the process.

We focus on the isotropic RBF kernel (eq. 1), fixing the marginal variance σ^2 to one. This choice does not limit generality: σ^2 scales the condition number without altering the structure of the Gram matrix, and, in practice, we could solve a linear system by post-multiplying by σ^2. We vary the length-scale parameter l and the noise variance λ in $\log_{10}$ scale. The top part of fig. 1 shows the number of iterations that the standard CG algorithm takes, where we have capped the number of iterations to 15,000.

The bottom part of the figure reports the improvement offered by various preconditioners measured as

$$\log_{10}\left(\frac{n_{\text{it PCG}}}{n_{\text{it CG}}}\right).$$

It is worth noting that when both CG and PCG fail to converge within the upper bound, the improvement will be marked as 0, i.e. neither a gain or a loss within the given bound. The results plotted in fig. 1 indicate that the low-rank preconditioners (PITC, FITC and Nyström) achieve significant reductions in the number of iterations for each dataset, and all approaches work best when the lengthscale is longer, characterising smoother processes. In contrast, preconditioning seems to be less effective when the lengthscale is shorter, corresponding to a kernel matrix that is more sparse. However, for cases yielding positive results, the improvement is often in the range of an order of magnitude, which can be substantial for cases where a large number of iterations is required by the CG algorithm. Although all preconditioners perform similarly across different regions of the chosen grid, the Nyström method frequently perform better than the rest.

The results also confirm that, as alluded to in the previous section, Block Jacobi preconditioning is generally a poor preconditioner, particularly when the corresponding kernel matrix is dense. The only minor improvements were observed when CG itself converges quickly, in which case preconditioning serves very little purpose either way.

The regularization approach with flexible conjugate gradient does not appear to be effective in any case either, partic-

ularly due to the substantial amount of iterations required for solving an inner system at every iteration of the PCG algorithm. This implies that introducing additional small jitter to the diagonal does not necessarily make the system much easier to solve, whilst adding an overly large offset would negatively impact convergence of the outer algorithm. One could assume that tuning the value of this parameter (perhaps using cross-validation) could result in slightly better results; however, preliminary experiments in this regard resulted in only minor improvements.

The results for SKI preconditioning are similarly discouraging at face value. Given that inner matrix-vector products can exploit Kronecker structure, we permitted the upper limit of 15,000 outer iterations. However, it transpired that when the matrix $K_{\mathbf{y}}$ is very badly conditioned, an excessive number of inner iterations are required for every iteration of outer PCG. This greatly increases the duration of solving such systems, and as a result, this method was not included in the comparison for the Protein dataset, where it was evident that preconditioning the matrix in this manner would not yield satisfactory improvements. Notwithstanding that these experiments depict a negative view of Kronecker preconditioning, it must be said that we assumed a fairly simplistic interpolation procedure in our experiments, where each data point was mapped to nearest grid location. The size of the constructed grid is also hindered considerably by the constraint imposed by our upper bound on complexity. Conversely, more sophisticated interpolation strategies or even grid formulation procedures could possibly speed up the convergence of CG for the inner systems. In line with this thought, however, one could argue that constructing the preconditioner would no longer be very straightforward to construct, which goes against our innate preference towards preconditioners that can be more easily derived.

5. Impact of preconditioning on GP learning

One of the primary objectives of this work is to reformulate GP regression and classification in such a way that preconditioning can be effectively exploited. In section 2, we demonstrated how preconditioning can indeed be applied to GP regression problems, and also proposed a novel way of rewriting GP classification in terms of solving linear systems (where preconditioning can thus be employed). We can now evaluate how the proposed preconditioned GP techniques compare to other state of the art methods.

To this end, in this section, we empirically report on the generalization ability of GPs as a function of the time taken to optimize parameters $\boldsymbol{\theta}$ and compute predictions. In particular, for each of the methods featured in our comparison, we iteratively run the optimization of kernel parameters for a few iterations and predict on unseen data, and assess how prediction accuracy varies over time for different methods.

The analysis provided in this report is inspired by (Chalupka et al., 2013), although we do not propose an *approximate* method to learn GP kernel parameters. Instead, we put forward a means of accelerating the optimization of kernel parameters *without any approximation*[2]. Given predictive mean and variance for the N_{test} test points, say m_{*i} and s^2_{*i}, we report two error measures, namely the Root Mean Square Error, $\text{RMSE} = \sqrt{\frac{1}{N_{\text{test}}}\sum_{i=1}^{N_{\text{test}}}(m_{*i} - y_{*i})^2}$, along with the negative log-likelihood on the test data, $-\sum_{i=1}^{N_{\text{test}}} \log[p(y_{*i} \mid m_{*i}, s^2_{*i})]$, where y_{*i} denotes the label of the ith of N_{test} data points. For classification, instead of the RMSE we report the error rate of the classifier.

The fact that we can compute stochastic gradients for GP models enables us to use an off-the-shelf stochastic gradient optimization algorithm. In order to reduce the number of parameters to tune, we employ ADAGRAD (Duchi et al., 2011) – an optimisation algorithm having a single step-size parameter. For the purpose of this experiment, we do not attempt to optimize this parameter, since this would require additional computations. Nonetheless, our experience with training GP models indicates that the choice of this parameter is not critical: we set the step-size to one.

Fig. 2 shows the two error measures over time for a selection of approaches. In the figure, PCG and CG refer to stochastic gradient optimization of kernel parameters using ADAGRAD, where linear systems are solved with PCG and CG, respectively. In view of the results obtained in our comparison of preconditioners, we decide to proceed with the Nyström preconditioning method, such that the preconditioner is constructed with $m = 4\sqrt{n}$ points randomly selected from the set of input data at each iteration. For these methods, stochastic estimates of trace terms are carried out using $N_{\mathbf{r}} = 4$ random vectors. The baseline CHOL method refers to the optimization of kernel parameters using the L-BFGS algorithm, where the exact log-marginal likelihood and its gradient are calculated using the full Cholesky decomposition of $K_{\mathbf{y}}$ or B.

Alongside these approaches for optimizing kernel parameters without approximation, we also evaluate the performance of approximate GP methods. For this experiment, we chose to compare against the approximations found in the software package GPstuff (Vanhatalo et al., 2013), namely the fully and partial independent training conditional approaches (FITC, PITC), and the sparse variational GP (VAR) (Titsias, 2009). In order to match the computational cost of CG/PCG, which is in $\mathcal{O}(n^2)$, for the approximate methods we set the number of inducing points to be

[2]The one proviso to this statement is that stochastic gradients target the approximate log-marginal likelihood obtained by the Laplace approximation for non-Gaussian likelihood functions.

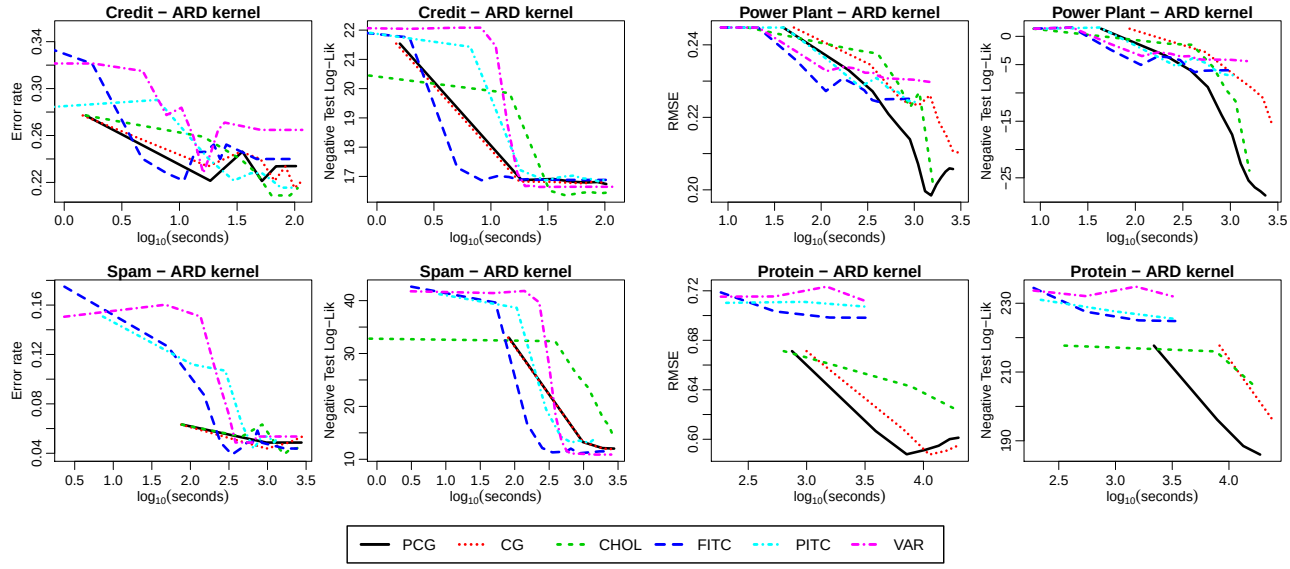

Figure 2. RMSE and negative log of the likelihood on $\sqrt{n}$ held out test data over time. GP models employ the ARD kernel in eq. 1. GP classification: Credit dataset ($n = 1000, d = 20$) and Spam dataset ($n = 4601, d = 57$). GP regression: Power Plant dataset ($n = 9568, d = 4$) and Protein dataset ($n = 45730, d = 9$). Curves are averaged over multiple repetitions.

$n^{2/3}$.

All methods are initialized from the same set of kernel parameters, and the curves are averaged over 5 folds (3 for the Protein and Spam datasets). For the sake of integrity, we ran each method in the comparison individually on a workstation with Intel Xeon E5-2630 CPU having 16 cores and 128GB RAM. We also ensured that all methods reported in the comparison used optimized linear algebra routines exploiting the multi-core architecture. This diligence for ensuring fairness gives credence to our assumption that the timings are not affected by external factors other than the actual implementation of the algorithms. The CG, PCG and CHOL approaches have been implemented in R; the fact that the approximate methods were implemented in a different environment and by a different developer may cast some doubt on the correctness of directly comparing results. However, we believe that the key point emerging from this comparison is that preconditioning feasibly enables the use of iterative approaches for optimization of kernel parameters in GPs, and the results are competitive with those achieved using popular GP software packages.

For the reported experiments, it was possible to store the kernel matrix K for all datasets, making it possible to compare methods against the baseline GP where computations use Cholesky decompositions. We stress, however, that iterative approaches based on CG/PCG can be implemented without the need to store K, whereas this is not possible for approaches that attempt to factorize K exactly. It is also worth noting that for the CG/PCG approach, calculating the log-likelihood on test data requires solving one linear system for each test point; this clearly penalizes the speed of these meth ods given the set-up of the experiment, where predictions are carried out every fixed number of iteration.

We make one final remark here on the fact that for GP classification the curves in fig. 2 for CG/PCG appear to be similar. Upon a closer inspection, we notice that this is due to the fact that most of the time is spent constructing K and its derivatives with respect to kernel parameters. PCG does allow for faster solution to the linear systems but this fact does not emerge from the time analysis.

6. Discussion and Conclusions

Careful attention to numerical properties is essential in scaling machine learning to large and realistic datasets. Here we have introduced the use of preconditioning to the implementation of kernel machines, specifically, prediction and learning of kernel parameters for GPs. Our novel scheme permits the use of any likelihood that factorizes over the data points, allowing us to tackle both regression and classification. We have shown robust performance improvements, in both accuracy and computational cost, over a host of state-of-the-art approximation methods for kernel machines. Notably, our method is exact in the limit of iterations, unlike approximate alternatives. We have also shown that the use of PCG is competitive with exact Cholesky decomposition in modestly sized datasets, when the Cholesky factors can be feasibly computed. When data and thus the kernel matrix grow large enough, the Cholesky factor becomes infeasible, leaving PCG as the optimal choice.

One of the key features of a PCG implementation is that it does not require storage of any $\mathcal{O}(n^2)$ objects. We plan to extend our implementation to compute the elements of K on the fly in one case, and in another case store K in a distributed fashion (e.g. in TensorFlow/Spark). Furthermore, while we have focused on solving linear systems, we can also use preconditioning for other iterative algorithms involving the K matrix, e.g., those to solve $\log(K)\mathbf{v}$ and $K^{1/2}\mathbf{v}$ (Chen et al., 2011), as is often useful in estimating marginal likelihoods for probabilistic kernel models like GPs.

Acknowledgements

KC and MF are grateful to Pietro Michiardi and Daniele Venzano for assisting the completion of this work by providing additional computational resources for running the experiments. JPC acknowledges support from the Sloan Foundation, The Simons Foundation (SCGB#325171 and SCGB#325233), and The Grossman Center at Columbia University.

References

Anitescu, M., Chen, J., and Wang, L. A Matrix-free Approach for Solving the Parametric Gaussian Process Maximum Likelihood Problem. *SIAM J. Scientific Computing*, 34(1), 2012.

Asuncion, A. and Newman, D. J. UCI machine learning repository, 2007.

Candela, J. Q. and Rasmussen, C. E. A Unifying View of Sparse Approximate Gaussian Process Regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

Chalupka, K., Williams, C. K. I., and Murray, I. A framework for evaluating approximation methods for Gaussian process regression. *Journal of Machine Learning Research*, 14, 2013.

Chen, J., Anitescu, M., and Saad, Y. Computing f(A)b via Least Squares Polynomial Approximations. *SIAM Journal on Scientific Computing*, 33(1):195–222, 2011.

Davies, A. *Effective Implementation of Gaussian Process Regression for Machine Learning*. PhD thesis, University of Cambridge, 2014.

Duchi, J., Hazan, E., and Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, July 2011.

Filippone, M. and Engler, R. Enabling scalable stochastic gradient-based inference for Gaussian processes by employing the Unbiased LInear System SolvEr (ULISSE). In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, July 6-11, 2015*, 2015.

Filippone, M., Zhong, M., and Girolami, M. A comparative evaluation of stochastic-based inference methods for Gaussian process models. *Machine Learning*, 93(1):93–114, 2013.

Gibbs, M. N. *Bayesian Gaussian processes for regression and classification*. PhD thesis, University of Cambridge, 1997.

Gilboa, E., Saatci, Y., and Cunningham, J. P. Scaling Multidimensional Inference for Structured Gaussian Processes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(2): 424–436, 2015.

Golub, G. H. and Van Loan, C. F. *Matrix computations*. The Johns Hopkins University Press, 3rd edition, October 1996.

Halko, N., Martinsson, P. G., and Tropp, J. A. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Rev.*, 53(2):217–288, May 2011.

Kuss, M. and Rasmussen, C. E. Assessing Approximate Inference for Binary Gaussian Process Classification. *Journal of Machine Learning Research*, 6:1679–1704, 2005.

Lázaro-Gredilla, M., Quinonero-Candela, J., Rasmussen, C. E., and Figueiras-Vidal, A. R. Sparse Spectrum Gaussian Process Regression. *Journal of Machine Learning Research*, 11:1865–1881, 2010.

Murray, I., Adams, R. P., and MacKay, D. J. C. Elliptical slice sampling. *Journal of Machine Learning Research - Proceedings Track*, 9:541–548, 2010.

Nickisch, H. and Rasmussen, C. E. Approximations for Binary Gaussian Process Classification. *Journal of Machine Learning Research*, 9:2035–2078, October 2008.

Notay, Y. Flexible Conjugate Gradients. *SIAM Journal on Scientific Computing*, 22(4):1444–1460, 2000.

Rahimi, A. and Recht, B. Random Features for Large-Scale Kernel Machines. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T. (eds.), *Advances in Neural Information Processing Systems 20*, pp. 1177–1184. Curran Associates, Inc., 2008.

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

Robbins, H. and Monro, S. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22:400–407, 1951.

Schölkopf, B. and Smola, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

Snelson, E. and Ghahramani, Z. Sparse Gaussian Processes using Pseudo-inputs. In *NIPS*, 2005.

Snelson, E. and Ghahramani, Z. Local and global sparse Gaussian process approximations. In Meila, M. and Shen, X. (eds.), *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, AISTATS 2007, San Juan, Puerto Rico, March 21-24, 2007*, volume 2 of *JMLR Proceedings*, pp. 524–531. JMLR.org, 2007.

Srinivasan, B. V., Hu, Q., Gumerov, N. A., Murtugudde, R., and Duraiswami, R. Preconditioned Krylov solvers for kernel regression, August 2014. arXiv:1408.1237.

Titsias, M. K. Variational Learning of Inducing Variables in Sparse Gaussian Processes. In Dyk, D. A. and Welling, M. (eds.), *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, volume 5 of *JMLR Proceedings*, pp. 567–574. JMLR.org, 2009.

Vanhatalo, J., Riihimäki, J., Hartikainen, J., Jylänki, P., Tolvanen, V., and Vehtari, A. Gpstuff: Bayesian modeling with gaussian processes. *The Journal of Machine Learning Research*, 14(1):1175–1179, 2013.

Williams, C. K. I. and Seeger, M. Using the Nyström method to speed up kernel machines. In Leen, T. K., Dietterich, T. G., Tresp, V., Leen, T. K., Dietterich, T. G., and Tresp, V. (eds.), *NIPS*, pp. 682–688. MIT Press, 2000.

Wilson, A. and Nickisch, H. Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP). In Blei, D. and Bach, F. (eds.), *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1775–1784. JMLR Workshop and Conference Proceedings, 2015.

A. Other results not included in the paper

In fig. 3 we report some of the runs that we did not include in the main text for lack of space. The figure reports plots on the error vs. time for the same regression cases considered in the main text but with an isotropic kernel, and results on the concrete dataset with isotropic and ARD kernels.

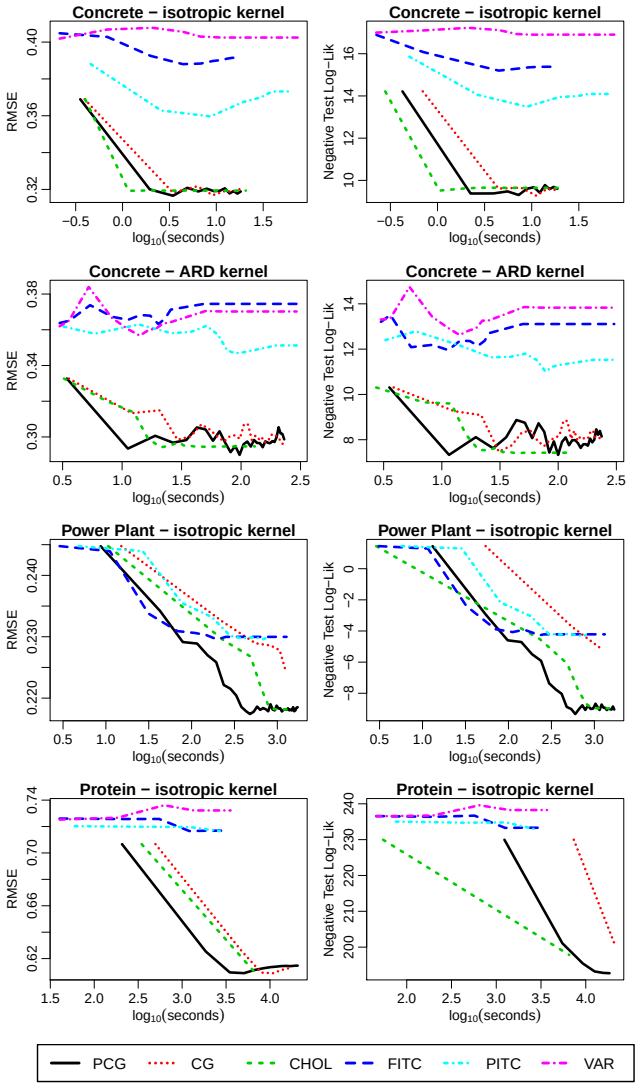

Figure 3. RMSE and log of the likelihood on held out test data over time.

B. Gaussian Processes with non-Gaussian likelihood functions

In this section we report the derivations of the quantities needed to compute an unbiased estimate of the log-marginal likelihood given by the Laplace approximation for GP models with non-Gaussian likelihood functions.

Throughout this section, we assume a factorizing likelihood

$$p(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^{n} p(y_i|f_i).$$

and we specialize the equations to the probit likelihood

$$p(y_i \mid f_i) = \Phi(y_i f_i). \tag{3}$$

where Φ denotes the cumulative function of the Gaussian density. The latent variables $\mathbf{f}$ are given a zero mean GP prior $\mathbf{f} \sim \mathcal{N}(\mathbf{f}|\mathbf{0}, K)$.

For a given value of the hyper-parameters $\boldsymbol{\theta}$, define

$$\Psi(\mathbf{f}) = \log[p(\mathbf{y} \mid \mathbf{f})] + \log[p(\mathbf{f} \mid \boldsymbol{\theta})] + \text{const}. \tag{4}$$

as the logarithm of the posterior density over $\mathbf{f}$. Performing a Laplace approximation amounts in defining a Gaussian $q(\mathbf{f} \mid \mathbf{y}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{f} \mid \hat{\mathbf{f}}, \hat{\Sigma})$, such that

$$\hat{\mathbf{f}} = \arg\max_{\mathbf{f}} \Psi(\mathbf{f}) \qquad \text{and} \qquad \hat{\Sigma}^{-1} = -\nabla_{\mathbf{f}}\nabla_{\mathbf{f}}\Psi(\hat{\mathbf{f}}). \tag{5}$$

As it is not possible to directly solve the maximization problem in equation 5, an iterative procedure based on the following Newton-Raphson formula is usually employed,

$$\mathbf{f}^{\text{new}} = \mathbf{f} - (\nabla_{\mathbf{f}}\nabla_{\mathbf{f}}\Psi(\mathbf{f}))^{-1}\nabla_{\mathbf{f}}\Psi(\mathbf{f}), \tag{6}$$

starting from some initial $\mathbf{f}$ until convergence. The gradient and the Hessian of the log of the target density are

$$\nabla_{\mathbf{f}}\Psi(\mathbf{f}) = \nabla_{\mathbf{f}} \log[p(\mathbf{y} \mid \mathbf{f})] - K^{-1}\mathbf{f} \quad \text{and} \tag{7}$$

$$\nabla_{\mathbf{f}}\nabla_{\mathbf{f}}\Psi(\mathbf{f}) = \nabla_{\mathbf{f}}\nabla_{\mathbf{f}} \log[p(\mathbf{y} \mid \mathbf{f})] - K^{-1} = -W - K^{-1}, \tag{8}$$

where we have defined $W = -\nabla_{\mathbf{f}}\nabla_{\mathbf{f}} \log[p(\mathbf{y} \mid \mathbf{f})]$, which is diagonal because the likelihood factorizes over observations. Note that if $\log[p(\mathbf{y} \mid \mathbf{f})]$ is concave, such as in probit classification, $\Psi(\mathbf{f})$ has a unique maximum.

Standard manipulations lead to

$$\mathbf{f}^{\text{new}} = (K^{-1} + W)^{-1}(W\mathbf{f} + \nabla_{\mathbf{f}} \log[p(\mathbf{y} \mid \mathbf{f})]).$$

We can rewrite the inverse of the negative Hessian using the matrix inversion lemma:

$$\left(K^{-1} + W\right)^{-1} = K - KW^{\frac{1}{2}}B^{-1}W^{\frac{1}{2}}K$$

where

$$B = I + W^{\frac{1}{2}}KW^{\frac{1}{2}}.$$

This means that each iteration becomes:

$$\mathbf{f}^{\text{new}} = (K - KW^{\frac{1}{2}}B^{-1}W^{\frac{1}{2}}K)(W\mathbf{f} + \nabla_{\mathbf{f}} \log[p(\mathbf{y} \mid \mathbf{f})]).$$

We can define $\mathbf{b} = (W\mathbf{f} + \nabla_{\mathbf{f}} \log[p(\mathbf{y} \mid \mathbf{f})])$ and rewrite this expression as:

$$\mathbf{f}^{\text{new}} = K(\mathbf{b} - W^{\frac{1}{2}}B^{-1}W^{\frac{1}{2}}K\mathbf{b}).$$

Algorithm 2 Laplace approximation for GPs

Input: data X, labels $\mathbf{y}$, likelihood function $p(\mathbf{y} \mid \mathbf{f})$
$\mathbf{f} = \mathbf{0}$
repeat
  Compute $\mathrm{diag}(W)$, $\mathbf{b}$, $W^{\frac{1}{2}}K\mathbf{b}$
  $\mathrm{solve}(B, W^{\frac{1}{2}}K\mathbf{b})$
  Compute $\mathbf{a}$, $K\mathbf{a}$
  Compute $\mathbf{f}^{\mathrm{new}}$
until convergence
return $\hat{\mathbf{f}}$, $\mathbf{a}$

Algorithm 3 Stochastic gradients for GPs

Input: data X, labels $\mathbf{y}$, $\hat{\mathbf{f}}$, $\mathbf{a}$
$\mathrm{solve}(B, \mathbf{r}^{(i)})$ for $i = 1, \ldots, N_{\mathbf{r}}$
Compute first term of $\tilde{g}_i$
Compute second term of $\tilde{g}_i$
$\mathrm{solve}(B, W^{\frac{1}{2}}K\mathbf{r}^{(i)})$ for $i = 1, \ldots, N_{\mathbf{r}}$
Compute $\tilde{\mathbf{u}}$
$\mathrm{solve}(B, W^{\frac{1}{2}}\frac{\partial K}{\partial \theta_i}\nabla_{\hat{\mathbf{f}}} \log[p(\mathbf{y} \mid \hat{\mathbf{f}})])$
Compute third term of $\tilde{g}_i$
return $\tilde{\mathbf{g}}$

From this, we see that at convergence

$$\mathbf{a} = K^{-1}\hat{\mathbf{f}} = (\mathbf{b} - W^{\frac{1}{2}}B^{-1}W^{\frac{1}{2}}K\mathbf{b}).$$

As we will se later, the definition of $\mathbf{a}$ is useful for the calculation of the gradient and for predictions.

Proceeding with the calculations from right to left we see that in order to complete a Newton-Raphson iteration the expensive operations are: (i) carry out one matrix-vector multiplication $K\mathbf{b}$, (ii) solve a linear system involving the B matrix, and (iii) carry out one matrix-vector multiplication involving K and the vector in the parenthesis. Calculating $\mathbf{b}$ and performing any multiplications of $W^{\frac{1}{2}}$ with vectors cost $\mathcal{O}(n)$.

All these operations can be carried out without the need to store K or any other $n \times n$ matrices. The linear system in (ii) can be solved using the CG algorithm that involves repeatedly multiplying B (and therefore K) with vectors.

B.1. Stochastic gradients

The Laplace approximation yields an approximate log-marginal likelihood in the following form:

$$\log[\hat{p}(\mathbf{y} \mid \boldsymbol{\theta}, X)] = -\frac{1}{2}\log|B| - \frac{1}{2}\hat{\mathbf{f}}^\top K^{-1}\hat{\mathbf{f}} + \log[p(\mathbf{y} \mid \hat{\mathbf{f}})] \tag{9}$$

Handy relationships that we will be using in the remainder of this section are:

$$\log|B| = \log|I + W^{\frac{1}{2}}KW^{\frac{1}{2}}| = \log|I + KW|;$$

$$(I + KW)^{-1} = W^{-\frac{1}{2}}B^{-1}W^{\frac{1}{2}}.$$

The gradient of the log-marginal likelihood with respect to the kernel parameters $\boldsymbol{\theta}$ requires differentiating the terms that explicitly depend on $\boldsymbol{\theta}$ and those that implicitly depend on it because a change in the parameters reflects in a change in $\hat{\mathbf{f}}$. Denoting by g_i the ith component of the gradient of $\frac{\partial \log[\hat{p}(\mathbf{y}|\boldsymbol{\theta})]}{\partial \theta_i}$, we obtain

$$\begin{aligned} g_i &= -\frac{1}{2}\mathrm{Tr}\left(B^{-1}\frac{\partial B}{\partial \theta_i}\right) \\ &\quad + \frac{1}{2}\hat{\mathbf{f}}^\top K^{-1}\frac{\partial K}{\partial \theta_i}K^{-1}\hat{\mathbf{f}} \\ &\quad + \left[\nabla_{\hat{\mathbf{f}}} \log[\hat{p}(\mathbf{y}|\boldsymbol{\theta})]\right]^\top \frac{\partial \hat{\mathbf{f}}}{\partial \theta_i} \end{aligned} \tag{10}$$

The trace term cannot be computed exactly for large n so we propose a stochastic estimate:

$$-\frac{1}{2}\left[\mathrm{Tr}\left(\widetilde{B^{-1}\frac{\partial B}{\partial \theta_i}}\right)\right] = -\frac{1}{2N_{\mathbf{r}}}\sum_{i=1}^{N_{\mathbf{r}}}(\mathbf{r}^{(i)})^\top B^{-1}\frac{\partial B}{\partial \theta_i}\mathbf{r}^{(i)}.$$

By noticing that the derivative of B is $W^{\frac{1}{2}}\frac{\partial K}{\partial \theta_i}W^{\frac{1}{2}}$, this simplifies to

$$-\frac{1}{2N_{\mathbf{r}}}\sum_{i=1}^{N_{\mathbf{r}}}(\mathbf{r}^{(i)})^\top B^{-1}W^{\frac{1}{2}}\frac{\partial K}{\partial \theta_i}W^{\frac{1}{2}}\mathbf{r}^{(i)},$$

so we need to solve $N_{\mathbf{r}}$ linear systems involving B.

The second term contains the linear system $K^{-1}\hat{\mathbf{f}}$ that we already have from the Laplace approximation and is $\mathbf{a}$.

The third term is slightly more involved and will be dealt with in the next sub-section.

B.1.1. Implicit derivatives

The last (implicit) term in the last equation can be simplified by noticing that:

$$\log[\hat{p}(\mathbf{y} \mid \boldsymbol{\theta})] = \Psi(\hat{\mathbf{f}}) - \frac{1}{2}\log|B|$$

and that the derivative of the first term wrt $\hat{\mathbf{f}}$ is zero because $\hat{\mathbf{f}}$ maximizes $\Psi(\hat{\mathbf{f}})$. Therefore:

$$\left[\nabla_{\hat{\mathbf{f}}} \log[\hat{p}(\mathbf{y} \mid \boldsymbol{\theta})]\right]^\top \frac{\partial \hat{\mathbf{f}}}{\partial \theta_i} = -\frac{1}{2}\left[\nabla_{\hat{\mathbf{f}}} \log|B|\right]^\top \frac{\partial \hat{\mathbf{f}}}{\partial \theta_i}$$

The components of $\left[\nabla_{\hat{\mathbf{f}}} \log |B|\right]$ can be obtained by considering the identity $\log |B| = \log |I + KW|$, so differentiating $\log |B|$ wrt the components of $\hat{\mathbf{f}}$ becomes:

$$\frac{\partial \log |I + KW|}{\partial (\hat{\mathbf{f}})_j} = \mathrm{Tr}\left((I + KW)^{-1} K \frac{\partial W}{\partial (\hat{\mathbf{f}})_j} \right)$$

We can rewrite this by gathering K inside the inverse and, due to the inversion of the matrix product, K cancels out:

$$\frac{\partial \log |I + KW|}{\partial (\hat{\mathbf{f}})_j} = \mathrm{Tr}\left((K^{-1} + W)^{-1} \frac{\partial W}{\partial (\hat{\mathbf{f}})_j} \right)$$

We notice here that the resulting trace contains the inverse of the same matrix needed in the iterations of the Laplace approximation and that the matrix $\frac{\partial W}{\partial (\hat{\mathbf{f}})_j}$ is zero everywhere except in the jth diagonal element where it attains the value:

$$\frac{\partial W}{\partial (\hat{\mathbf{f}})_j} = \frac{\partial^3 \log[p(\mathbf{y} \mid \hat{\mathbf{f}})]}{\partial (\hat{\mathbf{f}})_j^3}$$

For this reason, it would be possible to simplify the trace term as the product between the jth diagonal element of $(K^{-1} + W)^{-1}$ and $\frac{\partial^3 \log[p(\mathbf{y}|\hat{\mathbf{f}})]}{\partial (\hat{\mathbf{f}})_j^3}$. Bearing in mind that we need n of these quantities, we could define

$$D = \mathrm{diag}\left[\mathrm{diag}\left[(K^{-1} + W)^{-1}\right]\right]$$

$$(\mathbf{d})_j = \frac{\partial^3 \log[p(\mathbf{y} \mid \hat{\mathbf{f}})]}{\partial (\hat{\mathbf{f}})_j^3}$$

and rewrite

$$-\frac{1}{2}\left[\nabla_{\hat{\mathbf{f}}} \log |B|\right] = -\frac{1}{2} D \mathbf{d}$$

which is the standard way to proceed when computing the gradient of the approximate log-marginal likelihood using the Laplace approximation (Rasmussen & Williams, 2006). However, this would be difficult to compute exactly for large n, as this would require inverting $K^{-1} + W$ first and then compute its diagonal. Using the matrix inversion lemma would not simplify things as there would still be an inverse of B to compute explicitly. We therefore aim for a stochastic estimate of this term starting from:

$$\begin{aligned} \frac{\partial \log |I + KW|}{\partial (\hat{\mathbf{f}})_j} &= \mathrm{Tr}\left((K^{-1} + W)^{-1} \frac{\partial W}{\partial (\hat{\mathbf{f}})_j} \right) \\ &= \mathrm{Tr}\left((K^{-1} + W)^{-1} \frac{\partial W}{\partial (\hat{\mathbf{f}})_j} \mathrm{E}[\mathbf{r}\mathbf{r}^\top] \right) \end{aligned} \tag{11}$$

where we have introduced the $\mathbf{r}$ vectors with the property $\mathrm{E}[\mathbf{r}\mathbf{r}^\top] = I$. So an unbiased estimate of the trace for each component of $\hat{\mathbf{f}}$ is:

$$\begin{aligned} (\tilde{\mathbf{u}})_j &= \left[\widetilde{\frac{\partial \log |I + KW|}{\partial (\hat{\mathbf{f}})_j}} \right] \\ &= \frac{1}{N_{\mathbf{r}}} \sum_{i=1}^{N_{\mathbf{r}}} (\mathbf{r}^{(i)})^\top (K^{-1} + W)^{-1} \frac{\partial W}{\partial (\hat{\mathbf{f}})_j} \mathbf{r}^{(i)} \end{aligned} \tag{12}$$

which requires solving $N_{\mathbf{r}}$ linear systems involving the B matrix:

$$(K^{-1} + W)^{-1} \mathbf{r}^{(i)} = K(\mathbf{r}^{(i)} - W^{\frac{1}{2}} B^{-1} W^{\frac{1}{2}} K \mathbf{r}^{(i)})$$

The derivative of $\hat{\mathbf{f}}$ wrt θ_i can be obtained by differentiating the expression $\hat{\mathbf{f}} = K \nabla_{\hat{\mathbf{f}}} \log[p(\mathbf{y} \mid \hat{\mathbf{f}})]$:

$$\frac{\partial \hat{\mathbf{f}}}{\partial \theta_i} = \frac{\partial K}{\partial \theta_i} \nabla_{\hat{\mathbf{f}}} \log[p(\mathbf{y} \mid \hat{\mathbf{f}})] + K \nabla_{\hat{\mathbf{f}}} \nabla_{\hat{\mathbf{f}}} \log[p(\mathbf{y} \mid \hat{\mathbf{f}})] \frac{\partial \hat{\mathbf{f}}}{\partial \theta_i}$$

Given that $\nabla_{\hat{\mathbf{f}}} \nabla_{\hat{\mathbf{f}}} \log[p(\mathbf{y} \mid \hat{\mathbf{f}})] = -W$ we can rewrite:

$$(I + KW) \frac{\partial \hat{\mathbf{f}}}{\partial \theta_i} = \frac{\partial K}{\partial \theta_i} \nabla_{\hat{\mathbf{f}}} \log[p(\mathbf{y} \mid \hat{\mathbf{f}})]$$

which yields:

$$\frac{\partial \hat{\mathbf{f}}}{\partial \theta_i} = (I + KW)^{-1} \frac{\partial K}{\partial \theta_i} \nabla_{\hat{\mathbf{f}}} \log[p(\mathbf{y} \mid \hat{\mathbf{f}})]$$

So an unbiased estimate of the implicit term in the gradient of the approximate log-marginal likelihood becomes:

$$-\frac{1}{2} \tilde{\mathbf{u}}^\top (I + KW)^{-1} \frac{\partial K}{\partial \theta_i} \nabla_{\hat{\mathbf{f}}} \log[p(\mathbf{y} \mid \hat{\mathbf{f}})]$$

Rewriting the inverse in terms of B yields:

$$-\frac{1}{2} \tilde{\mathbf{u}}^\top W^{-\frac{1}{2}} B^{-1} W^{\frac{1}{2}} \frac{\partial K}{\partial \theta_i} \nabla_{\hat{\mathbf{f}}} \log[p(\mathbf{y} \mid \hat{\mathbf{f}})]$$

Putting everything together, the components of the stochastic gradient are:

$$\begin{aligned} \tilde{g}_i &= -\frac{1}{2N_{\mathbf{r}}} \sum_{i=1}^{N_{\mathbf{r}}} (\mathbf{r}^{(i)})^\top B^{-1} W^{\frac{1}{2}} \frac{\partial K}{\partial \theta_i} W^{\frac{1}{2}} \mathbf{r}^{(i)} \\ &\quad + \frac{1}{2} \mathbf{a}^\top \frac{\partial K}{\partial \theta_i} \mathbf{a} \\ &\quad - \frac{1}{2} \tilde{\mathbf{u}}^\top W^{-\frac{1}{2}} B^{-1} W^{\frac{1}{2}} \frac{\partial K}{\partial \theta_i} \nabla_{\hat{\mathbf{f}}} \log[p(\mathbf{y}|\hat{\mathbf{f}})] \end{aligned} \tag{13}$$

Algorithm 4 Prediction for GPs with Laplace approximation without Cholesky decompositions

Input: data X, labels $\mathbf{y}$, test input $\mathbf{x}_*$, $\hat{\mathbf{f}}$, $\mathbf{a}$
Compute μ_*
$\text{solve}(B, W^{\frac{1}{2}}\mathbf{k}_*)$
Compute s_*^2, $\Phi\left(\frac{m_*}{\sqrt{1+s_*^2}}\right)$
return $\Phi\left(\frac{m_*}{\sqrt{1+s_*^2}}\right)$

B.2. Predictions

To obtain an approximate predictive distribution, conditioned on a value of the hyper-parameters $\boldsymbol{\theta}$, we can compute:

$$p(y_* \mid \mathbf{y}, \boldsymbol{\theta}) = \int p(y_* \mid f_*) p(f_* \mid \mathbf{f}, \boldsymbol{\theta}) q(\mathbf{f} \mid \mathbf{y}, \boldsymbol{\theta}) df_* d\mathbf{f}. \tag{14}$$

Given the properties of multivariate normal variables, f_* is distributed as $\mathcal{N}(f_* \mid \mu_*, \beta_*^2)$ with $\mu_* = \mathbf{k}_*^\top K^{-1}\mathbf{f}$ and $\beta_*^2 = k_{**} - \mathbf{k}_*^\top K^{-1}\mathbf{k}_*$. Approximating $p(\mathbf{f} \mid \mathbf{y}, \boldsymbol{\theta})$ with a Gaussian $q(\mathbf{f} \mid \mathbf{y}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{f} \mid \boldsymbol{\mu}_q, \Sigma_q)$ makes it possible to analytically perform integration with respect to $\mathbf{f}$ in eq. 14. In particular, the integration with respect to $\mathbf{f}$ yields $\mathcal{N}(f_* \mid m_*, s_*^2)$ with

$$m_* = \mathbf{k}_*^\top K^{-1}\hat{\mathbf{f}}$$

and

$$s_*^2 = k_{**} - \mathbf{k}_*^\top (K + W^{-1})^{-1}\mathbf{k}_*$$

These quantities can be rewritten as:

$$m_* = \mathbf{k}_*^\top \mathbf{a}$$

and

$$s_*^2 = k_{**} - \mathbf{k}_*^\top W^{\frac{1}{2}} B^{-1} W^{\frac{1}{2}} \mathbf{k}_*$$

This shows that the mean is cheap to compute, whereas the variance requires solving another linear system involving B for each test point.

The univariate integration with respect to f_* follows exactly in the case of a probit likelihood, as it is a convolution of a Gaussian and a cumulative Gaussian

$$\int p(y_* \mid f_*) \mathcal{N}(f_* \mid m_*, s_*^2) df_* = \Phi\left(\frac{m_*}{\sqrt{1+s_*^2}}\right). \tag{15}$$

B.3. Low rank preconditioning

When a low rank approximation of the matrix K is available, say $\hat{K} = \Phi\Phi^\top$, the inverse of the preconditioner can be rewritten as:

$$(I + W^{\frac{1}{2}}\hat{K}W^{\frac{1}{2}})^{-1} = (I + W^{\frac{1}{2}}\Phi\Phi^\top W^{\frac{1}{2}})^{-1}$$

By using the matrix inversion lemma we obtain:

$$(I + W^{\frac{1}{2}}\Phi\Phi^\top W^{\frac{1}{2}})^{-1} = I - W^{\frac{1}{2}}\Phi(I + \Phi^\top W\Phi)^{-1}\Phi^\top W^{\frac{1}{2}}$$

Similarly to the GP regression case, the application of this preconditioner is in $\mathcal{O}(m^3)$, where m is the rank of Φ.