Preconditioned Krylov solvers for kernel regression

Balaji Vasan Srinivasan[1], Qi Hu[2], Nail A. Gumerov[3], Ramani Duraiswami[3]*
[1]Adobe Research Labs, Bangalore, India; [2]Facebook Inc., San Francisco, CA, USA
[3]Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA
balsrini@adobe.com, [huqi,gumerov,ramani]@umiacs.umd.edu

Abstract

A primary computational problem in kernel regression is solution of a dense linear system with the $N \times N$ kernel matrix. Because a direct solution has an O(N^3) cost, iterative Krylov methods are often used with fast matrix-vector products. For poorly conditioned problems, convergence of the iteration is slow and preconditioning becomes necessary. We investigate preconditioning from the viewpoint of scalability and efficiency. The problems that conventional preconditioners face when applied to kernel methods are demonstrated. A *novel flexible preconditioner* that not only improves convergence but also allows utilization of fast kernel matrix-vector products is introduced. The performance of this preconditioner is first illustrated on synthetic data, and subsequently on a suite of test problems in kernel regression and geostatistical kriging.

1 Introduction

The basic computations in kernel regression methods involve a number of linear algebra operations on matrices of kernel functions ($\hat{\mathbf{K}}$), which take as arguments the training and/or the testing data. A kernel function $k(x_i, x_j)$ generalizes the notion of the similarity between data points. Given $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}, x_i \in R^d$, the kernel matrix entries are given by,

$$\hat{\mathbf{K}} = \begin{pmatrix} k(x_1,x_1) & \ldots & k(x_1,x_N) \\ \vdots & \ddots & \vdots \\ k(x_N,x_1) & \ldots & k(x_N,x_N) \end{pmatrix}. \tag{1}$$

The kernel function k is chosen to reflect prior information, in the absence of which, the Gaussian kernel Φ is widely used, $\Phi(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$. We use this kernel, though the methods discussed apply to other kernels as well, as is illustrated in experiments. The kernel matrix is usually regularized $\mathbf{K} = \hat{\mathbf{K}} + \gamma\mathbf{I}$ with γ chosen appropriately according to the problem.

Kernel regression appears in many variations: e.g. ridge regression [2], Gaussian process regression [12] and geostatistical kriging [7]. The key computation in all these variants is the solution of a linear system with $\mathbf{K}$.

Direct solution for a dense kernel matrix system has a time complexity O(N^3) and a memory complexity O(N^2), which prevents its use with large datasets. Iterative Krylov methods [16] address this partially by reducing the time complexity to O(kN^2), k being the number of iterations [5, 3]. The dominant cost per Krylov iteration is a kernel matrix-vector product (MVP), whose structure has been utilized to reduce the O(N^2) space and time complexity further. The space requirement is reduced to O(N) by casting the MVP as a weighted kernel summation and computing $k(x_i, x_j)$ *on-the-fly* when evaluating the sum. Further, by using efficient kernel MVP [23, 13, 8, 9, 21], the cost of the MVP in each Krylov iteration can be reduced to O($N \log N$) or O(N^2/p), p being the number

*This work is a part of Chesapeake Bay Forecast System and we gratefully acknowledge National Ocean and Atmospheric Administration (NOAA) Award NA06NES4280016 for funding the project. We also acknowledge NSF award 0403313 and NVIDIA support for the Chimera cluster at the CUDA Center of Excellence at UMIACS.

of processors. *In these fast kernel MVP, there is usually a trade-off between accuracy and speed, and usually a MVP of reduced accuracy can be obtained faster.* This is either explicit (e.g. single precision SSE or graphical processors [21]) or algorithmic (IFGT [23, 13], dual-tree [8], Figtree [9]). However the convergence rate suffers as the problem size increases since the matrix condition number usually increases with data. To speedup iterative methods in these cases, apart from using fast MVP, we need to reduce the number of iterations.

The convergence of the Krylov methods is determined by the matrix condition number κ ($\kappa \geq 1$), $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}, 1 \leq \kappa < \infty$,where $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalues of $\mathbf{K}$ respectively. For smaller κ, the convergence is faster. For larger κ, there is a significant decrease in the convergence rate, necessitating a "preconditioner" [16] to improve the conditioning. Preconditioning has been suggested for kernel methods [3, 10], but to our knowledge, there has been *no previous work to design a preconditioner* for such matrices.

To be effective, the preconditioner matrix construction cost should be small, and be able to take advantage of fast MVPs. We propose a novel preconditioner that improves convergence and has the added benefit that it utilizes the fast MVPs available for the kernel matrix.

The paper is organized as follows. We discuss kernel regression and its variants that we seek to use in Sec. 2. We introduce Krylov methods and their convergence properties in Sec. 3 and survey different preconditioning techniques in Sec. 4. The new preconditioner is introduced and its parameters and convergence are studied in Sec. 5. Finally we test its performance on synthetic and standard datasets in Sec. 6.

2 Kernel regression

We are particularly interested in Gaussian process regression and geostatistical kriging.

Gaussian process regression (GPR) [12] is a probabilistic kernel regression approach which uses the prior that the regression function is sampled from a Gaussian process. Given $D = \{x_i, y_i\}_{i=1}^N$, where x_i is the input and y_i is the corresponding output, the function model is assumed to be $y = f(x) + \epsilon$, where ϵ is a Gaussian noise process with zero mean and variance γ. Rasmussen et al. [12] use the Gaussian process prior with a zero mean function and a covariance function defined by a kernel $\hat{\mathbf{K}}(x, x_*)$, which is the covariance between x and x_*, i.e. $f(x) \sim GP(0, \hat{\mathbf{K}}(x, x_*))$. With this prior, the posterior of the output $f(x_*)$ is also Gaussian with mean $m = k(x_*)^T(\hat{\mathbf{K}} + \gamma I)^{-1}y$ and variance $\Sigma = \hat{\mathbf{K}}(x_*, x_*) - k(x_*)^T(\hat{\mathbf{K}} + \gamma I)^{-1}k(x_*)$, where $x*$ is the input at which prediction is required and $k(x_*) = [\hat{\mathbf{K}}(x_1, x_*), \hat{\mathbf{K}}(x_2, x_*) \ldots, \hat{\mathbf{K}}(x_N, x_*)]^T$. Here "inverses" imply solution of the corresponding linear system. Hyper-parameters are estimated via maximum likelihood techniques [12]. Note that the noise variance γ results in the regularization of the kernel matrix.

Kriging [7] is a group of geostatistical techniques to interpolate the value of a random field at an unobserved location from observations of its value at nearby locations. It was first used with mining and has since been applied in several scientific disciplines including atmospheric science, environmental monitoring and soil management. There are several versions of kriging; the commonly used *simple kriging* results in a formulation similar to Gaussian process regression [7]. Given geostatistical values y_is recorded at locations x_is, the interpolation at a new point x_* is given by $y_* = k(x_*)(\hat{\mathbf{K}} + \gamma\mathbf{I})^{-1}y$, where $k(x_*)$ is similar to the posterior mean in GPR.

3 Krylov methods

Krylov methods are formulated as a "cost-minimization" problem over a set of basis vectors (the Krylov basis) created via matrix vector products of the matrix under consideration. A detailed discussion and analysis can be found in [16, 14]; we provide a brief overview here.

For solving $\mathbf{K}x = b$. Krylov methods begin with an initial guess $x^{(0)}$ and minimize the residual $r^{(k)} = b - \mathbf{K}x^{(k)}$ in some norm, by moving the iterates along directions in the Krylov subspace $\mathcal{K}_k = span(r_0, \mathbf{K}r_0, \ldots, \mathbf{K}^{k-1}r_0)$. The directions are augmented over each Krylov iteration, a significant difference from simpler iterative approaches like Gauss-Siedel where the next iterate depends only on the previous one.

At the k^{th} iteration, an orthogonal matrix $V^{(k)} = [v_1, v_2, \ldots, v_k]$ is generated such that columns of $V^{(k)}$ span the Krylov subspace $\mathcal{K}_k$ [16], $\mathbf{K}V^{(k)} = V^{(k+1)}\bar{\mathbf{H}}^{(k)}$, where $\bar{\mathbf{H}}^{(k)}$ is an augmented

Hessenberg matrix,

$$\bar{\mathbf{H}}^{(k)} = \begin{pmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \dots & h_{1,k} \\ h_{2,1} & h_{2,2} & h_{2,3} & \dots & h_{2,k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & h_{k,k-1} & h_{k,k} \\ 0 & \dots & 0 & 0 & h_{k+1,k} \end{pmatrix},$$

where $h_{i,j} = (v_j^T \mathbf{K} v_i)$. The next iterate $x^{(k)}$ is $x^{(k)} = V^{(k)}\hat{y}$, where $\hat{y}$ is obtained by solving the least squares problem, $\min_{\hat{y}} \|\bar{\mathbf{H}}^{(k)}\hat{y} - \beta e_1\|; e_1 = [1, 0, \dots, 0]^T$. This is the *Arnoldi* iteration for system solution [16].

The *conjugate gradient (CG)* method is the most widely used Krylov method with symmetric matrices. For symmetric $\mathbf{K}$, $\bar{\mathbf{H}}^{(k)}$ is tridiagonal making CG particularly efficient. The *generalized minimum residual (GMRES)* is usually used for non-symmetric problems; GMRES minimizes the residuals $r^{(k)}$ in the $2-$norm. CG minimizes the $\mathbf{K}$-norm of the residual and utilizes the conjugacy in the resulting formulation, which results in not requiring to store the Krylov basis vectors. CG, therefore, is more efficient (lower cost per iteration) than GMRES. Kernel matrices are symmetric and satisfy the Mércer conditions $a^T\mathbf{K}a > 0$, for any a; and hence $\mathbf{K}$ is positive definite. Therefore, when preconditioning is not used, CG has been the preferred choice [5]; however, GMRES has also been used [3].

The convergence rate is given by the ratio of the error (e_k) at k^{th} iteration to the initial error (e_0) in some norm. For example, the ratio for CG [16] is, $\frac{\|e_k\|_{\mathbf{K}}}{\|e_0\|_{\mathbf{K}}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k$. A similar expression may be derived for GMRES [16].

Fast matrix-vector products: The key computation in each Krylov step is the MVP $\mathbf{Kq}$ or $\sum_{i=1}^{N} q_i k(x_i, x_j)$ for some vector q. Existing approaches to accelerate the MVP either approximate it [23, 13, 8, 9] and/or parallelize it [21]; and have their pros and cons. In this paper, we present results with GPUML [21], an open-source package that parallelizes kernel summation on graphical processors (GPUs) though we also tried with FIGTREE [9]. GPUML is easily extendable to generic kernels and works well with reasonable input data dimensions (up to 100).

Need for preconditioning: The condition number κ of kernel matrices depends on the data point distribution and the kernel hyper-parameters. For the Gaussian, the hyper-parameters are the bandwidth σ and the regularizer γ. While x_i's are given, the hyper-parameters are generally evaluated using ML. Fig. 1 shows the κ and number of CG iterations to converge for a kernel matrix constructed from data points chosen uniformly at random from inside a unit cube. We observe the following: there is a direct correspondence between κ and number of CG iterations. For larger regularizer (γ) and smaller bandwidths (σ), the convergence is much better. The data point distribution influences the conditioning as well. It is however not possible to hand select these parameters for each problem. It is necessary to "precondition" [16] the system to be yield better convergence irrespective of the underlying matrix.

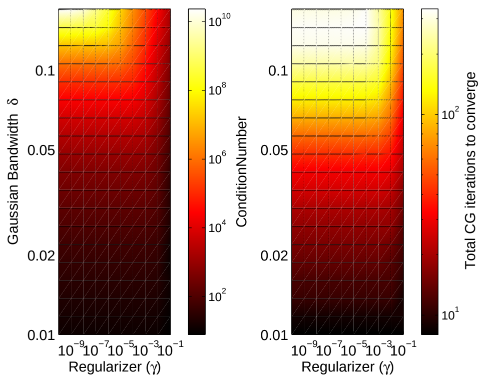


Figure 1: *Effect of kernel hyper-parameters on the matrix conditioning and CG iterations*

4 Preconditioning techniques

A left preconditioner ($\mathbf{M}^{-1}$) operates on $\mathbf{K}x = b$ as $\mathbf{M}^{-1}\mathbf{K}x = \mathbf{M}^{-1}b$; and a right preconditioner operates as, $\mathbf{K}\mathbf{M}^{-1}y = b, \quad y = \mathbf{M}x$. The Preconditioner $\mathbf{M}^{-1}$ should be chosen so that the $\mathbf{M}^{-1}\mathbf{K}$ or $\mathbf{K}\mathbf{M}^{-1}$ have a low κ. An ideal preconditioner ($\mathbf{M}^{-1}$) should well approximate $\mathbf{K}^{-1}$, but be easy to compute.

Conventional preconditioners: Standard preconditioners used in the literature were developed for sparse matrices that arise in the solution of differential equations, and include Jacobi and Symmetric Successive Over-Relaxation (SSOR). For general sparse matrices, incomplete LU or Cholesky preconditioners are often used. The triangular factors $\mathbf{L}$ and $\mathbf{U}$ for a sparse matrix may not be sparse, but *incomplete LU* factorizations leads to sparse $\mathbf{L}$ and $\mathbf{U}$ matrices by setting the coefficients leading to zero entries of the sparse matrix to zero. For a dense matrix, elements are sparsified using a cut-off threshold.

Preconditioners to radial basis function interpolation are a closely related problem. Fast preconditioners have been proposed [1, 4, 6], however, these approaches are limited to low data dimensions ($\sim$ 3 dimensions for X).

Flexible preconditioners: A left preconditioner modifies the right-hand side b in the problem whereas the right preconditioner leaves it as is. This property of right preconditioners can be exploited to create "flexible" preconditioning techniques where a different preconditioner can be used in each Krylov step [15, 18, 11], since the preconditioner only appears implicitly. Flexible preconditioning can be used with both CG [11] and GMRES [15].

Algorithm 1 Flexible GMRES [15]

```
1: $r_0 = (b - \mathbf{K}x_0)$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$
2: Define the $m + 1 \times m$ matrix, $\bar{H}_m = \{h_{i,j}\}_{1\le i\le j+1; 1\le j\le m}$
3: for $j = 0$ to $iter$ do
4:     Solve $\mathbf{M_j} z_j = v_j$ (inner preconditioner)
5:     $w = \mathbf{K}z_j$ (matrix-vector product)
6:     for $i = 0$ to $j$ do
7:         $h_{i,j} = (w, v_i)$, $w = w - h_{i,j}v_i$
8:     end for
9:     $h_{j+1,j} = \|w\|_2$, $v_{j+1} = w/h_{j+1,j}$
10: end for
11: $\mathbf{Z}_{(iter)} = [z_1, \ldots, z_{iter}]$,
12: $y_{iter} = \arg\min_y \|\beta e_1 - \bar{H}^{iter} y\|_2$, $x_{iter} = x_0 + \mathbf{Z}_{iter} y_{iter}$
13: IF satisfied STOP, else $x_0 = x_{iter}$ and GOTO 1
```

Although many papers have shown the convergence of flexible preconditioners under exact arithmetic, it is very hard to estimate convergence rate or the number of outer iterations accurately under inexact arithmetic since the underlying subspaces, $x_0 + span\{\mathbf{M_1}^{-1}v_1, \mathbf{M_2}^{-1}v_2, \ldots, \mathbf{M_k}^{-1}v_k\}$ are no longer a standard Krylov subspace. This affects CG since conjugacy is essential and cannot be guaranteed. Notay [11] proposes 2 modifications to a preconditioned flexible CG. The iterates should be "reorthogonalized" at each step to maintain conjugacy; and the preconditioner system should be solved with high accuracy. Flexible preconditioners are however easily used with GMRES. This fact will be observed in results below, where a poorer performance is observed for flexible CG relative to flexible GMRES.

The algorithmic details of flexible GMRES are shown in Algorithm 1, and the corresponding unpreconditioned version is obtained by replacing the Ms with identity matrices. A similar extension is available for CG as well. The iterations are stopped when $\epsilon = \frac{b - \mathbf{K}x_i}{N}$ drops below a certain tolerance.

Krylov method as a flexible preconditioner: In Algorithm 1, all that is needed to prescribe the right preconditioner is a black-box routine which returns the solution to a linear system with the preconditioner matrix $\mathbf{M}$. Thus, instead of explicitly specifying $\mathbf{M}^{-1}$, it is possible to specify it implicitly by solving a linear system with $\mathbf{M}$ using another Krylov method such as CG. *However, because this iteration does not converge exactly the same way each time it is applied, this is equivalent in exact arithmetic to using a different* $\mathbf{M}$ *for each iteration [18].* We refer to the preconditioner, operating with matrix $\mathbf{M}$ as "inner Krylov" and to the main solver with matrix $\mathbf{KM}^{-1}$ as "outer Krylov".

5 Preconditioning kernel matrices

Conventional preconditioners require construction of the complete preconditioner matrix $\mathbf{M}$ initially, followed by expensive matrix decompositions. Thus they have a computational cost of O(N^3) and a memory requirement of at least $O(N^2)$. Additionally, the preconditioner evaluations will require a O(N^2) "unstructured" matrix-vector product, which does not have any standard acceleration technique and is harder to parallelize. This limits their application to very large datasets and will ruin any advantage gained by the use of fast matrix-vector products (as will be seen later in Sec. 5).

This leads us to propose a key requirement for any preconditioning approach for a kernel matrix: ***the preconditioner should operate with an asymptotic time complexity and memory requirement that are at least the same as the fast matrix vector product.*** One of the main contributions of the paper is a particularly simple construction of a right preconditioner, which also has a fast matrix vector product.

We propose to use a *regularized kernel matrix* $\mathbf{K}$ as a right preconditioner, $\mathbf{M} = \mathbf{K} + \delta\mathbf{I}$. Regularization is a central theme in statistics and machine learning [22], and is not a new concept for kernel machines, e.g. ridge regression, where the kernel matrix ($\hat{\mathbf{K}}$) is regularized as $\hat{\mathbf{K}} + \gamma\mathbf{I}$. However, the γ is chosen by statistical techniques, and hence cannot be controlled.

Our use of this old trick of regularization is in a new context – in the preconditioner matrix $\mathbf{M}$. The simple prescription achieves the following properties: $\mathbf{1}$. improves condition number of matrix $\mathbf{M}$, leading to faster convergence of inner iterations. $\mathbf{2}$. improves conditioning of outer matrix $\mathbf{KM}^{-1}$. To translate this idea in to a useful preconditioner, we need a prescription for selecting the regularization parameter δ and specifying the accuracy ϵ to which the inner system needs to be solved. Because CG is more efficient for unpreconditioned symmetric systems, we use it to solve the inner system.

Preconditioner acceleration: A preconditioner improves convergence at the expense of an increased cost per iteration. This is because there is a cost associated with the preconditioner construction (amortized over all iterations) and cost of the inner iteration. To be useful, the total time taken by the preconditioned approach should be smaller.

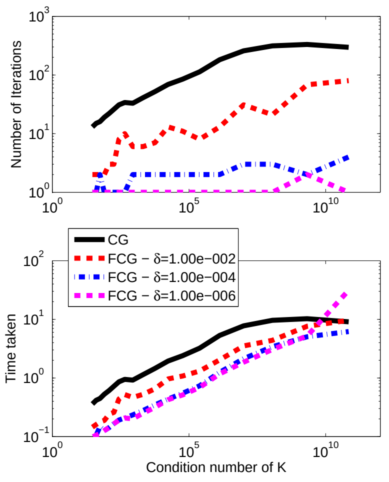
The key advantages of the proposed preconditioner is that, because $\mathbf{M}$ is derived from $\mathbf{K}$, given $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}, x_i \in R^d$ it is not necessary to explicitly construct the preconditioner $\mathbf{M}^{-1}$. Further, the key computation in the inner Krylov iteration is a MVP, $\mathbf{M}x$. This can be accelerated using the same fast algorithm as for $\mathbf{K}$. Further, the preconditioner system only needs to be solved approximately (with a low residual CG tolerance and with a lower accuracy MVP). In our experiments we use low-accuracy fast matrix vector products for the inner iterations (single precision on the GPU). For the outer iterations, the products are performed in double-precision.

Preconditioner parameters: The preconditioner regularizer δ must be chosen on the one hand to converge quickly, while on the other hand not causing it to deviate too much from the inverse of $\mathbf{K}$. The convergence of CG for a kernel matrix for different δ's is shown in Fig. 1. It can be seen that for large enough δ, CG converges rapidly. The CG can also be forced to have an early termination by setting a low solution accuracy (ϵ).

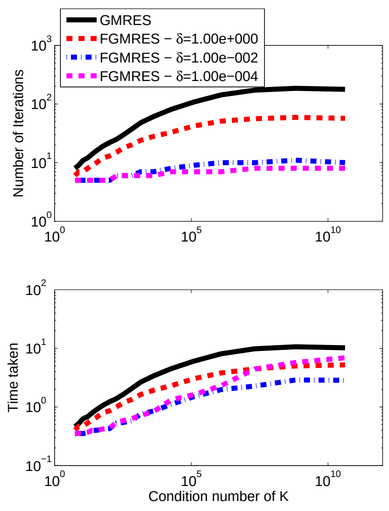
Effect of regularization parameter (δ): In flexible Krylov methods, the outer GMRES iteration solves $\mathbf{KM}^{-1}y = b$, and the inner CG solves $\mathbf{M}x = y$. For small δ, $\mathbf{M}$ is closer to $\mathbf{K}$. Therefore, the outer iteration is better conditioned; however, when $\mathbf{K}$ is ill-conditioned, $\mathbf{M}$ will also be somewhat ill-conditioned, thus slowing the inner iterations.

To demonstrate this, we generated data as before by taking 2000 random samples in a unit cube and generated a matrix for the Gaussian kernel. We tested the convergence with this preconditioner for various regularizer values (Figs. 2(a) and 2(b)). For smaller δ, the convergence of the outer iterations is faster, but the cost per iteration increases due to slow convergence of the inner iterations. Large regularization results in a poor preconditioner $\mathbf{M}$. An intermediate value of the regularizer is therefore optimal. This is observed for both flexible CG (FCG) and flexible GMRES (FGMRES). However, because of its formulation, the optimal FCG regularizer δ is an order of magnitude lower than that for FGMRES.

The choice of a regularizer involves a trade-off between the preconditioner's accurate representation of the kernel matrix and its desired conditioning.



(a) Effect on flexible CG

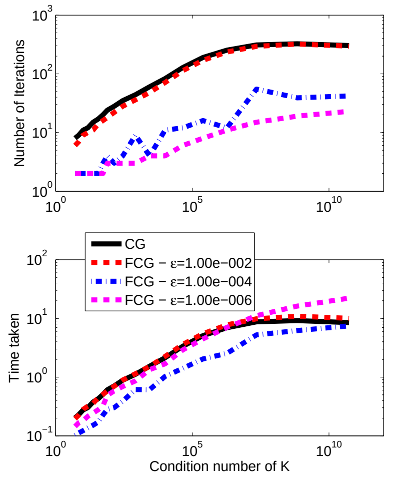


(b) Effect on flexible GMRES

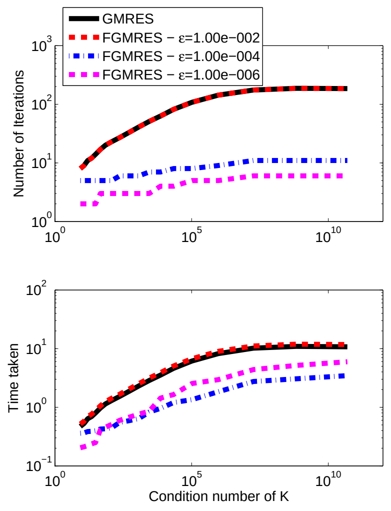
Figure 2: *Effect of regularizer δ on the convergence for FCG and FGMRES for different conditioning of K. The condition number is adjusted by increasing the Gaussian bandwidth σ for* $\mathbf{K}$.

Effect of CG tolerance (ϵ): We tested the performance of the preconditioner for various tolerances in the inner iterations (Figs. 3(a) and 3(b)). There is a consistent improvement in the outer convergence for more precise convergence settings of the inner solver. However, the cost of inner iterations increases. Therefore, an optimal intermediate value of ϵ works best for both FCG and FGMRES.

The choice of tolerance for CG iterations is a trade-off between the required solution accuracy of the preconditioner system (and hence the convergence of the outer iterations) and the related computational cost.



(a) Effect on flexible CG



(b) Effect on flexible GMRES

Figure 3: *Effect of CG tolerance ϵ on the convergence for FCG and FGMRES for different conditioning of K. The condition number is adjusted by increasing the Gaussian bandwidth σ for* $\mathbf{K}$.

Test of convergence We compared the performance of FCG and FGMRES against ILU preconditioned CG and GMRES and the unpreconditioned CG and GMRES. We set the preconditioner δ and tolerance ϵ to $\{10^{-4}, 10^{-4}\}$ respectively for FCG and $\{10^{-2}, 10^{-4}\}$ for FGMRES respectively. 2000 data points were generated randomly in a unit cube for testing the convergence. The computational performance and convergence is shown in Fig. 4. The number of iterations of the preconditioned approaches are always less than those for the unpreconditioned cases. The computational cost per iteration is the least for CG compared to GMRES, FCG, and FGMRES. Incomplete LU (ILU) based preconditioners are marginally better in convergence (iterations) compared to our approach for better conditioned cases. But ILU (and other similar preconditioners) require explicit kernel matrix construction and rely on sparsity of the matrix to be solved and the absence of these properties in kernel matrices result in significantly higher computational cost compared to our preconditioners as well as the unpreconditioned solver. This makes them impractical.

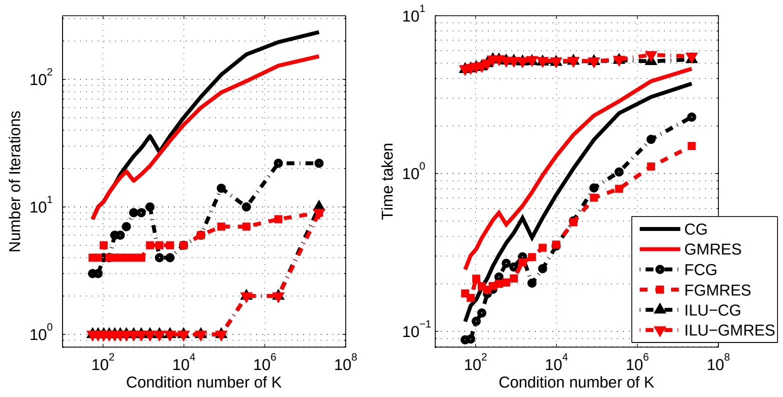


Figure 4: *Performance of our preconditioner with CG and GMRES against ILU-preconditioned and unpreconditioned versions for different conditioning of K. The condition number is adjusted by increasing the Gaussian bandwidth σ for* $\mathbf{K}$.

We see that FCG needs increased accuracy of the inner linear system solution. In contrast, FGMRES is more forgiving of inner linear system error and only requires coarse accuracy to reduce the number of outer iterations to the same magnitude as FCG. On the other hand, especially for the ill-conditioned matrices, solving the inner Krylov method with fine accuracy takes much more time. Hence, given the ill-conditioned kernel matrices, the best FMGRES has the lesser number of outer iterations as well as smaller computation time.

The unpreconditioned algorithm of choice is CG, because of its lower storage and efficiency. However, FGMRES is the method of choice for preconditioned iterations. While GMRES requires extra storage in comparison to CG, FCG also requires this extra storage (for reorthogonalization), and we do not pay a storage penalty for our choice of FGMRES over FCG. In the sequel, we accordingly use FGMRES.

6 Experiments

The preconditioner performance is illustrated on various datasets on different variants of kernel regression. We first look at GPR with a Gaussian kernel and then experiment on kriging [7] and report results on a large geostatistical dataset.

Although dataset-specific tuning of the preconditioner parameters can yield better results, this is impractical. We therefore use the following rules to set the preconditioner parameters. The tolerance (ϵ) for the preconditioner system solution is set at an order of magnitude larger than the outer iteration tolerance (e.g., outer tolerance $= 10^{-4}$, inner tolerance $= 10^{-3}$). Similarly, the preconditioner regularizer δ is also set to an order of magnitude higher than the kernel regularizer γ. When the outer regularizer is 0, the inner regularizer is set to 10^{-3}. While this might not yield the best preconditioner system, it performs well in most cases from our experiments. In all experiments, the outer iteration tolerance was set to 10^{-6}.

Gaussian process regression (GPR): The key computational bottleneck in GPR involves the solution of a linear system of kernel matrix. Direct solution via decompositions such Cholesky [12] requires O(N^2) space and O(N^3) time requirements. Alternatively, Mackay et al. [5] use *CG* to solve the GPR.

In our experiments, the covariance kernel parameters are estimated via maximum likelihood as in [12] with a small subset of the input data. We compare the performance of our preconditioner against a direct solution using [12], our implementation of the CG approach in [5] and Incomplete LU based preconditioner on various standard datasets(`www.liaad.up.pt/~ltorgo/Regression/`).

The kernel matrix vector product in all compared scenarios was also accelerated using GPUML. Table 1 shows the corresponding result.

The convergence of the preconditioned FGMRES (both with ILU and our preconditioner) is consistently better than the unpreconditioned approach. Although for smaller datasets there is very little separating the computational performance of the solvers, the performance of our FGMRES with our preconditioner gets better for larger data sizes. This is because, for larger problems, cost per iteration in both CG and FGMRES increases, and thus a FGMRES which converges faster becomes significantly better than the CG-based approach. Further, for larger problems, both the direct method and ILU-preconditioning run into space issues due to the requirement of the physical construction of the kernel matrix.

Table 1: *Performance of FGMRES based Gaussian process regression against the direct, CG [5] and ILU-preconditioned solvers; d is the dimension and N is the size of the regression dataset with the Gaussian kernel. Total time taken for prediction is shown here, with the number of iterations for convergence indicated within parenthesis. The mean error in prediction between the two approaches was less than 10^{-6} in all the cases.*

Datasets ($d \times N$)	Direct [12]	CG [5]	ILU	FGMRES
Diabetes (3×43)	0.03	$\mathbf{0.03}$ (8)	0.25 (4)	0.04 (3)
Boston Housing (14×506)	0.86	0.67 (33)	0.86 (3)	$\mathbf{0.62}$ (3)
Pumadyn (9×4499)	63.61	5.61 (32)	73.45 (4)	$\mathbf{3.61}$ (3)
Bank (1) (9×4499)	64.18	6.53 (35)	74.73 (4)	$\mathbf{4.28}$ (3)
Robot Arm (9×8192)	232.61	23.81 (75)	268.37 (3)	$\mathbf{11.79}$ (4)
Bank (2) (33×4500)	66.85	49.40 (38)	76.54 (3)	$\mathbf{37.74}$ (3)
Census (1) (9×22784)	$--$	117.45 (42)	$--$	$\mathbf{90.31}$ (4)
Ailerons (41×7154)	170.76	131.34 (31)	208.87 (2)	$\mathbf{128.22}$ (4)
2D Planes (11×40768)	$--$	469.41 (31)	$--$	$\mathbf{415.30}$ (6)
Census (2) (17×22784)	$--$	663.70 (83)	$--$	$\mathbf{482.50}$ (5)
Sarcos (28×44484)	$--$	$--$	$--$	$\mathbf{1090.85}$ (4)
Kriging [Pacific Coast Data]	$--$	$2,301 \pm 800$s	$--$	$\mathbf{725 \pm 190}$s
($3 \times 179,065 \pm 35,405$)	$--$	(46 ± 12)	$--$	(3 ± 1)

Low rank approaches [20, 17, 19] also address the time complexity in kernel regression by working on an "active set" of set M and reducing the time to O(M^2N). We compared with the low rank GPR based on [19], and found our approach to be superior. Because these approaches involve the solution of a large optimization problem, straightforward algorithmic acceleration or parallelization is not possible. Since the methods and accelerations used in this paper are significantly different from those in [19], we have not reported these here.

Kriging: We compared FGMRES-based kriging against the CG version on the ocean chlorophyll concentration data recorded along the Pacific coast of North America obtained from National Oceanic and Atmospheric Administration(`http://coastwatch.pfel.noaa.gov/`). We look at the 7-day aggregate of the chlorophyll concentration, which is recorded on a grid of 416×600. However, this includes several locations with missing data or those located over land. This results in approximately $179,065 \pm 35,405$ data samples per week.

It was observed that the CG-based approach converges in 46 ± 12 iterations in $2,301 \pm 800$s, whereas, the FGMRES converges in just $\mathbf{3 \pm 1}$ (outer) iterations in $\mathbf{725 \pm 190}$s, resulting in over 3X speedup.

7 Conclusions and discussions

A method to improve convergence of Krylov methods used in kernel methods was demonstrated. The key contributions of the paper are as follows,

- A novel yet simple preconditioner is proposed to solve a linear system with a kernel matrix using flexible Krylov methods.
- A technique to accelerate the inner preconditioner system using truncated CG with fast matrix vector products was developed.
- Rules to select the preconditioner parameter were shown.

References

[1] R.K. Beatson, J.B. Cherrie, and C.T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Advances in Computational Mathematics*, 11:253–270, 1999.

[2] C. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[3] N. de Freitas, Y. Wang, M. Mahdaviani, and D. Lang. Fast Krylov methods for n-body learning. In *Advances in Neural Information Processing Systems*, 2005.

[4] A.C. Faul, G. Goodsell, and M.J.D. Powell. A Krylov subspace algorithm for multiquadric interpolation in many dimensions. *IMA Journal of Numerical Analysis*, 25:1–24(24), 2005.

[5] M. Gibbs and D. Mackay. Efficient implementation of Gaussian processes. Technical report, 1997.

[6] N.A. Gumerov and R. Duraiswami. Fast radial basis function interpolation via preconditioned Krylov iteration. *SIAM Journal on Scientific Computing*, 29(5):1876–1899, 2007.

[7] E.H. Isaaks and R.M. Srivastava. *Applied Geostatistics*. Oxford University Press, 1989.

[8] D. Lee, A. Gray, and A. Moore. Dual-tree fast Gauss transforms. In *Advances in Neural Information Processing Systems 18*, pages 747–754. 2006.

[9] V. Morariu, B.V. Srinivasan, V.C. Raykar, R. Duraiswami, and L. Davis. Automatic online tuning for fast Gaussian summation. In *Advances in Neural Information Processing Systems*, 2008.

[10] I. Murray. Gaussian processes and fast matrix-vector multiplies. In *Numerical Mathematics in Machine Learning workshop*, 2009.

[11] Y. Notay. Flexible conjugate gradients. *SIAM J. Sci. Comput.*, 22(4):1444–1460, 2000.

[12] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[13] V.C. Raykar and R. Duraiswami. The improved fast Gauss transform with applications to machine learning. In *Large Scale Kernel Machines*, pages 175–201, 2007.

[14] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press, 1992.

[15] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993.

[16] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

[17] M Seeger, C.K.I Williams, N Lawrence, and S. Dp. Fast forward selection to speed up sparse gaussian process regression. In *Workshop on AI and Statistics 9*, 2003.

[18] V. Simoncini and D.B. Szyld. Flexible inner-outer Krylov subspace methods. *SIAM J. Numer. Anal.*, 40(6):2219–2239, 2002.

[19] E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264, 2006.

[20] E. Snelson and Z. Ghahramani. Local and global sparse gaussian process approximations. In *Artificial Intelligence and Statistics (AISTATS)*, 2007.

[21] B.V. Srinivasan, Q. Hu, and R. Duraiswami. GPUML: Graphical processors for speeding up kernel machines. In *Workshop on High Performance Analytics - Algorithms, Implementations, and Applications*. Siam International Conference on Data Mining, 2010.

[22] V. Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, 2nd edition, November 1999.

[23] C. Yang, C. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast gauss transform. In *Advances in Neural Information Processing Systems*, 2004.