FAST MATRIX-VECTOR PRODUCT BASED FGMRES FOR KERNEL MACHINES

BALAJI VASAN SRINIVASAN,
M.S. SCHOLARLY PAPER
DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF MARYLAND, COLLEGE PARK

Abstract. Kernel based approaches for machine learning have gained huge interest in the past decades because of their robustness. In some algorithms, the primary problem is the solution of a linear system involving the kernel matrix. Iterative Krylov approaches are often used to solve these efficiently [2, 3]. Fast matrix-vector products can be used to accelerate each Krylov iteration to further optimize the performance. In order to reduce the number of iterations of the Krylov approach, a preconditioner becomes necessary in many cases. Several researchers have proposed flexible preconditioning methods where the preconditioner changes with each iteration, and this class of preconditioners are shown to have good performance [6, 12]. In this paper, we use a ***Tikhonov regularized kernel matrix*** as a preconditioner for flexible GMRES [12] to solve kernel matrix based systems of equations. We use a truncated conjugate gradient (CG) method to solve the preconditioner system and further accelerate each CG iteration using fast matrix-vector products. The convergence of the proposed preconditioned GMRES is shown on synthetic data. The performance is further validated on problems in Gaussian process regression and radial basis function interpolation. Improvements are seen in each case.

Key words. flexible GMRES, graphical processing unit, Gaussian process regression, radial basis function interpolation

1. Introduction.

Algorithms based on kernel methods play a central role in statistical machine learning. At their core are a number of linear algebra operations on matrices of kernel functions which take as arguments the training and testing data. A kernel function $\Phi(x_i, x_j)$ generalizes the notion of the similarity between a test and training point. Given a set of data points, $\mathbf{X} = \{x_1, x_2, \dots, x_N\}, x_i \in R^d$, the kernel matrix is given by,

$$\mathbf{K} = \begin{pmatrix} k_{11} & \dots & k_{1N} \\ \vdots & \ddots & \vdots \\ k_{N1} & \dots & k_{NN} \end{pmatrix}, \tag{1.1}$$

where $k_{ij} = \Phi(x_i, x_j)$. Kernel matrices are symmetric and satisfy the Mércer conditions [1], $a^T\mathbf{K}a \geq 0$, for any a; and hence $\mathbf{K}$ is positive semi-definite. Φ is generally chosen to reflect prior information about the problem. In the absence of any prior knowledge, the negative squared exponential (Gaussian) is the most widely used kernel function and is the one that we use in this paper,

$$k_{ij} = s \exp\left(-\frac{\|x_i - x_j\|^2}{h^2}\right). \tag{1.2}$$

However, the methods that we shall discuss are generic enough for other kernels as well.

In many learning algorithms, it might be necessary to solve a linear system with the matrix $\mathbf{K}$. Kernel regression and Gaussian process regression are examples from learning. An example from function-fitting is radial basis function (RBF) interpolation. Iterative Krylov approaches [2, 3, 5] are often used for efficient solution of such systems. The iterative approaches are advantageous over the direct solvers for large systems because they do not require the explicit construction of the underlying kernel matrix, saving both computations and memory. Because the kernel matrices are symmetric positive semidefinite, the *conjugate gradient (CG)* method has been a common choice [3, 10], however other Krylov approaches like GMRES have also been used in some cases [2].

REMARK 1.1. *The dominant cost per iteration in the Krylov approaches is a matrix-vector product. Thus, they are especially useful for matrices for which fast matrix-vector products are available. (eg. sparse matrices, Fourier and related matrices, structured matrices)*

The structure of the kernel matrix has been exploited to accelerate the Krylov iterations, eg. [10] for Gaussian kernel. One obvious way is to avoid the explicit construction of the matrix $\mathbf{K}$, and compute the function $\Phi(x_i, x_j)$ *on-the-fly* resulting in O(N) memory requirement. Further, by using efficient approximations [11, 7, 8] or parallelization (GPUs [16]), the cost of the matrix-vector product in each Krylov iteration can be significantly reduced. A detailed discussion on the accelerated kernel matrix-vector products is given in Section 2.

However, when the underlying system is ill-conditioned, there is a significant degradation of the performance of iterative approaches, necessitating the use of a preconditioner for the Krylov iterations. Either left or right preconditioning may be used, though left preconditioner is more common. Consider a system $\mathbf{K}x = b$; a left preconditioner $\mathbf{M}^{-1}$ operates on this system as,

$$\mathbf{M}^{-1}\mathbf{K}x = \mathbf{M}^{-1}b. \tag{1.3}$$

A right preconditioner, on the other hand, operates as follow,

$$\mathbf{K}\mathbf{M}^{-1}(\mathbf{M}x) = b. \tag{1.4}$$

An ideal preconditioner ($\mathbf{M}^{-1}$) is a matrix that is close to the inverse of the matrix $\mathbf{K}$. Consequently the matrix $\mathbf{M}$ is "close" to the matrix $\mathbf{K}$. Of course $\mathbf{K}^{-1}$ is not known, otherwise there would be no need to solve the system!

Popular preconditioners like Jacobi, SSOR, ILU [13] can be used to improve the convergence, however, these preconditioners have an O(N^2) space and computation requirement for dense matrices, which would ruin any advantage gained by the fast matrix-vector products. The preconditioner must have a representation that allows for a fast matrix-vector product just like the kernel matrix. In this paper, we propose such a novel preconditioner for a flexible GMRES algorithm for solving a kernel system and its performance is illustrated with examples from machine learning. The motivation behind the particular choice of GMRES is detailed in section 3.

This paper is organized as follows. We introduce accelerated kernel matrix vector products based on approximation algorithms and GPU-based parallelizations in section 2. In section 3, we introduce the flexible GMRES algorithm and the proposed preconditioner and further discuss the possible accelerations of the FGMRES in the light of the desired accuracy. We discuss our experiments in section 4 and illustrate the performance of the proposed approach.

2. Kernel matrix-vector products. The key computation in each Krylov iteration is the matrix-vector product,

$$\begin{pmatrix} f_1 \\ \vdots \\ f_N \end{pmatrix} = \begin{pmatrix} k_{11} & \dots & k_{1N} \\ \vdots & \ddots & \vdots \\ k_{N1} & \dots & k_{NN} \end{pmatrix} \begin{pmatrix} q_1 \\ \vdots \\ q_N \end{pmatrix}, \quad \mathbf{f} = \mathbf{K}\mathbf{q}. \tag{2.1}$$

This can also be written as a weighted summation of kernel functions,

$$f(x_j) = \sum_{i=1}^{N} q_i K(x_i, x_j). \tag{2.2}$$

Existing approaches to accelerate these kernel summations either approximate the kernel summation or parallelize them.

2.1. Approximation. The algorithms we consider in this category are also called ϵ-exact approximations. The objective is to evaluate $\hat{f}$ in linear time such that $\|\hat{f} - f\| \leq \epsilon$ in some norm. The key idea here is to utilize the structure of the problem and special data structures to efficiently approximate the sum in o(N^2) time.

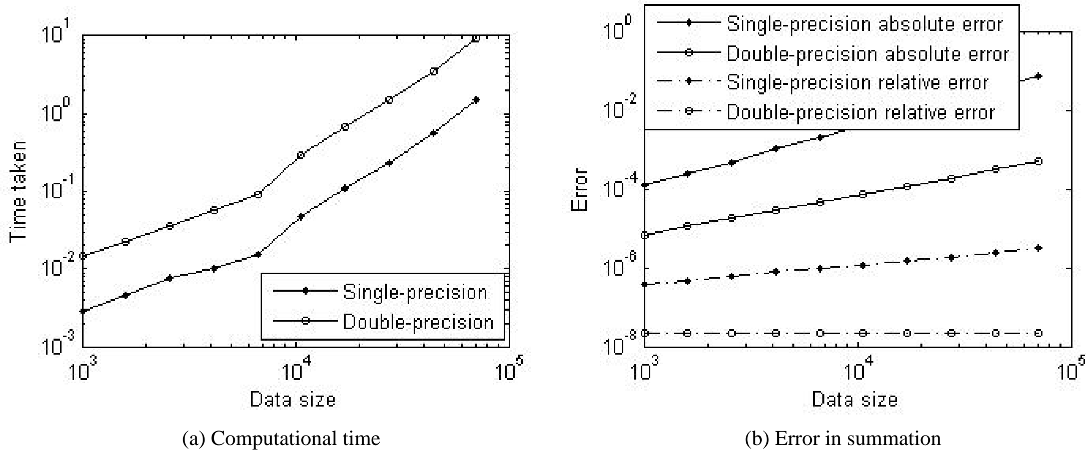
Several approximations exist for the Gaussian kernel summations, notable ones are the fast Gauss transform [4] (does reasonably for data dimension upto 3), improved fast Gauss transforms (IFGT) [11] (uses different data structures and expansions to work well for higher dimensional data) and dual-trees [7]. These approaches evaluate the weighted Gaussian sum using special data structures and approximation techniques. The advantage of the approximation based accelerations is that the computational complexity can be linear (O(N)). However, the performance is data-dependent. The tree-based approach performs well for low data bandwidth (h in Eq. 1.2), whereas the IFGT performs well for larger bandwidths. Both these approaches perform badly for large dimensions ($>$ 10). Morariu et al. combine the advantages of these two approaches and propose an algorithm, **FIGTREE** [8] which automatically selects the fastest approach for a particular data and desired accuracy. Morariu et al. further note that in some cases, a direct summation is faster than any approximation algorithm; therefore FIGTREE chooses between IFGT, a tree-based approximation and the direct summation based on the data (dimension, bandwidth and desired accuracy); we use this in our paper.

Pros: Linear time computation, possible to obtain any desired accuracy

Cons: Data dependent performance, curse of dimensionality

2.2. Parallelization. Computer chip-makers are no longer able to easily improve the clock speed on processors, with the result that computer architectures of the future will have more cores, rather than more capable faster cores. This era of multicore computing requires algorithms which are particularly suited for data parallel architecture. A particularly capable set of data parallel processors are the graphical processing units (GPU), which have evolved into highly capable compute coprocessors. A GPU is a highly parallel, multi-threaded, multi-core processor with tremendous computational power, and it is possible to perform general purpose scientific computations on them, thus utilizing their superior performance. While GPUs can do double precision, most speedup is gained on single precision computations, and *currently double precision is advised only when it is absolutely essential for algorithmic correctness*.

The kernel matrix-vector product have been efficiently parallelized on the GPUs [15]. For our experiments, we used the open source package **GPUML (GPU for Machine Learning)** [15] to compute the kernel based matrix vector product associated with the preconditioner evaluations. In GPUML, the matrix vector product is distributed across several threads in light of the memory accesses. Each thread is assigned to evaluate one element of the output



(a) Computational time

(b) Error in summation

Fig. 2.1: *Performance comparison between GPUML single and double precision computations.*

vector (f_j) and is further optimized for memory accesses by efficient utilization of the shared memory and registers in the GPU. Both single and double precision versions are available in GPUML [15].

A comparison of computational performance and errors (relative and absolute) of GPUML (single and double precision) on a 3-dimensional data generated at random between 0 and 1 is shown in Fig. 2.1. It can be seen that single-precision computations are approximately 6-times faster than the double-precision computations. The errors reported here are obtained from comparison with FIGTREE with $\epsilon = 10^{-12}$. The absolute error in both cases increases with increasing data size because of the round-off errors on the GPU. However, the error in single-precision summation is an order higher than that of the double-precision counterpart. A similar difference is observed in the relative errors as well. However, because of the comparison with a FIGTREE (with $\epsilon = 10^{-12}$), the relative error for double precision GPUML is flat.

Pros: Data independent speedup, performs well upto 100 dimensions

Cons: Quadratic complexity, memory restrictions, curse of dimensionality (for large dimensions)

2.3. GPUML vs FIGTREE. While both GPU based parallelization and CPU approximation are promising, each has its own pros and cons, and it is important to utilize the correct approach for a given problem for the best performance. In order to illustrate the differences in performance, random data points were generated in a unit hypercube and used with FIGTREE and GPUML for single and double precision accuracy. GPU's accuracy cannot be controlled beyond the specification of single/double precision operations. However, FIGTREE offers a wide range of accuracy control by the specification of ϵ. We set ϵ to 10^{-6} for single-precision and 10^{-12} for double-precision computations.

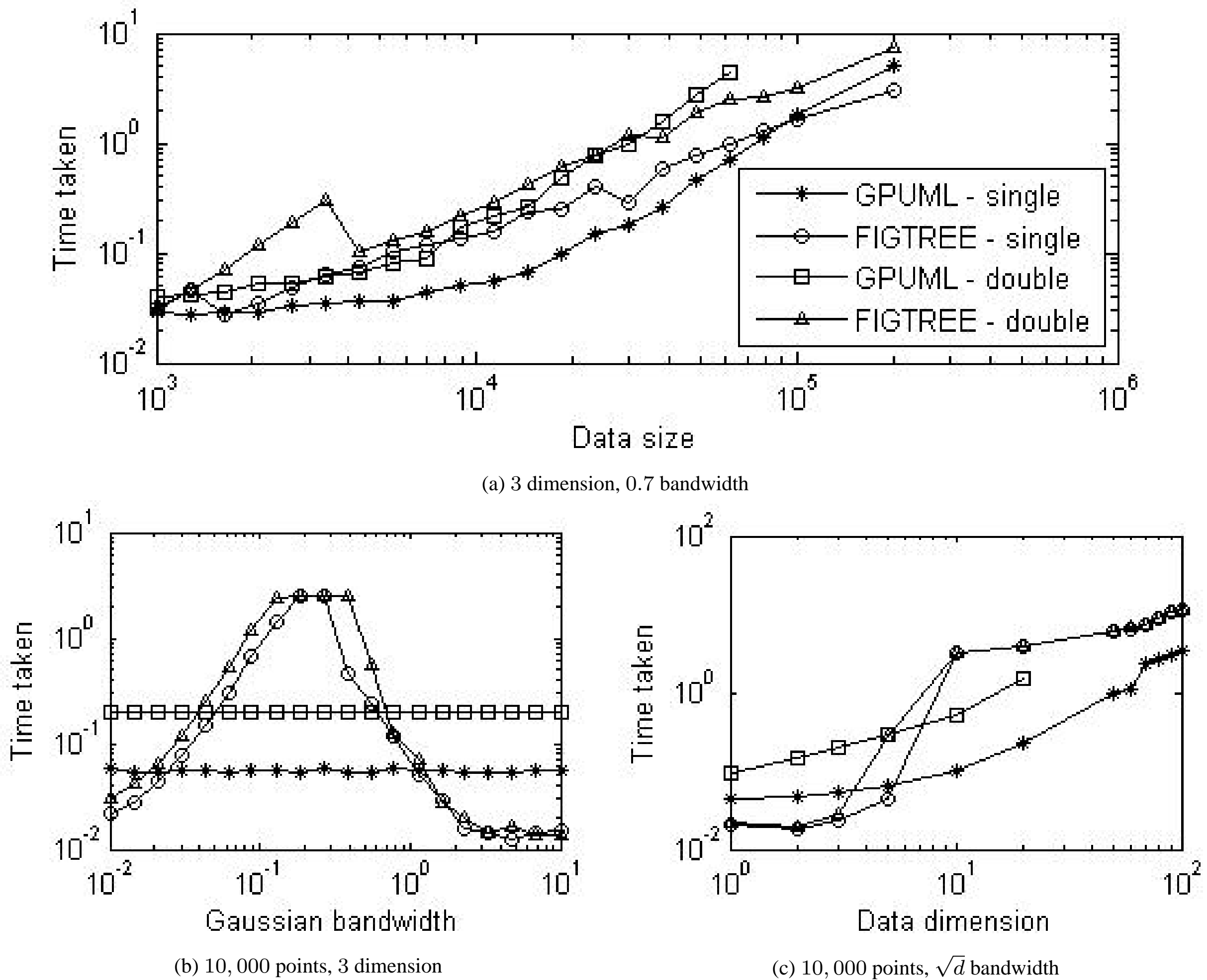
The time taken by the two approaches for various sizes of a 3-dimensional input data x is given in Fig. 2.2a. It can be seen that the single precision GPU acceleration is significant upto data size of $\sim 100,000$, but for large data sizes, the quadratic complexity hinders the performance, therefore FIGTREE (with single precision accuracy) outperforms GPUML. Corresponding double-precision analysis reveal that FIGTREE outperforms GPUML even for a data size of $20,000$. Because of the memory restrictions on the GPU, GPUML can handle only upto $70,000$ 3-dimensional data points in double-precision. It should be noted that FIGTREE chooses a direct summation when the approximations become too computationally expensive.

The bandwidth dependence of FIGTREE is illustrated in Fig. 2.2b, where a set of $10,000$-point 3-dimension data were tested for various bandwidths. For low bandwidths, the tree-based approach is optimal; for large bandwidths, IFGT is optimal. However, both the approaches perform badly for medium bandwidths, where GPUML outperforms FIGTREE.

It was mentioned earlier that the curse of dimensionality is significant in both the approaches. However, a comparison of the performances reveal that the effect of dimensions is more significant in FIGTREE than GPUML (which gives significant speedup upto 100 dimensions), as seen in Fig. 2.2c for $10,000$-data points.

REMARK 2.1. *The choice of fast approximations and GPUML, particularly for large data, is mainly based on three factors: desired accuracy, dimensionality and size of the data. For dimensions* < 10 *and low accuracy requirements, FIGTREE (single-precision) can be used for* $N > 100,000$*, GPUML can be used in all other instances. For high accuracy, although GPUML (double-precision) outperforms FIGTREE (double-precision) for small data sizes, the advantage gained is limited. Therefore, FIGTREE can be chosen for all such cases.*

3. Flexible GMRES. As mentioned earlier, when the underlying system is ill-conditioned, there is a significant degradation in the performance of iterative approaches, necessitating the use of a preconditioner for the Krylov ap-



(a) 3 dimension, 0.7 bandwidth

(b) 10, 000 points, 3 dimension

(c) 10, 000 points, $\sqrt{d}$ bandwidth

Fig. 2.2: *Speedup comparison between GPUML and FIGTREE*

proaches. A left preconditioner (Eq. 1.3) modifies the right-hand side in the problem whereas the right preconditioner (Eq. 1.4) does not. This difference has been exploited in creating the flexible GMRES [12] algorithm where the preconditioner is changed every step. In flexible GMRES, the preconditioner $\mathbf{M}$ is allowed to change over each GMRES iteration. Such a flexible preconditioning has been proved to have good performance.

Flexible GMRES is a modified right preconditioning for GMRES, where the underlying preconditioner can be changed during each iteration of the algorithm, thus resulting in a flexible approach. The flexible GMRES algorithm is shown in Algorithm 1. FGMRES can be accelerated by evaluating the $\mathbf{K}z_j$ using FIGTREE/GPUML. Because the accuracy of the matrix-vector product is important for the performance of the FGMRES algorithm, double-precision computations must be used inspite of the additional cost that will be incurred.

REMARK 3.1. *The FGMRES algorithm does not explicitly need the preconditioner* $\mathbf{M}^{-1}$. *Instead, all that is needed is a way to solve a linear system with the matrix* $\mathbf{M}$.

Simocini et al. [14] use a Krylov approach as a preconditioner, this approach is termed as the *inner-outer Krylov method*. In other words, the system $\mathbf{M_j} z_j = v_j$ (Step 3 in Algorithm 1) is solved using another Krylov method (inner Krylov), which is then used to precondition the outer Krylov iteration. We adopt a similar technique for our preconditioner, where we use a low-accuracy inner Krylov method.

REMARK 3.2. *Because it is difficult to guarantee the symmetry of the system for all the flexible preconditioned iterations, conjugate gradient cannot be easily used for outer iterations.*

3.1. Design of preconditioner. A good left/right preconditioner ($\mathbf{M}$) is an approximate representation of the matrix $\mathbf{K}$, for which the solution $\mathbf{M_j} z_j = v_j$ should be easy to obtain. Conventional preconditioning relies on the sparsity in the matrix and applying these approaches to dense kernel matrices would require O(N^2) time and memory complexity, which would negate the advantage gained by the fast matrix-vector products. Another possible strategy is the construction of approximate inverses by banding the kernel matrix. However, banding the kernel matrix might not work for larger bandwidths. Instead in this paper, we use a preconditioner in which the matrix $\mathbf{M}$ is a Tikhonov

Algorithm 1 Flexible GMRES

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

regularized approximation to the matrix $\mathbf{K}$.

Because a kernel matrix is positive semi-definite, an ill-conditioned kernel matrix would result if several eigenvalues are close to zero. For a Gaussian kernel, this occurs whenever the bandwidth h is very large. For the preconditioner $\mathbf{M}$, we propose to use,

$$\mathbf{M} = \mathbf{K} + \sigma \mathbf{I} \tag{3.1}$$

Such a formulation would result in $\mathbf{M}$ whose eigenvalues are σ more than those of $\mathbf{K}$, but still close to $\mathbf{K}$. For large enough σ, the preconditioner would be significantly better-conditioned compared to $\mathbf{K}$. A standard conjugate gradient (CG) approach can be used to solve the system $\mathbf{M_j} z_j = v_j$ in Algorithm 1. Because a preconditioner is required to be only an approximate representation, the tolerance for the CG can be set at a higher value [6] (early truncation). As the CG residuals are not the same for all the outer iterations, the preconditioner changes over each iteration, thus the approach becomes a flexible variant of a preconditioned GMRES.

REMARK 3.3. *A good preconditioner will improve the convergence of the iterative approach at the expense of an increased cost per iteration. For a preconditioner to be useful, the total time taken by the preconditioned approach should be less than the unpreconditioned approach.*

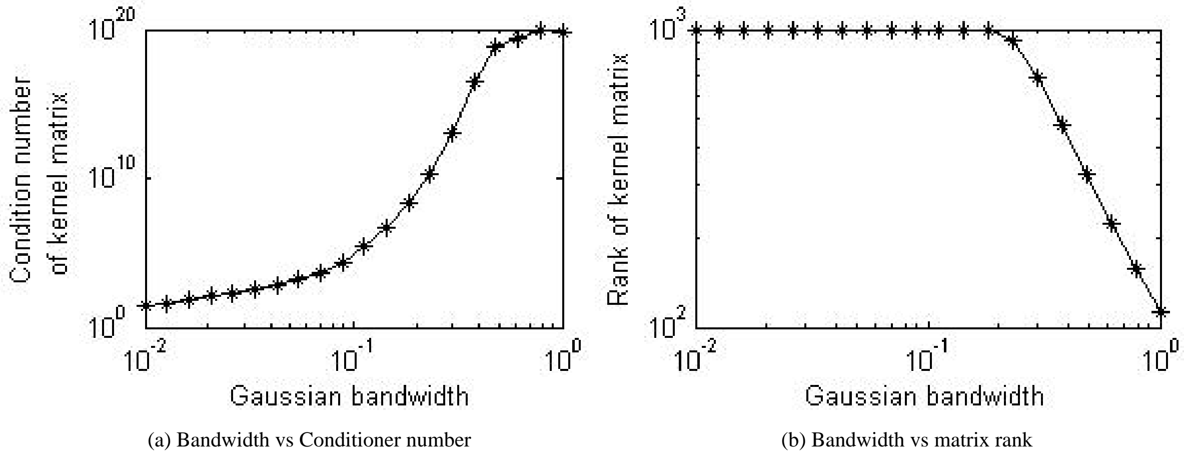
The key advantages of the proposed preconditioner is that, because $\mathbf{M}$ has a functional representation, given $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}, x_i \in R^d$ it is not necessary to explicitly construct the preconditioner $\mathbf{M}^{-1}$ as in conventional preconditioners. The key computation in a CG iteration is a matrix-vector product, $\mathbf{M}x$ and it is possible to accelerate this using approaches discussed in section 2. Because the preconditioners are required only to be approximate, the output accuracy will not be hampered by the use of single-precision computations as long as the outer iteration is evaluated with high accuracy. For lower dimensions (< 10) and large data-sizes ($>\sim 80,000$) where the FIGTREE performs better than GPUML, a low ϵ-FIGTREE (single-precision) can be used. Single-precision GPUML can be used in all other cases.

For a good trade-off between speed and accuracy of the FGMRES, we propose to use double-precision matrix-vector products for the outer iterations and single-precision matrix-vector products for the inner iterations.

3.2. Effect of preconditioner parameters. Before testing the FGMRES algorithm with the proposed preconditioner on various kernel methods, we explore the effect of the preconditioner parameters on the convergence. Because a restarted GMRES loses the vectors generated in previous iterations, we used GMRES without restarts in all our experiments, although this may increase the computational cost per iteration. The two parameters of the proposed preconditioner are the regularizer constant σ and the tolerance for the termination of the inner conjugate gradient iterations. We now explore the effect of these parameters on the performance of the preconditioner.

We generated random data points within a unit cube in 3D. By varying the bandwidth (h in Eq. 1.2) of the Gaussian kernel, we tested the convergence of the GMRES iterations with and without the proposed preconditioner for a 1000-sample dataset. Fig. 3.1a shows the conditioning of the Gaussian kernel matrix for various bandwidths (h) and Fig. 3.1b shows the corresponding ranks. We did not consider the rank-deficient cases in these experiments.

3.2.1. Effect of regularizer (σ). In this test, we evaluated the performance of our preconditioner for various values of the regularizer (σ). Note that for small values of the σ, the preconditioner $\mathbf{M}$ is closer to the actual matrix $\mathbf{K}$. However, when $\mathbf{K}$ is ill-conditioned, $\mathbf{M}$ will also be ill-conditioned, thus hindering the convergence of the inner



(a) Bandwidth vs Conditioner number

(b) Bandwidth vs matrix rank

Fig. 3.1: ***Left:*** *Effect of Gaussian bandwidth on condition number of* $\mathbf{K}$,***Right:*** *Rank of the kernel matrix for various bandwidths*

CG iterations. The iterations and time taken by the preconditioned and unpreconditioned approaches are shown in Figs. 3.2a and 3.2b. The maximum value the Gaussian kernel takes is 1 and hence a regularizer of 1 results in a preconditioner $\mathbf{M}$ which is largely varying from $\mathbf{K}$, therefore its convergence is slower than other regularizers. As the regularizer is decreased, there is no significant time/iteration difference between 10^{-3} and 10^{-6}. A regularizer of 10^{-3} is therefore more appropriate because the resulting $\mathbf{M}$ is better conditioned than 10^{-6}.

REMARK 3.4. *The choice of a regularizer involves a trade-off between the preconditioner's accurate representation of the kernel matrix and its desired conditioning.*

3.2.2. Effect of CG tolerance. In this evaluation, we tested the performance of the preconditioner for various tolerance of the inner CG iterations. The corresponding performances are shown in Figs. 3.2c and 3.2d. A very large CG tolerance (10^{-2}) results in an early termination and hence poor performance of the preconditioner (in fact there is no difference in the convergence by using such a preconditioner). There is a consistent difference in the convergence (number of iterations to converge) for tolerances 10^{-4} and 10^{-6}, but there is little to separate the time taken to converge in the two cases. This is because although the outer iterations converge faster, the CG with 10^{-6} tolerance takes more iterations to converge to the desired accuracy and hence an increased cost per outer iteration.

REMARK 3.5. *The choice of tolerance for CG iterations is a trade-off between the required accuracy of the solution of the preconditioner system (and hence the convergence of the outer iterations) and the related computational complexity.*

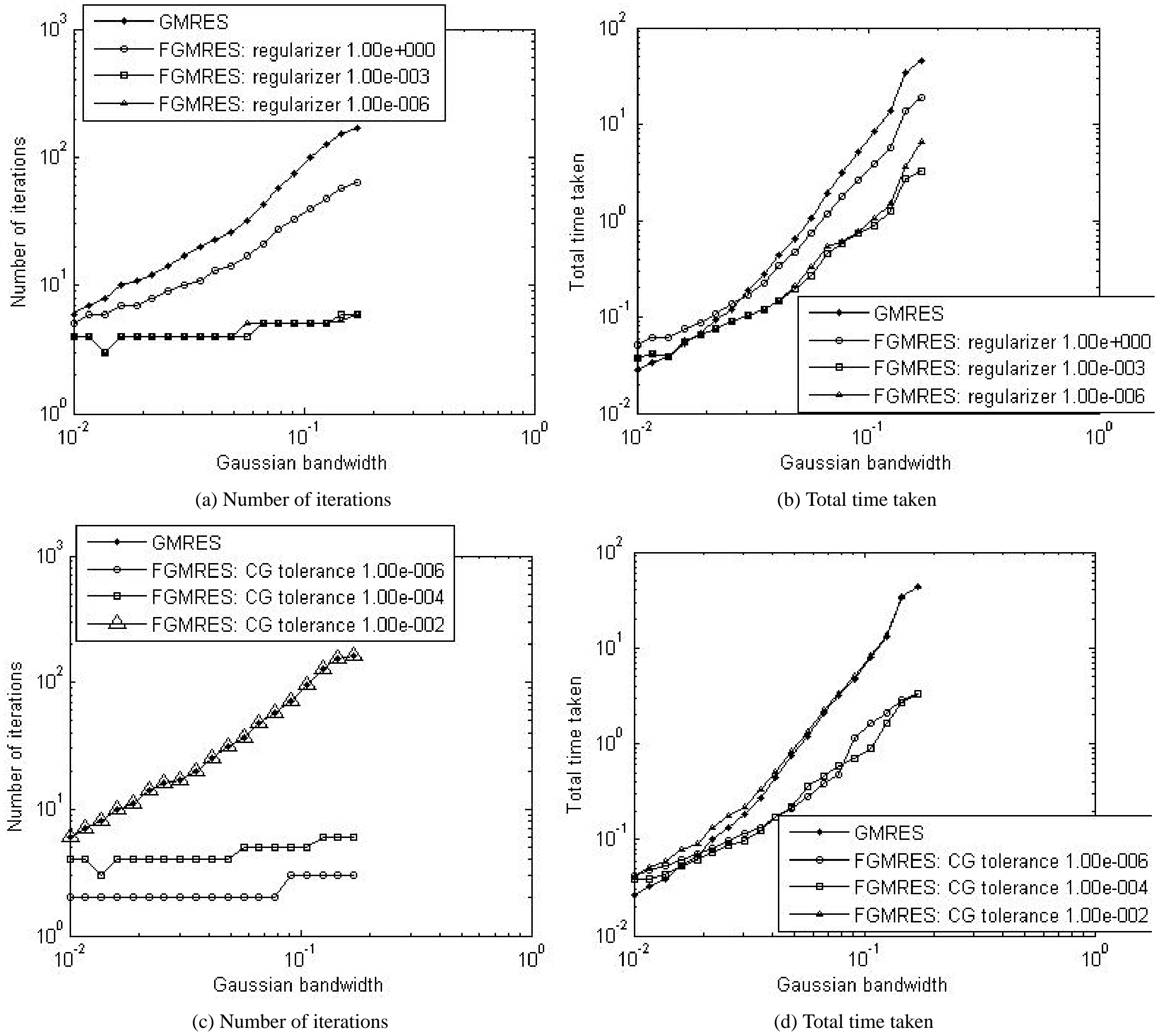
4. Experiments. We performed 3 experiments in order to test the performance of our preconditioner. In the first experiment, we tested the convergence of our approach with synthetic data. In the second experiment, we show the performance of our FGMRES for radial basis function (RBF) interpolation, throwing light on the choice of FIGTREE and GPUML. Finally, we compare the performance of FGMRES against a CG-based Gaussian process regression, thus showing the usability of the proposed approach on standard datasets.

4.1. Experiment1: Test of convergence. In this experiment, we tested the convergence of our preconditioner by comparing FGMRES with unpreconditioned CG and GMRES approaches. Because the kernel matrices are symmetric positive semi-definite, CG is the widely preferred choice because of its lower cost per iteration. But we show in this experiment, that FGMRES will beat the performance of CG as $\mathbf{K}$ becomes ill-conditioned.

For our preconditioner, we set the regularizer σ to 10^{-3} and the tolerance of the inner CG iterations to 10^{-4}. Data points were generated randomly in a 3-dimensional unit cube, and the convergence was tested on 1000 and 5000 points. The results are shown in Fig. 4.1. The number of iterations of our preconditioned approach is always less than those for unpreconditioned GMRES and CG approaches.

The computational cost per iteration is the least for CG compared to GMRES and FGMRES. However, as the kernel matrix becomes ill-conditioned, the number of CG iteration increases significantly; for highly ill conditioned matrices, it can be seen that our FGMRES outperforms even the CG. As the data-size increases, FGMRES outperforms CG even for smaller bandwidth as seen from Fig. 4.1d.

4.2. Experiment2: Radial basis function (RBF) interpolation. RBF interpolation [5, 18] can be used for reconstruction of damaged images, filling gaps and for restoring missing data in images. The key computation in RBF interpolation involves the solution of a kernel system. Most of the data in this application are 2 or 3 dimensional,



(a) Number of iterations

(b) Total time taken

(c) Number of iterations

(d) Total time taken

Fig. 3.2: *Effect of regularizer (σ) and error tolerance of inner CG iterations on the preconditioner performance.* ***Top:*** *Regularizer (σ)* ***Bottom:*** *Tolerance of inner CG iteration*

where FIGTREE outperforms GPUML for some cases (Fig. 2.2a). In this experiment, we use the proposed FGMRES with FIGTREE and GPUML for the inner and outer iterations (resulting in 4 such combinations; FIGTREE for both inner and outer Krylov iterations, GPUML for both inner and outer Krylov iterations, FIGTREE for outer Krylov, GPUML for inner Krylov and vice-versa) and study the resulting performance in each case.

4.2.1. Image Restoration. In this experiment, 80% of the pixels in a 2D image were removed randomly and the remaining pixels were used to interpolate data back using the Gaussian kernel. The interpolations were performed on standard images (sizes 256×256 and 512×512). The original image, the noisy image input to the interpolation and the interpolated image for a *Baboon* image are shown in Fig. 4.2. The performances of *Cameraman*, *Lena* and *Baboon* images are tabulated in Table 4.1. Although for smaller image (256×256), GPUML for inner and outer iterations has the best performance, for larger images, FIGTREE for the outer iteration and GPUML for inner Krylov iteration performs the best, in coherence with the proposed strategy in Section 3.1. The peak signal-to-noise ratio for a processed image (R) gives the objective quality of the image with respect to the original image (I) and is given by,

$$10\log_{10}\left(\frac{\left(\max_{(i,j)} I(i,j)\right)^2}{\sqrt{\left(\frac{1}{rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\left[I(i,j)-R(i,j)\right]^2\right)}}\right) \tag{4.1}$$

where, the images I and R are of size $r \times c$. A PSNR value above 20dB indicates good image quality. The PSNR for the noisy and restored image for each of the restored images is shown in Table 4.1.

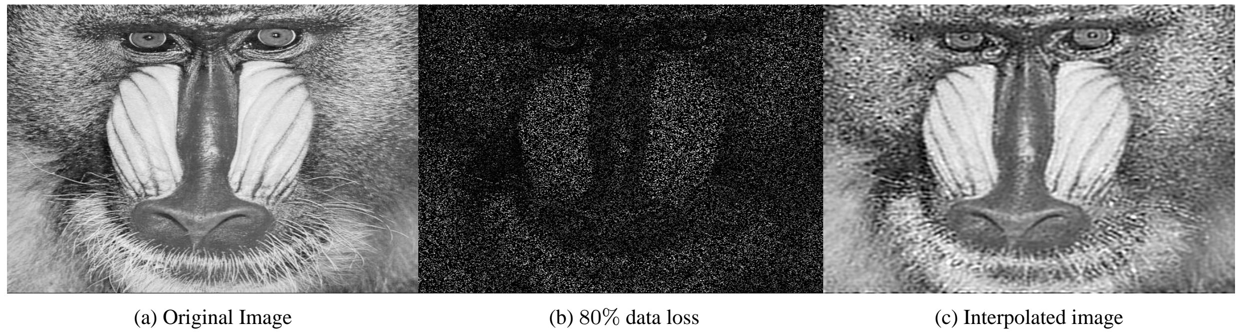
(a) 1000 points, 3 dimension; Total iterations

(b) 1000 points, 3 dimension; Total time taken

(c) 5000 points, 3 dimension; Total iterations

(d) 5000 points, 3 dimension; Total time taken

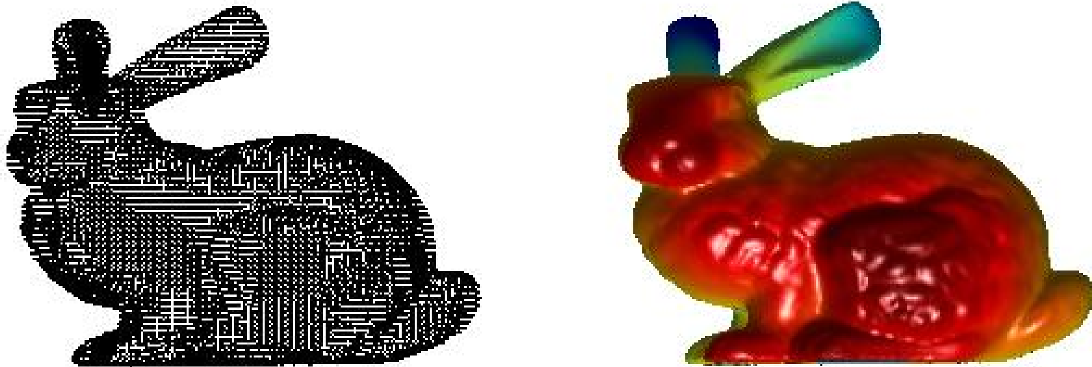
Fig. 4.1: *Performance of our preconditioner: regularizer* $\sigma = 0.1$*, tolerance for inner CG iteration:* 10^{-6}



(a) Original Image (b) 80% data loss (c) Interpolated image

Fig. 4.2: *RBF image restoration for Baboon image with* 80% *data loss*

4.2.2. Implicit function fitting [18]. In this experiment, we used a point cloud (of the "Stanford" bunny) with 34834 points (Fig. 4.3a), extended to 104502 points by adding points along the normal inside and outside. Interpolation here is similar to the image restoration, however the data is 3 dimensional. After the interpolation coefficients were determined, the interpolants were evaluated at 8×10^6 points in a regular grid of $200 \times 200 \times 200$ from which an isosurface (Fig. 4.3b) was generated using standard routines [18]. Double-precision data in this case is too large to fit in the GPU; therefore GPUML for outer Krylov iterations was not considered. The performances for the other combinations are shown in Table. 4.1. It can be seen that because the data size is $> 100,000$ here, FIGTREE for inner and outer iterations performs significantly better than other combinations, again agreeing with the strategy proposed in Section 3.1.



(a) Point Cloud (b) Implicit fitting

Fig. 4.3: *RBF function fitting of a bunny point cloud*

Data	Type	Method combinations				
Outer Krylov		FIGTREE		GPUML		# iterations
Inner Krylov		FIGTREE	GPUML	FIGTREE	GPUML	
Cameraman	Image (256×256)	62s	21s	57s	**15*s***	17
Lena	Image (512×512)	93s	**53*s***	109s	70s	9
Baboon	Image (512×512)	135s	**76*s***	156s	96s	15
Bunny	Point Cloud ($104,502$ points)	**49*s***	262s	$--$	$--$	9

Table 4.1: *Comparison of the performance of FGMRES with FIGTREE and GPUML for outer and inner Krylov iterations.* ***Cameraman image:*** *the PSNR of the input image was* 6.55*dB; the PSNR of the restored images by all the methods were* 20.86*dB.* ***Lena Image:*** *the PSNR of the input image was* 6.63*dB; the PSNR of the restored images by all the methods were* 28.75*dB.* ***Baboon image:*** *the PSNR of the input image was* 5.49*dB; the PSNR of the restored images by all the methods were* 18.08*dB.* ***Bunny:*** *the mean error in the interpolation was* $< 10^{-5}$ *in all the cases.*

4.3. Experiment3: Gaussian process regression. Gaussian process regression is a probabilistic kernel regression approach which uses the prior that the regression function ($f(X)$) is sampled from a Gaussian process. For regression, given a set of datapoints $D = \{X, y\}_{i=1}^{N}$, where X is the input and y 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 σ^2. Rasmussen et al. [9] use the Gaussian process prior with a zero mean function and a covariance function defined by a kernel $K(x, x')$ which is the covariance between x and x', i.e. $f(x) \sim GP(0, K(x, x'))$. They further show that with this prior, the posterior of the output y is also Gaussian with mean m and covariance V given by,

$$m = k(x*)^T (K + \sigma^2 I)^{-1} y$$
$$V = K(x_*, x_*) - k(x_*)^T (K + \sigma^2 I)^{-1} k(x_*)$$

where $x*$ is the input at which prediction is required and $k(x*) = [K(x_1, x*), K(x_2, x*) \ldots, K(x_N, x*)$. Here m gives the prediction at $x*$ and V the variance estimate of prediction, and the "inverses" here imply the solution of the corresponding linear system.

The parameters associated with the kernels (eg. h in Eq. 1.2) are called the *hyperparameters* of the Gaussian process and there are different approaches to estimate these [9]. Given the hyperparameters, the core operation in Gaussian processes involves solving a linear system involving the kernel covariance matrix. Gibbs et al. [3] suggest a *conjugate gradient* based approach to solve the Gaussian process problem. Alternatively, it is also possible to use FGMRES to solve the kernel system. Table 4.2 shows a comparison of the performance of Gaussian process regression based on our FGMRES and CG [3] on various standard datasets [17]. We use FIGTREE for the outer iterations (both in FGMRES and CG [3]) and compare the performance with both FIGTREE and GPUML based inner iterations for FGMRES.

As the size and dimension of dataset increase, the performance of the GPUML-based inner iterations improves significantly over the other two approaches (in coherence with Section 3.1). This is because, for larger problems, cost per iteration in both CG and FGMRES increases, and thus a FGMRES that converges faster with lower computation

Dataset	Dimension	Data size	Gibbs et al.	FGMRES (FIGTREE)	FGMRES (GPUML)
Diabetes	3	43	0.04s (8)	0.13s (**5**)	**0.07s** (**5**)
Boston housing	14	506	1.05s (52)	3.22s (**6**)	**0.90s** (**6**)
Stock domain	10	950	4.51s (84)	12.89s (**9**)	**2.25s** (**9**)
Abalone	8	4177	26.66s (39)	54.78s (**3**)	**9.78s** (5)
Computer activity	13	8192	343.16s (89)	514.98s (**7**)	**113.28s** (12)
California housing	9	20640	3183.16s (144)	4911.03s (**11**)	**994.08s** (19)

Table 4.2: *Performance of our FGMRES based Gaussian process regression against a CG based approach in [3]. Total time taken for prediction is show 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*

time becomes significantly better than the CG-based approach. This experiment further illustrates the utility of the proposed FGMRES on a standard dataset. Note that, for larger datasets, although GPUML-based preconditioner has a good computational performance, the convergence of the FIGTREE-based preconditioner is marginally better because of the single-precision round-off errors on the GPU.

5. Conclusion. In this paper, we have proposed a novel *Tikhonov regularized preconditioner* for FGMRES (Algorithm 1) to solve a linear system with kernel matrix. We have discussed two classes of algorithms (FIGTREE and GPUML) to accelerate the kernel matrix-vector product in each iteration of FGMRES and have provided strategies to select an optimal approach for a particular data. The convergence with proposed preconditioner is shown to be an order of magnitude better than the unpreconditioned approach. Further, the proposed preconditioner can also be accelerated using the fast matrix-vector product algorithms (FIGTREE and GPUML), resulting in a computationally efficient solver. The performance is further illustrated in popular learning approaches namely, radial basis function interpolation and Gaussian process regression. There is an improvement of upto $\sim$ 8X in the number of iterations to converge and ~ 3.5X in the total time taken compared to a conjugate gradient based approach. The core preconditioning strategy proposed here will soon be released as an open source package.

REFERENCES

[1] C. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag New York, Inc., 2006.
[2] 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.
[3] M. Gibbs and D. Mackay, *Efficient implementation of Gaussian processes*, tech. report, 1997.
[4] L. Greengard and J. Strain, *The fast Gauss transform*, in SIAM Journal of Scientic and Statistical Computing, vol. 12, pp. 79–94.
[5] N. Gumerov and R. Duraiswami, *Fast radial basis function interpolation via preconditioned Krylov iteration*, SIAM J. Sci. Comput., 29 (2007), pp. 1876–1899.
[6] N. Gumerov and R. Duraiswami, *A broadband fast multipole accelerated boundary element method for the three dimensional Helmholtz equation*, The Journal of the Acoustical Society of America, 125 (2009), pp. 191–205.
[7] D. Lee, A. Gray, and A. Moore, *Dual-tree fast Gauss transforms*, in Advances in Neural Information Processing Systems 18, 2006, pp. 747–754.
[8] 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. Available at `http://sourceforge.net/projects/figtree/`.
[9] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, The MIT Press, 2005.
[10] V.C. Raykar and R. Duraiswami, *Fast large scale Gaussian process regression using approximate matrix-vector products*, Learning workshop, (2007).
[11] V.C. Raykar and R. Duraiswami, *The improved fast Gauss transform with applications to machine learning*, in Large Scale Kernel Machines, 2007, pp. 175–201.
[12] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Comput., 14 (1993), pp. 461–469.
[13] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, 2003.
[14] V. Simoncini and D.B. Szyld, *Flexible inner-outer Krylov subspace methods*, SIAM J. Numer. Anal., 40 (2002), pp. 2219–2239.
[15] B.V. Srinivasan and R. Duraiswami, *Scaling kernel machine learning algorithm via the use of GPUs*, in GPU Technology Conference, NVIDIA Research Summit. Available at `http://www.umiacs.umd.edu/~balajiv/GPUML.htm`.
[16] D. Thanh-Nghi and V. Nguyen, *A novel speed-up SVM algorithm for massive classification tasks*, July 2008, pp. 215–220.
[17] L. Torgo. Available at `http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html`.
[18] G. Turk and J.F. O'brien, *Modelling with implicit surfaces that interpolate*, ACM Trans. Graph., 21 (2002), pp. 855–873.