FAST ALGORITHMS FOR INTEGRAL EQUATIONS

by

Wing-Fai NG

Thesis

Submitted to the Faculty of the Graduate School of

The Chinese University of Hong Kong

(Division of Mathematics)

In partial fulfillment of the requirements

for the Degree of

Master of Philosophy

DECLARATION

The author declares that the thesis represents his own work based on the ideas suggested by Dr. Raymond H.F. Chan, the author's supervisors. All the work is done under the supervision of Dr. Raymond H.F. Chan during the period 1994-1996 for the degree of Master of Philosophy at The Chinese University of Hong Kong. The work submitted has not been previously included in a thesis, dissertation of report submitted to any institution of a degree, diploma or other qualifications.

[illegible]

Wing-Fai NG

ACKNOWLEDGMENTS

I wish to express my sincere and deepest gratitude to my supervisor Dr. Raymond H.F. Chan for his inspired guidance and discussions during the course of my M.Phil. studies at the Chinese University of Hong Kong. I am deeply grateful to Prof. T.M. Shih of the Department of Applied Mathematics, the Hong Kong Polytechnic University for his invaluable help and encouragement. Besides, I would also like to thank all my academic brothers — Dr. X.Q. Jin, Dr. K.P. Ng, Mr. C.K. Wong, Mr. W.K. Ching, Dr. F.R. Lin, Mr. H.C. Chan, Mr. T.M. Tso, Mr. H.W. Sun, Mr. H.M. Zhou, Mr. C.P. Cheung and Mr. K.W. Mak for their very helpful discussions and supports.

CONTENTS

Abstract

In this thesis, we present a fast algorithm (named the *fast dense matrix method*) to solve non-convolution type integral equations by conjugate gradient type methods. Fast dense matrix method is a fast multiplication scheme that provides a *dense* matrix approximation $\mathbf{A}$ to a given integral equation or its discretization matrix. Our method is based on using low-order polynomial interpolation of the kernel function and a divide-and-conquer strategy. The construction cost and storage requirement of the approximation $\mathbf{A}$ are both of $O(n)$ complexity and that the matrix-vector multiplication $\mathbf{Ax}$ can be done in $O(n \log n)$ operations, where n is the size of the matrix $\mathbf{A}$. Thus our method is suitable for conjugate gradient type methods, since the complexity of solving the discretized system can be done in $O(n \log n)$ operations per iteration. Our numerical results indicate that the algorithm is very accurate, and is very stable for high degree polynomial interpolation.

However, for some integral equations, such as the Fredholm type integral equations of the first kind, the system will be ill-conditioned and therefore the convergence rate of the method may be slow. In these cases, preconditioning is required to speed up the convergence rate of the method. The preconditioner we consider in this thesis is the optimal circulant preconditioner, which is the minimizer of $||\mathbf{C} - \mathbf{A}||_F$ in Frobenius norm over all circulant matrices $\mathbf{C}$. It can be obtained by taking the arithmetic average of the entries of $\mathbf{A}$. Thus for general dense matrices, the cost of constructing the preconditioner is of

$O(n^2)$ operations. In this thesis, we give an $O(n \log n)$ method of constructing the preconditioner for matrices $\mathbf{A}$ arising from the fast dense matrix method. Applications to first kind integral equations from potential equations are also given, in which preconditioning has been shown to make the ill-conditioned problem well-conditioned. Besides, numerical examples are given to illustrate the fast construction of the preconditioner and the fast convergence of the preconditioned systems.

The thesis is based on the following two papers, which will be referred to in the text by Paper I and Paper II.

Paper I. [6] R. Chan, F. Lin and W. Ng, *Fast Dense Matrix Method for the Solution of Integral Equations of the Second Kind*, submitted to SIAM J. Sci. Comput..

Paper II. [8] R. Chan, W. Ng and H. Sun, *Fast Construction of Optimal Circulant Preconditioners for Matrices from Fast Dense Matrix Method*, submitted to SIAM J. Sci. Comput..

Introduction

Solution of integral equations is a much studied subject and various direct and iterative methods have been proposed for their numerical solutions, see for instance Baker [2] and Delves and Mohamed [10]. However, one overriding drawback of these methods is the high cost of working with the associated dense matrices. For problems discretized with n quadrature points, classical direct methods such as Gaussian elimination method requires $O(n^3)$ operations to obtain the numerical solutions. For iterative methods such as the conjugate gradient method (see Golub and Van Loan [12]), each iteration requires $O(n^2)$ operations. Therefore even for well-conditioned problems, the method requires $O(n^2)$ operations, which is often prohibitive for large-scale problems.

In recent years, a number of algorithms for the fast numerical solutions of integral equations have been developed, see for instance [13, 14, 3, 1]. These methods try to obtain an n-by-n approximation matrix $\mathbf{A}$ to the given integral equation such that the matrix-vector multiplication of $\mathbf{A}$ with any vector can be done in $O(n)$ or $O(n \log n)$ operations, depending on the smoothness of the kernel functions. For example, the fast multipole method proposed in [13] combines the use of low-order polynomial interpolation of the kernel function with a divide-and-conquer strategy. For kernel functions that are Coulombic or gravitational in nature, it results in an order $O(n)$ algorithm for the matrix-vector multiplications. In [3], an $O(n \log n)$ algorithm is developed by exploiting the connections between the use of wavelets and their applications on Calderon-Zygmund operators. In [1], wavelet-like bases are used to transform the dense discretization matrices into sparse matrices, which then is inverted by the Schulz method. The complexity of the resulting algorithm is bounded by $O(n \log^2 n)$.

In Paper I, a fast matrix-vector multiplication scheme, named the fast dense matrix method, is developed. In this paper, we consider Fredholm integral equations of the second kind that are studied in [1], i.e. the kernel functions are either smooth, non-oscillatory and possessing only finite number of singularities or products of such functions with highly oscillatory coefficient functions. Our approximation scheme starts with the same approach as in [1]. More precisely, we write the discretized dense matrix $\mathbf{B}$ of the integral equation as the sum of a sequence of block matrices where the blocks are of increasing size. Then we use polynomial interpolation as in [13, 1] for each of the block matrix. However, we do not use wavelet-like bases as in [1] to further approximate the operator to get a sparse representation. Our resulting approximation $\mathbf{A}$ will therefore be a dense matrix in general.

However, we show in paper I that the approximation $\mathbf{A}$ can be obtained in $O(n)$ operations and only $O(n)$ storage is required. We also show that matrix-vector multiplication of the form $\mathbf{Ax}$ can be done in $O(n \log n)$ operations. Thus for second-kind integral equations, which are in general well-conditioned problems, solving the approximated systems by conjugate gradient type methods requires only $O(n \log n)$ operations. We have applied our scheme to kernel functions tested in [1] and also to kernel functions where the algorithm in [1] is inapplicable. Our numerical results show that our method is more accurate and is stable even when higher degree polynomials are used in the approximation.

However, for some integral equations, such as the Fredholm type integral equations of the first kind, the convergence rate of conjugate gradient method may be slow and preconditioning are required to accelerate the convergence rate. Optimal circulant preconditioners have been proposed and used successfully in preconditioning various kinds of integral equations, see for instance [11, 4, 5, 9]. Given the discretization matrix $\mathbf{B}$ of the integral equation, we know that if the optimal circulant preconditioner of $\mathbf{B}$, denoted by $c(\mathbf{B})$, is used as a preconditioner in the preconditioned conjugate gradient method, then we have to compute the product $[c(\mathbf{B})]^{-1}\mathbf{x}$ in each iteration for some vector $\mathbf{x}$. By using the circulant structure of $c(\mathbf{B})$, this product can be obtained efficiently in $O(n \log n)$ operations by using fast Fourier transforms, see for instance Chan and Ng [7].

Also, from Tyrtyshnikov [15], we know that the construction cost of $c(\mathbf{B})$ is of $O(n^2)$ operations for general matrices $\mathbf{B}$, and this cost count can be reduced to $O(n)$ when $\mathbf{B}$ is a band matrix or a Toeplitz matrix. Therefore, the optimal circulant preconditioner has been used in solving convolution type integral equations, see [11, 4] for instance. Since the discretization matrix arising from these integral equations are Toeplitz matrices if rectangular quadrature rule is used, so solving such circulant preconditioned Toeplitz system by preconditioned conjugate gradient method requires $O(n \log n)$ operations per iteration.

But for non-convolution type integral equations, where the discrete matrices are no longer Toeplitz. Convergence analysis of optimal circulant preconditioners on solving this kind of integral equations has also been studied, see [5, 9] for instance. For example, for boundary integral equations arising from potential equations, which are ill-conditioned, non-convolution type integral equations with condition number increasing like $O(n)$, the preconditioned systems have been shown to be well-conditioned, see Chan, Sun and Ng [9]. However, since $\mathbf{B}$ is dense, forming the optimal circulant preconditioner $c(\mathbf{B})$ will require $O(n^2)$ operations.

To overcome this difficulty, we developed in Paper II a fast algorithm for constructing the optimal circulant preconditioner $c(\mathbf{A})$ for matrix $\mathbf{A}$ that are obtained from the fast dense matrix method. By using the special structure of the approximation matrix $\mathbf{A}$, the circulant matrix $c(\mathbf{A})$ can be obtained in $O(n \log n)$ operations. Thus, used in conjunction with the fast dense matrix method, the circulant preconditioned conjugate gradient method for integral equations is a method of $O(n \log n)$ complexity.

To demonstrate the accuracy and stability of the fast dense matrix method, and its effectiveness in performing matrix-vector multiplications, a variety of numerical examples on second kind Fredholm integral equations are given in Paper I. To illustrate the efficiency of our construction scheme of the optimal circulant preconditioner and the effectiveness of using circulant preconditioners, numerical examples on solving first kind integral equations from potential equations are given in Paper II.

References

[1] B. Alpert, G. Beylkin, R. Coifman and V. Rokhlin, *Wavelets for the Fast Solution of Second-Kind Integral Equations*, SIAM J. Sci. Comput., 14 (1993), 159–184.

[2] C. Baker, *The Numerical Treatment of Integral Equations*, Clarendon Press, Oxford, 1977.

[3] G. Beylkin, R. Coifman and V. Rokhlin, *Fast Wavelet Transforms and Numerical Algorithms I*, Comm. Pure Appl. Math., 46(1991), 141–183.

[4] R. Chan, X. Jin and M. Ng, *Circulant Integral Operators as Preconditioners for Wiener-Hopf Equations*, Integr. Equat. Oper. Theory, 21 (1995), 12–23.

[5] R. Chan and F. Lin, *Preconditioned Conjugate Gradient Methods for Integral Equations of the Second Kind Defined on the Half-Line,* J. Comput. Math., to appear.

[6] R. Chan, F. Lin and W. Ng, *Fast Dense Matrix Method for the Solution of Integral Equations of the Second Kind*, submitted.

[7] R. Chan and M. Ng, *Conjugate Gradient Methods for Toeplitz Systems*, SIAM Review, to appear.

[8] R. Chan, W. Ng and H. Sun, *Fast Construction of Optimal Circulant Preconditioner for Matrices from Fast Dense Matrix Method*, submitted.

[9] R. Chan, H. Sun and W. Ng, *Circulant Preconditioner for Ill-Conditioned Boundary Integral Equations from Potential Equations*, submitted.

[10] L. Delves and J. Mohamed, *Computational Methods for Integral Equations*, Cambridge University Press, Cambridge, 1985.

[11] I. Gohberg, M. Hanke, and I. Koltracht, *Fast Preconditioned Conjugate Gradient Algorithms for Wiener-Hopf Integral Equations*, SIAM J. Numer. Anal., 31 (1994), 429–443.

[12] G. Golub and C. Van Loan, *Matrix Computations*, 2nd ed., John Hopkins University Press, Baltimore, 1989.

[13] L. Greengard and V. Rokhlin, *A Fast Algorithm for Particle Simulations*, J. Comput. Phys., 73(1987), 325–348.

[14] L. Reichel, *Fast Solution Methods for Fredholm Integral Equations of the Second Kind*, Numer. Math., 57(1989), 719–736.

[15] E. Tyrtyshnikov, *Optimal and Super-optimal Circulant Preconditioners,* SIAM Matrix Anal. Appl., 13 (1992), 459–473.

Fast Dense Matrix Method for the Solution of Integral Equations of the Second Kind

Abstract

We present a fast algorithm based on polynomial interpolation to approximate matrices arising from discretization of second-kind integral equations where the kernel function is either smooth, non-oscillatory and possessing only a finite number of singularities or a product of such function with a highly oscillatory coefficient function. Contrast to wavelet-like approximations, our approximation matrix is not sparse. However, the approximation can be constructed in $O(n)$ operations and requires $O(n)$ storage, where n is the number of quadrature points used in the discretization. Moreover, the matrix-vector multiplication cost is of order $O(n \log n)$. Thus our scheme is well suitable for conjugate gradient type methods. Our numerical results indicate that the algorithm is very accurate and stable for high degree polynomial interpolation.

AMS(MOS) subject classifications. 45L10, 65R20.

Key Words. Fredholm integral equation, polynomial interpolation.

1 Introduction

Solution of integral equations of the second kind is a much studied subject and various direct and iterative methods have been proposed for their numerical solutions, see [4] for instance. However, one overriding drawback of these methods is the high cost of working with the associated dense matrices.

For problems discretized with n quadrature points, classical direct methods such as Gaussian elimination method requires $O(n^3)$ operations to obtain the numerical solutions. For iterative methods such as the conjugate gradient method (see [5]), each iteration requires $O(n^2)$ operations. Therefore even for well-conditioned problems, the method requires $O(n^2)$ operations, which for large-scale problems is often prohibitive.

In recent years, a number of algorithms for the fast numerical solutions of integral equations have been developed, see for instance [6, 9, 2, 1]. The fast multipole method proposed in [6] combines the use of low-order polynomial interpolation of the kernel function with a divide-and-conquer strategy. For kernel functions that are Coulombic or gravitational in nature, it results in an order $O(n)$ algorithm for the matrix-vector multiplications. In [9], the integral equation is discretized at Chebyshev points and the resulting matrix is approximated by a low-rank modification of the identity matrix which can be obtained in $O(n \log n)$ operations. However, the solution of the discretized system still requires $O(n^2)$ operations to obtain. In [2], an $O(n \log n)$ algorithm is developed by exploiting the connections between the use of wavelets and their applications on Calderon-Zygmund operators. In [1], wavelet-like bases are used to transform the dense discretization matrices into sparse matrices, which then is inverted by the Schulz method. The complexity of the resulting algorithm is bounded by $O(n \log^2 n)$.

In this paper, we will consider Fredholm integral equations of the second kind that are studied in [1], i.e. the kernel functions are either smooth, non-oscillatory and possessing only finite number of singularities or products of such functions with highly oscillatory coefficient functions, see (5). We will start with the same approach as in [1]. More precisely, we write the discretized dense matrix $\mathbf{A}$ as the sum of a sequence of block matrices where the blocks are of increasing size. Then we use polynomial interpolation as in [6, 1] for each of the block matrix. However, we do not use wavelet-like bases as in [1] to further approximate the operator to get a sparse representation. Our resulting approximation $\tilde{\mathbf{A}}$ will therefore be a dense matrix in general.

However, we show that the approximation $\tilde{\mathbf{A}}$ can be obtained in $O(n)$ operations and only $O(n)$ storage is required. We also show that matrix-vector multiplication of the form $\tilde{\mathbf{A}}\mathbf{x}$ can be done in $O(n \log n)$ operations. Thus for second-kind integral equations, which are in general well-conditioned problems, solving the approximated systems by conjugate gradient type methods requires only $O(n \log n)$ operations. We have applied our scheme to kernel functions tested in [1] and also to kernel functions where the algorithm in [1] is inapplicable. Our numerical results show that our method is more accurate and is stable even when higher degree polynomials are used in the approximation.

The outline of the paper is as follows. In §2, we recall the Nyström method for the numerical solution of integral equations. In §3 we derive our procedure in approximating integral operators. In §4, we discuss the construction cost of the approximation, the matrix-vector multiplication cost and the storage requirement. A variety of numerical examples are given in §5 to demonstrate the accuracy and stability of our proposed algorithm, its effectiveness in performing matrix-vector multiplications and the convergence of the conjugate gradient type methods for the approximate systems.

2 The Problem

Consider the linear Fredholm integral equation of the second kind:

$$f(x) - \int_0^1 a(x,t)f(t)dt = g(x), \quad x \in [0,1],$$

where the kernel function $a(x,t)$ is in $L^2[0,1]^2$ and the unknown function $f(x)$ and the right-hand side function $g(x)$ are in $L^2[0,1]$. Define the integral operator

$$(\mathcal{A}f)(x) \equiv \int_0^1 a(x,t)f(t)dt. \tag{1}$$

Then the integral equation can be written as

$$(\mathcal{I} - \mathcal{A})f = g, \tag{2}$$

where $\mathcal{I}$ is the identity operator.

As in [1], we concern ourselves first with kernel functions $a(x,t)$ which are analytic except at $x = t$, where it possesses an integrable singularity. It is well-known that integral operators with weakly singular kernels are compact operators, see for instance [8, Theorem 2.21]. Therefore the operator $\mathcal{I} - \mathcal{A}$ is well-conditioned unless 1 is the eigenvalue of $\mathcal{A}$, in which case, the operator is singular. Thus a good method for solving these well-conditioned equations is the conjugate gradient method or its variants, see for instance [5, 3]. They converge to the solution in a linear rate, cf. [7] and Table 2 in §5.

To find the solution numerically, we discretize (2) by Nyström's method (see [4]) at equally spaced points $(i-1)/(n-1)$, $i = 1, \ldots, n$, on $[0,1]$. This results in a matrix equation

$$(\mathbf{I} - \mathbf{A})\mathbf{f} = \mathbf{g}, \tag{3}$$

where **I** is the identity matrix, **g** is a given vector and **f** is the unknown vector. As in [1], we define the entries of the discretization matrix **A** to be

$$[\mathbf{A}]_{i,j} = \begin{cases} \frac{1}{n-1} a\left(\frac{i-1}{n-1}, \frac{j-1}{n-1}\right) & i \neq j, \\ 0 & i = j. \end{cases} \tag{4}$$

This corresponds to a primitive, trapezoid-like quadrature discretization of the integral operator $\mathcal{A}$.

We can solve (3) by using conjugate gradient type methods. However, for these methods to work efficiently, the matrix-vector multiplication $\mathbf{Ay}$ should be done fast for any vector $\mathbf{y}$. For **A** defined in (4), the multiplication requires $O(n^2)$ operations. In §3, we will find an approximation $\tilde{\mathbf{A}}$ of **A**, such that $\tilde{\mathbf{A}}\mathbf{y}$ can be computed fast in $O(n \log n)$ operations. The main idea is to take advantage of the smoothness of the kernel function $a(x,t)$. We know that smooth functions can be approximated quite accurately by polynomials. As an example mentioned in [1], for any $c > 0$, the function $\log|x|$ can be approximated within 4^{-9} accuracy on $[c, 2c]$ by using polynomials of degree at most 7. Since **A** possesses the same smoothness properties as that of the kernel $a(x,t)$, we see that if $a(x,t)$ is smooth, we can approximate **A** or submatrices of **A** by low rank matrices obtained via polynomial interpolation. This is done in the next section.

Besides smooth kernels, the authors in [1] have also studied a more general class of integral equations which are of the form:

$$f(x) - d(x)\int_0^1 a(x,t)f(t)dt = g(x), \quad x \in [0,1], \tag{5}$$

where $a(x,t)$ is again analytic except with an integrable singularity at $x = t$ and the coefficient function $d(x)$ can be oscillatory. These problems lie between the problems with smooth kernels and those with arbitrary oscillatory kernels. The corresponding operator equation is of the form

$$(\mathcal{I} - \mathcal{D}\mathcal{A})f = g, \tag{6}$$

where $\mathcal{A}$ is given in (1) and $\mathcal{D}$ is the operator defined by

$$(\mathcal{D}f)(x) \equiv d(x)f(x).$$

In [1], it is assumed that $d(x)$ is positive and a new wavelet bases is applied to the symmetrized operator $\mathcal{D}^{1/2}\mathcal{A}\mathcal{D}^{1/2}$ to obtain a sparse representation. However, we note that as long as $d(x)$ is bounded (no need to be positive), then $\mathcal{D}$ will be a bounded operator. Therefore if $a(x,t)$ is at most weakly singular, then $\mathcal{A}$ and hence $\mathcal{D}\mathcal{A}$ will be a compact operator. This is because product of bounded operator and compact operator is still compact, see for instance [10, Theorem 4.18]. Hence $\mathcal{I} - \mathcal{D}\mathcal{A}$ will still be well-conditioned and we can solve (6) by conjugate gradient type methods and the convergence rate will again be linear.

Clearly, the discretized equation of (6) is given by

$$(\mathbf{I} - \mathbf{D}\mathbf{A})\mathbf{f} = \mathbf{g}, \tag{7}$$

where $\mathbf{A}$ is given by (4) and $\mathbf{D}$ is a diagonal matrix with entries given by

$$[\mathbf{D}]_{i,i} = d(\frac{i-1}{n-1}), \quad i = 1, \cdots, n.$$

We will approximate $\mathbf{A}$ by low rank matrices to obtain the approximation $\tilde{\mathbf{A}}$, where the matrix-vector product $\tilde{\mathbf{A}}\mathbf{y}$ can be obtained in $O(n \log n)$ operations for any vector $\mathbf{y}$. Since $\mathbf{D}$ is diagonal, we see that the product $(\mathbf{I} - \mathbf{D}\tilde{\mathbf{A}})\mathbf{y}$ can be computed in $O(n \log n)$ operations.

3 The Approximation

The main idea in getting an approximation of $\mathbf{A}$ is to approximate $\mathbf{A}$ by low rank matrices. However, if the whole matrix $\mathbf{A}$ is approximated by one low rank matrix, the approximation will not be good in general, especially for kernels with diagonal singularities. Therefore a general idea is to divide $\mathbf{A}$ into blocks of different sizes and approximate each of the block by a low rank, say rank k, matrix. We will follow the partition as suggested in [1] (see also Figure 4 therein) and assume the size of $\mathbf{A}$ is given by $n = k \cdot 2^l$. Here k is a small integer that depends on the smoothness of the kernel function $a(\cdot,\cdot)$.

With the partition, the matrix $\mathbf{A}$ is cut into blocks of different sizes. The blocks near the main diagonal are of size k-by-k, those next remote are of size $2k$-by-$2k$, and so forth up to the largest blocks of size $2^{l-2}k$-by-$2^{l-2}k$. By grouping blocks of the same size into one matrix, we can express the matrix $\mathbf{A}$ as

$$\mathbf{A} = \mathbf{A}^{(0)} + \mathbf{A}^{(1)} + \cdots + \mathbf{A}^{(l-2)}, \tag{8}$$

where $\mathbf{A}^{(u)}$, $u = 0, \ldots, l-2$, consists only of blocks of size $2^u k$-by-$2^u k$. We can easily check that the number of nonzero blocks in $\mathbf{A}^{(u)}$ is given by

$$v_u = \begin{cases} 6 \cdot 2^l - 8 & u = 0, \\ 6(2^{l-1-u} - 1) & u = 1, \cdots, l-2. \end{cases} \tag{9}$$

We will denote these nonzero blocks by $\mathbf{A}^{(u,v)}$, $v = 1, \cdots, v_u$. As an illustration, for $l = 5$, $\mathbf{A}^{(2)}$ is of the form

$$\mathbf{A}^{(2)} = \left[\begin{array}{cccccccc}
 & & \boxed{\mathbf{A}^{(2,1)}} & \boxed{\mathbf{A}^{(2,2)}} & & & & \\
 & & & \boxed{\mathbf{A}^{(2,3)}} & & & & \\
\boxed{\mathbf{A}^{(2,10)}} & & & & \boxed{\mathbf{A}^{(2,4)}} & \boxed{\mathbf{A}^{(2,5)}} & & \\
\boxed{\mathbf{A}^{(2,11)}} & \boxed{\mathbf{A}^{(2,12)}} & & & & \boxed{\mathbf{A}^{(2,6)}} & & \\
 & & \boxed{\mathbf{A}^{(2,13)}} & & & & \boxed{\mathbf{A}^{(2,7)}} & \boxed{\mathbf{A}^{(2,8)}} \\
 & & \boxed{\mathbf{A}^{(2,14)}} & \boxed{\mathbf{A}^{(2,15)}} & & & & \boxed{\mathbf{A}^{(2,9)}} \\
 & & & & \boxed{\mathbf{A}^{(2,16)}} & & & \\
 & & & & \boxed{\mathbf{A}^{(2,17)}} & \boxed{\mathbf{A}^{(2,18)}} & &
\end{array}\right], \tag{10}$$

where each $\mathbf{A}^{(2,v)}$ is a $4k$-by-$4k$ matrix and other blocks not written out explicitly are zero blocks. As in [1], our idea is to write each block $\mathbf{A}^{(u,v)}$ in $\mathbf{A}^{(u)}$ as

$$\mathbf{A}^{(u,v)} = \tilde{\mathbf{A}}^{(u,v)} + \mathbf{E}^{(u,v)},$$

where $\tilde{\mathbf{A}}^{(u,v)}$ is of rank k and the error matrix $\mathbf{E}^{(u,v)}$ has small norm.

Our approach of constructing $\tilde{\mathbf{A}}^{(u,v)}$ is as follows. Let the entries of $\mathbf{A}^{(u,v)}$ be given by

$$[\mathbf{A}^{(u,v)}]_{i,j} = \frac{1}{n-1} a((i_0 + i - 1)h, (j_0 + j - 1)h), \quad 1 \le i, j \le 2^u k, \tag{11}$$

i.e. the entries of $\mathbf{A}^{(u,v)}$ are obtained by evaluating the kernel function $a(x,t)$ in the domain $[i_0h, i_0h+(2^uk-1)h]\times[j_0h, j_0h+(2^uk-1)h]$. Our idea is to map this domain to $[-1,1]^2$ and do our approximation there. On the domain $[-1,1]^2$, we will take k^2 samples of the function $a(\cdot,\cdot)$ at equally-spaced points and use the values to approximate the matrix $\mathbf{A}^{(u,v)}$. The resulting transformation matrix will be more stable and requires less storage to store, see §4 and §5.

Clearly, the transformation is given by

$$\left\{\begin{array}{rcl} \bar{x} & = & -1+2\dfrac{x-i_0h}{(2^uk-1)h}, \\ \bar{t} & = & -1+2\dfrac{t-j_0h}{(2^uk-1)h}, \end{array}\right. \tag{12}$$

where $(\bar{x},\bar{t})$ will be in $[-1,1]^2$. For simplicity, let us denote $\bar{a}(\bar{x},\bar{t}) = a(x,t)$. We then construct the k-by-k sample matrix $\bar{\mathbf{A}}^{(u,v)}$ by evaluating $\bar{a}(\cdot,\cdot)$ at k^2 equally-spaced points in $[-1,1]^2$. That is

$$[\bar{\mathbf{A}}^{(u,v)}]_{i,j} = \frac{1}{n-1}\bar{a}(-1+2\frac{i-1}{k-1}, -1+2\frac{j-1}{k-1}), \quad 1\le i,j\le k. \tag{13}$$

Since by the assumptions on $a(\cdot,\cdot)$, the function $\bar{a}(\cdot,\cdot)$ is smooth and non-oscillatory in $[-1,1]^2$, it can be approximated accurately by polynomials of small degree. In particular, we have

$$\frac{1}{n-1}\bar{a}(\bar{x},\bar{t}) \approx \sum_{r=1}^{k}\sum_{s=1}^{k}\lambda_{rs}^{(u,v)}\bar{x}^{r-1}\bar{t}^{s-1}, \tag{14}$$

where $\lambda_{rs}^{(u,v)}$ are the coefficients of the Taylor series expansion of the function on the left hand side.

Combining (13) and (14), we then have

$$[\bar{\mathbf{A}}^{(u,v)}]_{i,j} \approx \sum_{r=1}^{k}\sum_{s=1}^{k}\lambda_{rs}^{(u,v)}(-1+2\frac{i-1}{k-1})^{r-1}(-1+2\frac{j-1}{k-1})^{s-1}, \quad 1\le i,j\le k. \tag{15}$$

In matrix terms, we then have

$$\bar{\mathbf{A}}^{(u,v)} \approx \mathbf{P}_k^T \mathbf{\Lambda}^{(u,v)} \mathbf{P}_k \tag{16}$$

where $\mathbf{P}_k$ and $\mathbf{\Lambda}^{(u,v)}$ are k-by-k matrices with entries given respectively by

$$[\mathbf{P}_k]_{i,j} = (-1 + 2\frac{j-1}{k-1})^{i-1}, \quad 1 \le i, j \le k, \tag{17}$$

and

$$[\mathbf{\Lambda}^{(u,v)}]_{i,j} = \lambda_{ij}^{(u,v)}, \quad 1 \le i, j \le k.$$

We are now ready to approximate $\mathbf{A}^{(u,v)}$ by a rank k matrix. By (11), (12) and (14), we have

$$\begin{aligned} [\mathbf{A}^{(u,v)}]_{i,j} &= \frac{1}{n-1} a((i_0 + i - 1)h, (j_0 + j - 1)h) \\ &= \frac{1}{n-1} \bar{a}(-1 + 2\frac{i-1}{2^u k - 1}, -1 + 2\frac{j-1}{2^u k - 1}) \\ &\approx \sum_{r=1}^{k} \sum_{s=1}^{k} \lambda_{rs}^{(u,v)} (-1 + 2\frac{i-1}{2^u k - 1})^{r-1} (-1 + 2\frac{j-1}{2^u k - 1})^{s-1}, \quad 1 \le i, j \le 2^u k. \end{aligned}$$

In matrix terms, we then have the approximation:

$$\mathbf{A}^{(u,v)} \approx (\mathbf{P}^{(u)})^T \mathbf{\Lambda}^{(u,v)} \mathbf{P}^{(u)} \tag{18}$$

where $\mathbf{P}^{(u)}$ is the k-by-$2^u k$ matrix with entries given by

$$[\mathbf{P}^{(u)}]_{i,j} = (-1 + 2\frac{j-1}{2^u k - 1})^{i-1}, \quad 1 \le i \le k, 1 \le j \le 2^u k. \tag{19}$$

Thus the approximation $\tilde{\mathbf{A}}^{(u,v)}$ of $\mathbf{A}^{(u,v)}$ is obtained as follows:

1. Compute approximation $\tilde{\lambda}_{rs}^{(u,v)}$ of $\lambda_{rs}^{(u,v)}$ by requesting that the approximate equation (15) holds exactly for all k^2 sampled points. More precisely, we compute approximate coefficients matrix $\tilde{\mathbf{\Lambda}}^{(u,v)}$ of $\mathbf{\Lambda}^{(u,v)}$ by

(16), i.e.

$$\tilde{\Lambda}^{(u,v)} \equiv (\mathbf{P}_k^{-1})^T \bar{\mathbf{A}}^{(u,v)} \mathbf{P}_k^{-1}, \tag{20}$$

where $\bar{\mathbf{A}}^{(u,v)}$ and $\mathbf{P}_k$ are given by (13) and (17) respectively.

2. The approximation $\tilde{\mathbf{A}}^{(u,v)}$ of $\mathbf{A}^{(u,v)}$ is then given by (18), i.e.

$$\tilde{\mathbf{A}}^{(u,v)} \equiv (\mathbf{P}^{(u)})^T \tilde{\Lambda}^{(u,v)} \mathbf{P}^{(u)}, \tag{21}$$

where $\mathbf{P}^{(u)}$ is given by (19).

We emphasize that we do not have to form the $2^u k$-by-$2^u k$ matrix $\mathbf{A}^{(u,v)}$ in order to get its approximation $\tilde{\mathbf{A}}^{(u,v)}$. If only matrix-vector multiplications are required, as is in the case of conjugate gradient type methods (see [5]), then there is no need to explicitly form the approximation $\tilde{\mathbf{A}}^{(u,v)}$, and we only have to store $\tilde{\Lambda}^{(u,v)}$ and $\mathbf{P}^{(u)}$.

We remark that by transforming into the domain $[-1,1]^2$, both basis function matrices $\mathbf{P}_k$ and $\mathbf{P}^{(u)}$ are now independent of the index v of the block we are approximating. Numerical results show that our basis function matrices are less ill-conditioned than those we would obtain without the transformation. For example, when $k = 8$ and 14, condition numbers of $\mathbf{P}_k$ are about 10^2 and 10^5 respectively, whereas if no transformation is used, the numbers will exceed 10^9 and 10^{17} respectively and vary with v. In [1], interpolation are done by using polynomial basis functions that are shifted and scaled by methods different from ours. The resulting basis function matrices are also stable with condition numbers (which vary with different blocks and depends on n) about the same order as that of our $\mathbf{P}_k$.

Another important advantage of having this v independence in our basis function matrices is that we can use the same $\mathbf{P}^{(u)}$ for all $\tilde{\mathbf{\Lambda}}^{(u,v)}$. Recalling the block structure of each $\mathbf{A}^{(u)}$ (cf. (10)) and the approximation $\tilde{\mathbf{A}}^{(u,v)}$ of each block $\mathbf{A}^{(u,v)}$ in (21), we see that $\mathbf{A}^{(u)}$ can now be approximated by

$$\tilde{\mathbf{A}}^{(u)} = \left[\mathbf{I}_{2^{l-u}} \otimes (\mathbf{P}^{(u)})^T\right] \cdot \tilde{\mathbf{\Lambda}}^{(u)} \cdot \left[\mathbf{I}_{2^{l-u}} \otimes \mathbf{P}^{(u)}\right]. \tag{22}$$

Here $\mathbf{I}_{2^{l-u}}$ is the identity matrix of size 2^{l-u}, $\otimes$ is the Kronecker tensor product and $\tilde{\mathbf{\Lambda}}^{(u)}$ is a matrix having the same block structure as $\mathbf{A}^{(u)}$, except that the blocks $\mathbf{A}^{(u,v)}$ in $\mathbf{A}^{(u)}$ are of size $2^u k$ whereas the blocks $\tilde{\mathbf{\Lambda}}^{(u,v)}$ in $\tilde{\mathbf{\Lambda}}^{(u)}$ are of size k. As an illustration, the matrix $\tilde{\mathbf{A}}^{(2)}$ for $l = 5$ is of the form (cf. (10)):

$$\tilde{\mathbf{A}}^{(2)} = \left[\mathbf{I}_8 \otimes \left(\mathbf{P}^{(2)}\right)^T\right] \left[\begin{array}{cccccccc}
 & & \boxed{\tilde{\mathbf{\Lambda}}^{(2,1)}} & \boxed{\tilde{\mathbf{\Lambda}}^{(2,2)}} & & & & \\
 & & & \boxed{\tilde{\mathbf{\Lambda}}^{(2,3)}} & & & & \\
\boxed{\tilde{\mathbf{\Lambda}}^{(2,10)}} & & & & \boxed{\tilde{\mathbf{\Lambda}}^{(2,4)}} & \boxed{\tilde{\mathbf{\Lambda}}^{(2,5)}} & & \\
\boxed{\tilde{\mathbf{\Lambda}}^{(2,11)}} & \boxed{\tilde{\mathbf{\Lambda}}^{(2,12)}} & & & & \boxed{\tilde{\mathbf{\Lambda}}^{(2,6)}} & & \\
 & & \boxed{\tilde{\mathbf{\Lambda}}^{(2,13)}} & & & & \boxed{\tilde{\mathbf{\Lambda}}^{(2,7)}} & \boxed{\tilde{\mathbf{\Lambda}}^{(2,8)}} \\
 & & \boxed{\tilde{\mathbf{\Lambda}}^{(2,14)}} & \boxed{\tilde{\mathbf{\Lambda}}^{(2,15)}} & & & & \boxed{\tilde{\mathbf{\Lambda}}^{(2,9)}} \\
 & & & & \boxed{\tilde{\mathbf{\Lambda}}^{(2,16)}} & & & \\
 & & & & \boxed{\tilde{\mathbf{\Lambda}}^{(2,17)}} & \boxed{\tilde{\mathbf{\Lambda}}^{(2,18)}} & &
\end{array}\right] \left[\mathbf{I}_8 \otimes \mathbf{P}^{(2)}\right]$$

where each $\tilde{\mathbf{\Lambda}}^{(2,v)}$ is a k-by-k matrix.

Having defined the approximation matrix $\tilde{\mathbf{A}}^{(u)}$ for each $\mathbf{A}^{(u)}$, $u = 1, \cdots, l-2$, we can now define our approximation matrix $\tilde{\mathbf{A}}$ to the original matrix $\mathbf{A}$:

$$\tilde{\mathbf{A}} \equiv \mathbf{A}^{(0)} + \tilde{\mathbf{A}}^{(1)} + \tilde{\mathbf{A}}^{(2)} + \cdots + \tilde{\mathbf{A}}^{(l-2)}, \tag{23}$$

see (8). In §5, we will compute the difference $\mathbf{A} - \tilde{\mathbf{A}}$ for different kernel functions $a(x,t)$ and different k and n to illustrate the accuracy of our approximation.

We remark that in [1], after the approximation with low-order polynomials, the operator is further approximated (by throwing away entries less than a given threshold) by using wavelet-like basis functions so that the final approximation matrix is sparse. In our case, we stop at the approximation by low-order polynomials and the approximation matrices $\tilde{\mathbf{A}}$ are in general dense. However, we emphasize that if we are going to solve the linear system relating to $\tilde{\mathbf{A}}$ by conjugate gradient type methods, then only matrix-vector multiplications of the form $\tilde{\mathbf{A}}\mathbf{y}$ are required. In this case, there is no need to explicitly form $\tilde{\mathbf{A}}$. All we need is to store $\mathbf{A}^{(0)}$, $\tilde{\mathbf{\Lambda}}^{(u,v)}$ and $\mathbf{P}^{(u)}$, see §4. We will also show in §4 that the matrix-vector multiplication $\tilde{\mathbf{A}}\mathbf{y}$ can be obtained in $O(n \log n)$ operations.

4 Complexity Analysis

In this section, we consider the complexity of obtaining and storing a representation of $\tilde{\mathbf{A}}$ so that matrix-vector multiplications of the form $\tilde{\mathbf{A}}\mathbf{x}$ can be done fast. We also consider the cost of doing such matrix-vector multiplication. We first recall that by (23) and (22), we have

$$\tilde{\mathbf{A}}\mathbf{x} = \mathbf{A}^{(0)}\mathbf{x} + \sum_{u=1}^{l-2} \left[\mathbf{I}_{2^{l-u}} \otimes (\mathbf{P}^{(u)})^T\right] \cdot \tilde{\mathbf{\Lambda}}^{(u)} \cdot \left[\mathbf{I}_{2^{l-u}} \otimes \mathbf{P}^{(u)}\right] \mathbf{x}. \tag{24}$$

Thus we see that for the computation of $\tilde{\mathbf{A}}\mathbf{x}$, it suffices to form and store $\mathbf{A}^{(0)}$, $\tilde{\mathbf{\Lambda}}^{(u)}$ and $\mathbf{P}^{(u)}$. For simplicity, in the following, we count only the number of multiplications in the operation counts. The number of additions is of the same order.

Storage Requirement:

Forming	Storage	Explanation
$\mathbf{A}^{(0)}$	$(6 \cdot 2^l - 8)k^2$	$\mathbf{A}^{(0)}$ consists of $(6 \cdot 2^l - 8)$ blocks of size k, see (9).
$\tilde{\mathbf{\Lambda}}^{(u)}$	$6(2^{l-1-u} - 1)k^2$	$\tilde{\mathbf{\Lambda}}^{(u)}$ is a block matrix with $6(2^{l-1-u} - 1)$ nonzero blocks, see (9). Each nonzero block $\tilde{\mathbf{\Lambda}}^{(u,v)}$ is of size k, see (20).
$\mathbf{P}^{(u)}$	$2^u k^2$	$\mathbf{P}^{(u)}$ is a k-by-$2^u k$ matrix, see (19).

Thus the total storage requirement is

$$(6 \cdot 2^l - 8)k^2 + \sum_{u=1}^{l-2} \left\{6(2^{l-1-u} - 1)k^2 + 2^u k^2\right\} < 10 \cdot 2^l k^2 = 10nk.$$

Construction Cost:

To form $\tilde{\mathbf{\Lambda}}^{(u,v)}$ using (20), we first form the basis function matrix $\mathbf{P}_k$ (see (17)) and its inverse. This requires $O(k^3)$ operations and the matrices can be used for all u and v. For a given u and v, we form $\tilde{\mathbf{\Lambda}}^{(u,v)}$ in (20) by forming $\bar{\mathbf{A}}^{(u,v)}$ first. By using (13), this requires k^2 function evaluations of the kernel function $a(\cdot,\cdot)$. Then $\tilde{\mathbf{\Lambda}}^{(u,v)}$ is obtained by using (20) which requires $2k^3$ multiplications. Thus each $\tilde{\mathbf{\Lambda}}^{(u,v)}$ can be obtained in k^2 function evaluations and $2k^3$ multiplications. In the following table, f.e. denotes function evaluation of $a(\cdot,\cdot)$.

Forming	Complexity	Explanation
$\mathbf{A}^{(0)}$	$(6\cdot 2^l - 8)k^2$ f.e.	$\mathbf{A}^{(0)}$ consists of $(6\cdot 2^l - 8)$ blocks of size k, see (9).
$\tilde{\mathbf{\Lambda}}^{(u)}$	$6(2^{l-1-u} - 1)k^2$ f.e. and $12(2^{l-1-u} - 1)k^3$	$\tilde{\mathbf{\Lambda}}^{(u)}$ contains $6(2^{l-1-u} - 1)$ nonzero blocks, see (9) and each nonzero block $\tilde{\mathbf{\Lambda}}^{(u,v)}$ requires k^2 f.e. and $2k^3$ multiplications.
$\mathbf{P}^{(u)}$	$2^u k^2$	$\mathbf{P}^{(u)}$ is a k-by-$2^u k$ matrix, and its entries can be formed row-wise to avoid taking power, see (19).

Summing all these costs together, we conclude that the cost of forming $\mathbf{A}^{(0)}$, $\tilde{\mathbf{\Lambda}}^{(u)}$ and $\mathbf{P}^{(u)}$ for all $u = 1, \cdots, l-2$ is

$$\sum_{u=1}^{l-2} \left\{12(2^{l-1-u} - 1)k^3 + 2^u k^2\right\} < 6\cdot 2^l k^3 + 2^l k^2 = 6nk^2 + O(nk)$$

multiplications and

$$(6\cdot 2^l - 8)k^2 + \sum_{u=1}^{l-2} 6(2^{l-1-u} - 1)k^2 < 9\cdot 2^l k^2 = 9nk$$

function evaluations. In contrast, forming $\mathbf{A}$ requires n^2 function evaluations.

Cost of Matrix-Vector Multiplication:

We compute matrix-vector multiplication $\tilde{\mathbf{A}}\mathbf{x}$ as in (24).

Forming	Complexity	Explanation
$\mathbf{A}^{(0)}\mathbf{x}$	$(6\cdot 2^l-8)k^2$	Each of the $(6\cdot 2^l-8)$ blocks in $\mathbf{A}^{(0)}$ need to multiply with the corresponding subvector of length k.
$\mathbf{y}_u \equiv (\mathbf{I}_{2^{l-u}} \otimes \mathbf{P}^{(u)})\mathbf{x}$	$2^l k^2$	There are 2^{l-u} copies of $\mathbf{P}^{(u)}$ in $\mathbf{I}_{2^{l-u}} \otimes \mathbf{P}^{(u)}$ and multiply each copy of $\mathbf{P}^{(u)}$ to length $2^u k$ vector requires $2^u k^2$ multiplications.
$\mathbf{z}_u \equiv \tilde{\mathbf{\Lambda}}^{(u)}\mathbf{y}_u$	$6(2^{l-1-u}-1)k^2$	There are $6(2^{l-1-u}-1)$ nonzero blocks of size k in $\tilde{\mathbf{\Lambda}}^{(u)}$ and multiply each of them to length k vector requires k^2 multiplications.
$\left[\mathbf{I}_{2^{l-u}} \otimes (\mathbf{P}^{(u)})^T\right]\mathbf{z}_u$	$2^l k^2$	There are 2^{l-u} copies of $(\mathbf{P}^{(u)})^T$ in $\mathbf{I}_{2^{l-u}}\otimes(\mathbf{P}^{(u)})^T$ and multiply each copy of $(\mathbf{P}^{(u)})^T$ to length k vector requires $2^u k^2$ multiplications.

Combining all these together, we conclude that the total number of multiplications required in forming $\tilde{\mathbf{A}}\mathbf{x}$ is

$$(6\cdot 2^l-8)k^2+\sum_{u=1}^{l-2}\left\{6(2^{l-1-u}-1)k^2+2\cdot 2^l k^2\right\}<(2l+5)2^l k^2=(2l+5)nk,$$

which is of order $O(n\log n)$. In contrast, the cost of forming $\mathbf{Ax}$ is n^2 multiplications.

5 Numerical Examples

In this section, we show the efficiency and accuracy of our scheme by applying it to the following six kernel functions:

(i) $\log|x-t|$,

(ii) $\cos(xt^2)\log|x-t|$,

(iii) $\cos(xt^2)|x-t|^{-1/2}$,

(iv) $\cos(xt^2)|x-t|^{1/2}$,

(v) $(1+\frac{1}{2}\sin(100x))\log|x-t|$, and

(vi) $\sin(100x)\log|x-t|$.

Kernel functions (i) to (v) are examples tested in [1]. We note that kernels (v) and (vi) are kernels with a highly oscillatory coefficient function $d(x)$ that is equal to $1+\frac{1}{2}\sin(100x)$ and $\sin(100x)$ respectively, see (5). Obviously, both coefficient functions are bounded and therefore our algorithm works for both examples, see §2. We note however that since $d(x)$ for kernel (vi) is not positive, the algorithm in [1] is not applicable for this kernel.

The discretized equations for kernels (i) to (iv) are given by (3) and for kernels (v) and (vi), they are given by (7). Given a kernel function $a(\cdot,\cdot)$, we compute the matrix $\mathbf{A}$ as defined in (4) and its approximation $\tilde{\mathbf{A}}$ by (23). We measure the accuracy of the approximation by computing the relative error $||\mathbf{A}-\tilde{\mathbf{A}}||_F/||\mathbf{A}||_F$, where $||\cdot||_F$ is the Frobenius norm. All our computations are done in MATLAB on a SUN Sparc-20 workstation. Table 1 shows the results for different k and l. We recall that the size of the matrices is $n=k\cdot 2^l$. Thus the largest matrix we tried is of size 14,336-by-14,336.

Note that kernel functions (v) and (vi) give the same dense matrix approximation $\tilde{\mathbf{A}}$ as that of (i) as the $a(x,t)$ for all three kernels are all equal to $\log|x-t|$. Therefore, in Table 1, results for kernel functions (v) and (vi) are omitted. We see from Table 1 that our scheme provides a very accurate approximation $\tilde{\mathbf{A}}$ to the original matrix $\mathbf{A}$ even for small k like 8. We recall from §4 that the cost of forming $\tilde{\mathbf{A}}$ is of order $O(nk^2)$ operations whereas the cost of forming $\mathbf{A}$ is of $O(n^2)$ operations.

	$k = 4$	$k = 8$	$k = 11$	$k = 14$
l	$a(x,t) = \log\lvert x-t\rvert$			
4	7.69E-05	3.06E-08	1.79E-10	1.04E-11
6	1.14E-04	4.68E-08	2.78E-10	1.86E-11
8	1.30E-04	5.40E-08	3.22E-10	2.27E-11
10	1.36E-04	5.67E-08	3.38E-10	2.41E-11
l	$a(x,t) = \cos(xt^2)\log\lvert x-t\rvert$			
4	7.57E-05	3.10E-08	1.82E-10	1.18E-11
6	1.13E-04	4.73E-08	2.82E-10	1.93E-11
8	1.29E-04	5.44E-08	3.25E-10	2.23E-11
10	1.35E-04	5.71E-08	3.42E-10	2.33E-11
l	$a(x,t) = \cos(xt^2)\lvert x-t\rvert^{-1/2}$			
4	9.18E-05	5.24E-08	3.54E-10	1.12E-11
6	1.56E-04	9.09E-08	6.25E-10	1.55E-11
8	1.98E-04	1.17E-07	8.07E-10	1.87E-11
10	2.25E-04	1.34E-07	9.29E-10	2.01E-11
l	$a(x,t) = \cos(xt^2)\lvert x-t\rvert^{1/2}$			
4	2.09E-05	5.53E-09	2.75E-11	2.29E-11
6	2.92E-05	7.85E-09	3.94E-11	2.26E-11
8	3.20E-05	8.59E-09	4.31E-11	2.39E-11
10	3.28E-05	8.80E-09	4.41E-11	2.53E-11

Table 1: $||\mathbf{A} - \tilde{\mathbf{A}}||_F / ||\mathbf{A}||_F$ for different kernels.

Next we illustrate the efficiency and accuracy of solving (3) and (7) using the approximation $\tilde{\mathbf{A}}$. For kernel functions (i) to (iv), we first choose a random vector $\mathbf{x}$ to generate the right hand side vector $\mathbf{b} = (\mathbf{I} - \mathbf{A})\mathbf{x}$. Then we solve the approximate equation $(\mathbf{I} - \tilde{\mathbf{A}})\tilde{\mathbf{x}} = \mathbf{b}$ for the approximate solution $\tilde{\mathbf{x}}$. For kernel functions (v) and (vi), we again choose a random vector $\mathbf{x}$ to generate the right hand side vector $\mathbf{b} = (\mathbf{I} - \mathbf{DA})\mathbf{x}$, see (7). Then we solve the approximate equation $(\mathbf{I}-\mathbf{D}\tilde{\mathbf{A}})\tilde{\mathbf{x}} = \mathbf{b}$ for the approximate solution $\tilde{\mathbf{x}}$. All equations are solved by the CGLS method (see [3]) which basically solves the normal equation of a given equation by the conjugate gradient method.

In the CGLS method, we choose the zero vector as the initial guess and the stopping criterion is

$$\frac{||\mathbf{r}_q||_2}{||\mathbf{r}_0||_2} < 10^{-10},$$

where $\mathbf{r}_q$ is the residual vector at the qth iteration. The numbers of iterations required for convergence for the six kernels are given in Table 2. To measure the accuracy of the approximate solution $\tilde{\mathbf{x}}$, we have computed the relative error $||\mathbf{x}-\tilde{\mathbf{x}}||_2/||\mathbf{x}||_2$. The results are shown in Table 3.

	$k=4$	$k=8$	$k=11$	$k=14$	$k=4$	$k=8$	$k=11$	$k=14$
l	$a(x,t)=\log\|x-t\|$				$a(x,t)=\cos(xt^2)\log\|x-t\|$			
4	13	13	13	13	13	13	13	13
6	13	13	13	13	13	13	13	13
8	13	13	13	13	13	13	13	13
10	13	13	13	13	13	13	13	13
l	$a(x,t)=\cos(xt^2)\|x-t\|^{-1/2}$				$a(x,t)=\cos(xt^2)\|x-t\|^{1/2}$			
4	19	23	25	26	8	8	8	8
6	26	29	31	36	8	8	8	8
8	33	32	32	33	8	8	8	8
10	32	32	33	34	8	8	8	8
l	$a(x,t)=(1+\frac{1}{2}\sin(100x))\log\|x-t\|$				$a(x,t)=\sin(100x)\log\|x-t\|$			
4	13	14	14	14	12	13	14	14
6	14	13	13	13	14	14	14	14
8	13	13	13	13	14	14	14	14
10	13	13	13	13	14	14	14	14

Table 2: Numbers of iterations required for convergence.

	$k=4$	$k=8$	$k=11$	$k=14$
l	$a(x,t)=\log\lvert x-t\rvert$			
4	3.45E-05	1.27E-08	1.28E-10	9.89E-11
6	4.74E-05	1.87E-08	9.55E-11	7.03E-11
8	5.27E-05	2.06E-08	8.68E-11	5.06E-11
10	5.38E-05	2.11E-08	7.75E-11	3.24E-11
l	$a(x,t)=\cos(xt^2)\log\lvert x-t\rvert$			
4	3.24E-05	1.22E-08	7.27E-11	7.31E-11
6	4.47E-05	1.80E-08	7.84E-11	2.99E-11
8	5.00E-05	1.98E-08	7.87E-11	3.26E-11
10	5.11E-05	2.03E-08	7.36E-11	2.38E-11
l	$a(x,t)=\cos(xt^2)\lvert x-t\rvert^{-1/2}$			
4	7.44E-05	4.16E-08	2.02E-10	3.32E-11
6	1.76E-04	1.54E-07	1.58E-09	1.48E-09
8	1.28E-03	2.03E-07	4.52E-10	7.03E-11
10	3.07E-04	1.44E-07	5.24E-10	5.93E-11
l	$a(x,t)=\cos(xt^2)\lvert x-t\rvert^{1/2}$			
4	6.96E-06	1.29E-09	8.57E-12	3.61E-11
6	1.27E-05	2.03E-09	1.60E-11	2.61E-11
8	1.45E-05	2.27E-09	1.23E-11	2.03E-11
10	1.47E-05	2.33E-09	5.36E-12	1.94E-11
l	$a(x,t)=(1+\frac{1}{2}\sin(100x))\log\lvert x-t\rvert$			
4	3.81E-05	1.31E-08	5.59E-11	2.40E-11
6	5.11E-05	1.84E-08	1.17E-10	1.27E-10
8	5.63E-05	2.04E-08	1.14E-10	4.75E-11
10	5.71E-05	2.06E-08	7.26E-11	6.02E-11
l	$a(x,t)=\sin(100x)\log\lvert x-t\rvert$			
4	3.02E-05	1.28E-08	4.28E-11	3.19E-11
6	6.27E-05	1.92E-08	7.72E-11	9.68E-11
8	7.46E-05	2.11E-08	9.69E-11	1.07E-10
10	7.75E-05	2.19E-08	8.85E-11	5.66E-11

Table 3: $\|\mathbf{x}-\tilde{\mathbf{x}}\|_2/\|\mathbf{x}\|_2$ for different kernels.

Since the kernel functions we tried are at most weakly singular, we see from Table 2 that the convergence rate is linear as expected, see [8, Theorem 2.21].

Recall from §4 that the cost of matrix-vector multiplication $\tilde{\mathbf{A}}\mathbf{y}$ is of $O(n \log n)$ operations, the total cost of solving the systems is thus of $O(n \log n)$ operations too. We emphasize again that in order to get the approximate solution $\tilde{\mathbf{x}}$, we only have to form $\tilde{\mathbf{A}}$ (which requires only $O(nk^2)$ operations) and no need to form $\mathbf{A}$.

We finally compare the operations required in computing the matrix-vector multiplications $\tilde{\mathbf{A}}\mathbf{x}$ and $\mathbf{A}\mathbf{x}$. Tables 4a–4d give the numbers of floating point operations (flops) required. We note that the counts do not depend on the kernel functions used. In the tables, the *ratios* denote the ratios of the operation counts when the size n of the matrix is doubled. We clearly see from the ratios that the cost of the matrix-vector multiplication $\tilde{\mathbf{A}}\mathbf{x}$ is approaching $O(nkl) = O(n \log n)$, whereas that of $\mathbf{A}\mathbf{x}$ is $O(n^2)$.

n	$\tilde{\mathbf{A}}\mathbf{x}$	*ratio*	$\mathbf{A}\mathbf{x}$	*ratio*
32	3,067	—	2,096	—
64	9,682	3.1568	8,288	3.9542
128	25,705	2.6549	32,960	3.9768
256	62,896	2.4468	131,456	3.9883
512	147,127	2.3392	525,056	3.9942
1024	334,846	2.2759	2,098,688	3.9971
2048	748,357	2.2349	8,391,680	3.9985
4096	1,651,084	2.2063	33,560,576	3.9993
8192	3,607,507	2.1849	134,230,016	3.9996
16384	7,821,850	2.1682	536,895,488	3.9998

Table 4a: Flops counts in computing $\tilde{\mathbf{A}}\mathbf{x}$ and $\mathbf{A}\mathbf{x}$ for $k = 4$.

n	$\tilde{\mathbf{A}}\mathbf{x}$	*ratio*	$\mathbf{Ax}$	*ratio*
64	9,767	—	8,272	—
128	29,322	3.0022	32,928	3.9807
256	76,221	2.5994	131,392	3.9903
512	184,224	2.4170	524,928	3.9951
1024	427,459	2.3203	2,098,432	3.9976
2048	967,206	2.2627	8,391,168	3.9988
4096	2,152,073	2.2250	33,559,552	3.9994
8192	4,731,372	2.1985	134,227,968	3.9997
16384	10,307,919	2.1786	536,891,392	3.9998

Table 4b: Flops counts in computing $\tilde{\mathbf{A}}\mathbf{x}$ and $\mathbf{Ax}$ for $k = 8$.

n	$\tilde{\mathbf{A}}\mathbf{x}$	*ratio*	$\mathbf{Ax}$	*ratio*
88	17,438	—	15,592	—
176	51,570	2.9573	62,160	3.9867
352	132,786	2.5749	248,224	3.9933
704	319,158	2.4036	992,064	3.9966
1408	737,794	2.3117	3,966,592	3.9983
2816	1,664,862	2.2565	15,863,040	3.9992
5632	3,696,602	2.2204	63,445,504	3.9996
11264	8,113,302	2.1948	253,768,704	3.9998
22528	17,651,154	2.1756	1,015,048,192	3.9999

Table 4c: Flops counts in computing $\tilde{\mathbf{A}}\mathbf{x}$ and $\mathbf{Ax}$ for $k = 11$.

n	$\tilde{\mathbf{A}}\mathbf{x}$	*ratio*	$\mathbf{Ax}$	*ratio*
112	27,377	—	25,216	—
224	80,262	2.9317	100,608	3.9898
448	205,515	2.5606	401,920	3.9949
896	492,216	2.3950	1,606,656	3.9975
1792	1,134,997	2.3059	6,424,576	3.9987
3584	2,556,306	2.2523	25,694,208	3.9994
7168	5,667,407	2.2170	102,768,640	3.9997
14336	12,423,564	2.1921	411,058,176	3.9998
28672	27,000,777	2.1734	1,644,199,936	3.9999

Table 4d: Flops counts in computing $\tilde{\mathbf{A}}\mathbf{x}$ and $\mathbf{Ax}$ for $k = 14$.

References

[1] B. Alpert, G. Beylkin, R. Coifman and V. Rokhlin, *Wavelets for the Fast Solution of Second-Kind Integral Equations*, SIAM J. Sci. Comput., 14(1993), 159–184.

[2] G. Beylkin, R. Coifman and V. Rokhlin, *Fast Wavelet Transforms and Numerical Algorithms I*, Comm. Pure Appl. Math., 46(1991), 141–183.

[3] A. Björck, *Least Squares Methods*, Handbook of Numerical Methods, P. Ciarlet and J. Lions, ed., V 1, Elsevier, North-Holland, 1989.

[4] L. Delves and J. Mohamed, *Computational Methods for Integral Equations*, Cambridge University Press, Cambridge, 1985.

[5] G. Golub and C. Van Loan, *Matrix Computations*, 2nd ed., John Hopkins University Press, Baltimore, 1989.

[6] L. Greengard and V. Rokhlin, *A Fast Algorithm for Particle Simulations*, J. Comput. Phys., 73(1987), 325–348.

[7] R. Hayes, *Iterative Methods of Solving Linear Problems on Hilbert Space*, Nat. Bur. Standards Appl. Math. Ser., 39(1954), 71–103.

[8] R. Kress, *Linear Integral Equations*, Applied Mathematical Sciences, V 82, Springer-Verlag, New York, 1989.

[9] L. Reichel, *Fast Solution Methods for Fredholm Integral Equations of the Second Kind*, Numer. Math., 57(1989), 719–736.

[10] W. Rudin, *Functional Analysis*, 2nd ed., McGraw-Hill, New York, 1991.

Fast Construction of Optimal Circulant Preconditioners for Matrices from Fast Dense Matrix Method

Abstract

In this paper, we consider solving non-convolution type integral equations by the preconditioned conjugate gradient method. The fast dense matrix method is a fast multiplication scheme that provides a dense discretization matrix A approximating a given integral equation. The dense matrix A can be constructed in $O(n)$ operations and requires only $O(n)$ storage where n is the size of the matrix. Moreover, the matrix-vector multiplication $A\mathbf{x}$ can be done in $O(n \log n)$ operations. Thus if the conjugate gradient method is used to solve the discretized system, the cost per iteration is $O(n \log n)$ operations. However, for some integral equations, such as the Fredholm integral equations of the first kind, the system will be ill-conditioned and therefore the convergence rate of the method will be slow. In these cases, preconditioning is required to speed up the convergence rate of the method. A good choice of preconditioner is the optimal circulant preconditioner which is the minimizer of $\|C - A\|_F$ in Frobenius norm over all circulant matrices C. It can be obtained by taking arithmetic averages of all the entries of A and therefore the cost of constructing the preconditioner is of $O(n^2)$ operations for general dense matrices. In this paper, we develop an $O(n \log n)$ method of constructing the preconditioner for dense matrices A obtained from the fast dense matrix method. Application of these ideas to boundary integral equations from potential theory will be given. These equations are ill-conditioned whereas their optimal circulant preconditioned equations will be well-conditioned. The accuracy of the approximation A, the fast construction of the preconditioner and the fast convergence of the preconditioned systems will be illustrated by numerical examples.

AMS(MOS) subject classifications. 45B05, 65F10, 65R20.

Key Words. Integral equations, circulant preconditioners, conjugate gradient method.

1 Introduction

Circulant matrices are matrices that have constant diagonals and that the first entry of each column is the last entry of its preceding column. More precisely, each column in the matrix is obtained by a cyclic shift of its preceding column. For an n-by-n matrix B, the optimal circulant preconditioner $c(B)$ of B is defined to be the minimizer of $\|C - B\|_F$ over all n-by-n circulant matrices C, see T. Chan [10]. Here $\|\cdot\|_F$ denotes the Frobenius norm. Since $c(B)$ is a circulant matrix, it is determined uniquely by its first column which can be obtained easily by taking the arithmetic average of the entries $b_{\alpha,\beta}$ of B. More precisely, the entries $\{c_\delta\}_{\delta=1}^{n}$ in the first column of $c(B)$ are given by

$$c_{\delta+1} = \frac{1}{n} \sum_{\alpha-\beta=\delta (\text{mod } n)} b_{\alpha,\beta}, \quad \delta = 0, \ldots, n-1, \tag{1}$$

see Tyrtyshnikov [15].

Using the circulant structure of $c(B)$, the inverse $[c(B)]^{-1}$ of $c(B)$ and the matrix-vector multiplication $[c(B)]^{-1}\mathbf{x}$ for any vector $\mathbf{x}$ can be obtained in $O(n \log n)$ operations by using fast Fourier transforms, see for instance Chan and Ng [8]. Moreover, $c(B)$ is positive-definite whenever B is, see Tyrtyshnikov [15]. This makes $c(B)$ a very attractive choice of preconditioner in the preconditioned conjugate gradient method for solving the system $B\mathbf{y} = \mathbf{b}$. For then, if B is positive-definite, the preconditioned matrix is positive-definite. Moreover, the cost of multiplying $[c(B)]^{-1}$ to a vector, which is required in each iteration of the method, can be obtained in $O(n \log n)$ operations by using fast Fourier transforms.

However, from (1), we see that the cost of constructing $c(B)$ is of $O(n^2)$ operations for general matrices B. This cost count can be reduced to $O(n)$ when B is a band matrix or a Toeplitz matrix (i.e. matrix with constant diagonals). Therefore, the optimal circulant preconditioner has been used in the numerical solutions of partial differential equations and in Toeplitz least squares problems from signal and image processing. Convergence results for the preconditioned systems arising from these problems have been established, see for instance Chan and Ng [8] and the references therein.

Optimal circulant preconditioners have also been proposed and used successfully in solving convolution type integral equations, see [12, 4]. The discrete matrices from these integral equations are Toeplitz matrices if the rectangular quadrature rule is used. For non-convolution type integral equations, where the discrete matrices are no longer Toeplitz, convergence analysis of optimal circulant preconditioners has also been studied, see [6, 9]. For example, for boundary integral equations arising from potential equations, which are ill-conditioned, non-convolution type integral equations with condition number increasing like $O(n)$, the preconditioned systems have been shown to be well-conditioned, see Chan, Sun and Ng [9] and also §5.

However, there are two main difficulties in using circulant preconditioned conjugate gradient methods for non-convolution type integral equations. The first one is that the discretization matrix B corresponding to the integral equation is dense. Hence multiplying B to a vector, which is required in each iteration of the conjugate gradient method, is of $O(n^2)$ complexity. The other difficulty is that since B is dense, forming the optimal circulant preconditioner $c(B)$ using (1) will require $O(n^2)$ operations. In this paper, we will address the second difficulty.

To overcome the first difficulty, a number of fast multiplication schemes have been developed in recent years, see for instance [13, 14, 3, 1]. These methods try to obtain an approximation A to the given integral equation such that the matrix-vector multiplication of A with any vector can be done in $O(n)$ or $O(n \log n)$ operations, depending on the smoothness of the kernel function of the integral equation. In Chan, Lin and Ng [7], we have proposed the *fast dense matrix method* for approximating integral equations. Our approximation matrix A is a dense matrix which can be obtained in $O(n)$ operations and only $O(n)$ storage is required. Moreover, the matrix-vector multiplication $A\mathbf{x}$ can be done in $O(n \log n)$ operations.

To deal with the second difficulty mentioned above, we will develop in this paper a fast algorithm for constructing the optimal circulant preconditioner $c(A)$ for matrices A that are obtained from our fast dense matrix method. Using the special structure of our A, the circulant matrix $c(A)$ can be obtained in $O(n \log n)$ operations. Thus, the construction of the discretization matrix A and its circulant preconditioner $c(A)$, and the cost of multiplying A or $[c(A)]^{-1}$ to any vector can all be done in $O(n \log n)$ operations. Hence used in conjunction with our fast dense matrix method, the circulant preconditioned conjugate gradient method for integral equations requires only $O(n \log n)$ operations per iteration.

In order to illustrate the efficiency of our construction and the effect of using circulant preconditioners, we will apply these ideas to solving first kind integral equations from potential equations. These equations are known to be ill-conditioned with condition number growing like $O(n)$. Chan, Sun and Ng [9] have shown however that the problem becomes well-conditioned if it is preconditioned by optimal circulant preconditioners. In particular, the preconditioned system converges in a fixed finite number of iterations independent of the size of the discretized system. Thus, if the system is solved by using preconditioned conjugate gradient method coupled with our fast dense matrix method, the total cost of solving the system is of the order $O(n \log n)$ operations.

Before we go on, let us define some terminologies to be used later on. Given an n-by-n matrix B with entries $b_{\alpha,\beta}$, we define the *diagonal sums* $\{d_\delta\}_{\delta=-(n-1)}^{n-1}$ of B to be the sum along each of the diagonals of B. More precisely,

$$d_\delta \equiv \begin{cases} \displaystyle\sum_{\alpha=\delta+1}^{n} b_{\alpha,\alpha-\delta}, & 0 \le \delta < n, \\ \displaystyle\sum_{\alpha=1}^{n+\delta} b_{\alpha,\alpha-\delta}, & 0 \le -\delta < n. \end{cases} \tag{2}$$

Thus d_0 is the sum of the main diagonal entries of B, d_1 is the sum of the entries on the sub-diagonal and d_{-1} is the sum of the entries on the super-diagonal. We note that once we have the diagonal sums of B, then $c(B)$ can be obtained in $O(n)$ operations. In fact, by comparing (1) and (2), the following lemma follows.

Lemma 1 *Let $\{d_\delta\}_{\delta=-(n-1)}^{n-1}$ be the diagonal sums of B. Then the first column entries $\{c_\delta\}_{\delta=1}^{n}$ of $c(B)$ are given by $c_1 = d_0$ and*

$$c_{\delta+1} = \frac{1}{n}(d_\delta + d_{\delta-n}), \quad \delta = 1, \ldots, n-1.$$

The outline of the paper is as follows. In §2, we briefly recall the main concept of the fast dense matrix method. Preliminary lemmas that are useful in computing diagonal sums of matrices are given in §3. In §4, we give an $O(n \log n)$ algorithm for constructing the optimal circulant preconditioners. In §5, we apply our ideas to solving boundary integral equations from potential equations where the use of circulant preconditioners will speed up the convergence. Finally, concluding remarks are given in §6.

2 Fast Dense Matrix Method

In this section, we give a brief introduction to the fast dense matrix method. We refer the reader to Chan, Lin and Ng [7] for details. In the following, n is the size of the discretization matrix under consideration, i.e. $1/n$ is proportional to the the mesh size with which we discretize the integral equation. We will set $n = k \cdot 2^l$, where k is a fixed small integer that depends on the smoothness of the kernel function of the given integral equation.

The fast dense matrix method proposed in [7] approximates a given integral equation or its discretization matrix by sum of low rank matrices using the partition suggested in Alpert *et. al.* [1]. With such partition, the approximation matrix A is divided into blocks of different sizes, with the blocks near the main diagonal are of size k-by-k, those next remote are of size $2k$-by-$2k$, and so forth up to the largest blocks of size $2^{l-2}k$-by-$2^{l-2}k$. Then, each of the blocks is approximated by a rank k matrix by using polynomial approximation. By grouping blocks of the same size into one matrix, the approximation matrix A is given by

$$A = A^{(0)} + A^{(1)} + \cdots + A^{(l-2)}, \tag{3}$$

with each $A^{(\mu)}$, $\mu = 0, \ldots, l-2$, consists only of blocks of size $2^{\mu}k$-by-$2^{\mu}k$. The number of nonzero blocks in $A^{(\mu)}$ is given by

$$\nu_{\mu} = \begin{cases} 6 \cdot 2^{l} - 8 & \mu = 0, \\ 6(2^{l-1-\mu} - 1) & \mu = 1, \ldots, l-2. \end{cases} \tag{4}$$

We will denote those nonzero blocks in $A^{(\mu)}$ by $A^{(\mu,\nu)}$, where $\nu = 1, \ldots, \nu_{\mu}$. As an illustration, for $l = 5$, $A^{(2)}$ is of the form

$$A^{(2)} = \left[\begin{array}{cccccccc}
 & & \boxed{A^{(2,10)}} & \boxed{A^{(2,16)}} & & & & \\
 & & & \boxed{A^{(2,11)}} & & & & \\
\boxed{A^{(2,4)}} & & & & \boxed{A^{(2,12)}} & \boxed{A^{(2,17)}} & & \\
\boxed{A^{(2,1)}} & \boxed{A^{(2,5)}} & & & & \boxed{A^{(2,13)}} & & \\
 & & \boxed{A^{(2,6)}} & & & & \boxed{A^{(2,14)}} & \boxed{A^{(2,18)}} \\
 & & \boxed{A^{(2,2)}} & \boxed{A^{(2,7)}} & & & & \boxed{A^{(2,15)}} \\
 & & & & \boxed{A^{(2,8)}} & & & \\
 & & & & \boxed{A^{(2,3)}} & \boxed{A^{(2,9)}} & &
\end{array}\right]. \tag{5}$$

Here each of the block $A^{(\mu,\nu)}$ is of size $2^{\mu}k$-by-$2^{\mu}k$ and is a rank k matrix of the form

$$A^{(\mu,\nu)} = (P^{(\mu)})^T \Lambda^{(\mu,\nu)} P^{(\mu)}, \tag{6}$$

where $\Lambda^{(\mu,\nu)}$ is a k-by-k matrix and $P^{(\mu)}$ is a k-by-$2^{\mu}k$ matrix which is the same for all $\nu = 1, \ldots, \nu_{\mu}$. For $\mu = 0$, $P^{(0)}$ is just the k-by-k identity matrix. Using (6) and the block structure of $A^{(\mu)}$ as depicted in (5), we see that $A^{(\mu)}$ can be written as

$$A^{(\mu)} = \left[I_{2^{l-\mu}} \otimes (P^{(u)})^T\right] \Lambda^{(\mu)} \left[I_{2^{l-\mu}} \otimes P^{(u)}\right]. \tag{7}$$

Here $I_{2^{l-\mu}}$ is the identity matrix of size $2^{l-\mu}$, $\otimes$ is the Kronecker tensor product and $\Lambda^{(\mu)}$ is a matrix having the same block structure as $A^{(\mu)}$, except that the blocks $\Lambda^{(\mu,\nu)}$ in $\Lambda^{(\mu)}$ are of size k. As an illustration, the matrix $A^{(2)}$ for $l = 5$ can be written as (cf. (5)):

$$A^{(2)} = \left[I_8 \otimes \left(P^{(2)}\right)^T\right] \begin{bmatrix} & & \Lambda^{(2,10)} & \Lambda^{(2,16)} & & & & \\ & & & \Lambda^{(2,11)} & & & & \\ \Lambda^{(2,4)} & & & & \Lambda^{(2,12)} & \Lambda^{(2,17)} & & \\ \Lambda^{(2,1)} & \Lambda^{(2,5)} & & & & \Lambda^{(2,13)} & & \\ & & \Lambda^{(2,6)} & & & & \Lambda^{(2,14)} & \Lambda^{(2,18)} \\ & & \Lambda^{(2,2)} & \Lambda^{(2,7)} & & & & \Lambda^{(2,15)} \\ & & & & \Lambda^{(2,8)} & & & \\ & & & & \Lambda^{(2,3)} & \Lambda^{(2,9)} & & \end{bmatrix} \left[I_8 \otimes P^{(2)}\right], \tag{8}$$

where each $\Lambda^{(2,v)}$ is a k-by-k matrix.

Combining (3) and (7), we then have

$$A = \sum_{\mu=0}^{l-2} A^{(\mu)} = \sum_{\mu=0}^{l-2} \left[I_{2^{l-\mu}} \otimes (P^{(\mu)})^T\right] \Lambda^{(\mu)} \left[I_{2^{l-\mu}} \otimes P^{(\mu)}\right]. \tag{9}$$

Thus for the computation of the matrix-vector product $A\mathbf{x}$ using (9), it suffices to form and store $\Lambda^{(\mu)}$ and $P^{(\mu)}$ for $\mu = 0, \ldots, l-2$ only. As shown in Chan, Lin and Ng [7], these matrices can be constructed in $O(n)$ operations and requires $O(n)$ storage, and the cost of the matrix-vector multiplication $A\mathbf{x}$ using (9) is of order $O(n \log n)$ operations.

In §4, we will discuss an $O(n \log n)$ algorithm for forming the optimal circulant preconditioner $c(A)$ of A using the decomposition in (9). Recall that by Lemma 1, $c(A)$ is determined once we have the diagonal sums of A. Hence we need to know how to form the diagonal sums of $A^{(\mu,\nu)}$ in (6), for they are the fundamental building blocks of $A^{(\mu)}$ and hence of A. This will be studied in the next section.

3 Preliminary Lemmas

In this section, we consider the cost and storage requirement for forming the diagonal sums of matrices of the form given in (6). This result is required in §4 when we construct the optimal circulant preconditioner $c(A)$ for A. We begin with the complexity counts of forming diagonal sums for rank 1 matrices.

Lemma 2 *Let $\mathbf{p} = (p_1, \ldots, p_m)$ and $\mathbf{q} = (q_1, \ldots, q_m)$. Then the diagonal sums of the m-by-m matrix $\mathbf{p}^t\mathbf{q}$ can be obtained in $O(m \log m)$ operations and the storage required is $O(m)$.*

Proof: Recall from (2) that the diagonal sums of $\mathbf{p}^t\mathbf{q}$ are given by

$$d_\delta = \begin{cases} \sum\limits_{\alpha=\delta+1}^{m} p_\alpha q_{\alpha-\delta}, & 0 \le \delta < m, \\ \sum\limits_{\alpha=1}^{m+\delta} p_\alpha q_{\alpha-\delta}, & 0 \le -\delta < m. \end{cases}$$

In matrix terms, this amounts to

$$\begin{pmatrix} p_m & & & & 0 \\ p_{m-1} & p_m & & & \\ \vdots & \ddots & \ddots & & \\ p_2 & \vdots & \ddots & \ddots & \\ p_1 & p_2 & \vdots & \ddots & p_m \\ & p_1 & \ddots & \vdots & p_{m-1} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & p_2 \\ 0 & & & & p_1 \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ \vdots \\ q_m \end{pmatrix} = \begin{pmatrix} d_{m-1} \\ d_{m-2} \\ \vdots \\ d_1 \\ d_0 \\ d_{-1} \\ \vdots \\ d_{-(m-2)} \\ d_{-(m-1)} \end{pmatrix}, \qquad (10)$$

where the matrix is a column circulant matrix.

It is easy to augment the column circulant matrix to make it a square circulant matrix (which in fact is determined uniquely by its first column). The diagonal sums $\{d_j\}_{j=-m+1}^{m-1}$, i.e. the right hand side vector in (10), can be obtained by multiplying the augmented square circulant matrix to the augmented vector $(q_1, q_2, \ldots, q_m, 0, \ldots, 0)^t$. This matrix-vector product can be obtained efficiently by using three fast Fourier transforms of length $2m-1$, see for instance Chan and Ng [8]. Thus the cost of obtaining the diagonal sums is $O(m \log m)$ operations and the storage required is $O(m)$. $\square$

Corollary 1 *Let P and Q be two given k-by-m matrices. Then the diagonal sums of the product P^tQ can be obtained in $O(km \log m)$ operations and the storage required is $O(m)$.*

Proof: We have

$$P^tQ = \sum_{\alpha=1}^{k} \mathbf{p}_\alpha^t \mathbf{q}_\alpha \qquad (11)$$

where $\mathbf{p}_\alpha$ and $\mathbf{q}_\alpha$ are the αth row of P and Q respectively. Notice that the sum of the diagonal sums is equal to the diagonal sums of the sum. Therefore we can form the diagonal sums of each term $\mathbf{p}_\alpha^t\mathbf{q}_\alpha$ and sum them up to get the diagonal sums of P^tQ. By Lemma 2, the diagonal sums for each $\mathbf{p}_\alpha^t\mathbf{q}_\alpha$, $\alpha = 1, \ldots, k$, can be obtained in $O(m \log m)$ operations with $O(m)$ storage. Once the diagonal sums of $\mathbf{p}_\alpha^t\mathbf{q}_\alpha$ are formed, they can be accumulated to the final result and there is no need to store the intermediate diagonal sums for each $\alpha = 1, \ldots, k$. Thus the cost of obtaining the diagonal sums of P^tQ is $O(km \log m)$ operations and storage requirement is $O(m)$. $\square$

Corollary 2 *Let P be a given k-by-m matrix and Λ be a k-by-k matrix. Then the diagonal sums of the product $P^t\Lambda P$ can be obtained in $O(km \log m) + O(k^2m)$ operations and the storage required is $O(m)$.*

Proof: We just need to compute the product ΛP first and then apply Corollary 1 to the product $P^t(\Lambda P)$. In computing ΛP, we need one row of the product at any one time (see (11)) and therefore the total cost of forming ΛP is k^2m operations and the total storage required is $O(m)$. $\square$

The fast dense matrix method introduced in §2 provides a good approximation A to a given integral equation or its discretization matrix B, see Chan, Lin and Ng [7] or §5. In the next section, we will construct the optimal circulant preconditioner $c(A)$ for A. We note that since the operator norm of the operator $c(\cdot)$ in matrix 2-norm is equal to 1 (see Chan, Jin and Yeung [5, Theorem 3]), we have,

$$\|c(A) - c(B)\|_2 \leq \|A - B\|_2. \tag{12}$$

Thus if A is a good approximation to B, we expect $c(A)$ to be a better approximation to $c(B)$.

4 Construction of Optimal Circulant Preconditioner

In this section, we develop an $O(n \log n)$ method of constructing the optimal circulant preconditioner $c(A)$ for the approximation A given in (9). By (1), it is clear that $c(\cdot)$ is a linear operator. Therefore by (3), we see that

$$c(A) = c(\sum_{\mu=0}^{l-2} A^{(\mu)}) = \sum_{\mu=0}^{l-2} c(A^{(\mu)}),$$

where we recall that $n = k \cdot 2^l$ with k fixed independent of n. By Lemma 1, $c(A^{(\mu)})$ can be obtained easily if we have the diagonal sums of $A^{(\mu)}$. In view of the block structure of $A^{(\mu)}$ (cf. (5)), we can have the diagonal sums of $A^{(\mu)}$ if we have the diagonal sums of its sub-blocks $A^{(\mu,\nu)}$. Thus in the following, we first consider the complexity of computing the diagonal sums of the sub-blocks $A^{(\mu,\nu)}$. Then the results will be pieced together to get the complexity counts for computing the diagonals sums of A.

We begin by noting that for $\mu = 1, \ldots, l-2$, $A^{(\mu)}$ is a block matrix made up of sub-blocks $A^{(\mu,\nu)}$ that concentrate only on four block-diagonals (cf (5)). Since the sum of diagonal sums is equal to the diagonal sums of the sum, one can obtain the diagonal sums of $A^{(\mu)}$ by summing the sub-blocks $A^{(\mu,\nu)}$ along the four block-diagonals first and computing the diagonal sums afterward. To be more specific, let us consider the example in (5) first. Here $\mu = 2$. The diagonal sums of $A^{(2)}$ can be obtained from the diagonal sums of the following four matrices:

$$\sum_{\nu=1}^{3} A^{(2,\nu)}, \quad \sum_{\nu=4}^{9} A^{(2,\nu)}, \quad \sum_{\nu=10}^{15} A^{(2,\nu)} \quad \text{and} \quad \sum_{\nu=16}^{18} A^{(2,\nu)}.$$

By (6), these four matrices can be rewritten as

$$\begin{aligned}
\sum_{\nu=1}^{3} A^{(2,\nu)} &= (P^{(2)})^t\{\sum_{\nu=1}^{3} \Lambda^{(2,\nu)}\}P^{(2)} \equiv (P^{(2)})^t\Lambda_1^{(2)}P^{(2)}, \\
\sum_{\nu=4}^{9} A^{(2,\nu)} &= (P^{(2)})^t\{\sum_{\nu=4}^{9} \Lambda^{(2,\nu)}\}P^{(2)} \equiv (P^{(2)})^t\Lambda_2^{(2)}P^{(2)}, \\
\sum_{\nu=10}^{15} A^{(2,\nu)} &= (P^{(2)})^t\{\sum_{\nu=10}^{15} \Lambda^{(2,\nu)}\}P^{(2)} \equiv (P^{(2)})^t\Lambda_3^{(2)}P^{(2)}, \\
\sum_{\nu=16}^{18} A^{(2,\nu)} &= (P^{(2)})^t\{\sum_{\nu=16}^{18} \Lambda^{(2,\nu)}\}P^{(2)} \equiv (P^{(2)})^t\Lambda_4^{(2)}P^{(2)}. \qquad (13)
\end{aligned}$$

From the diagonal sums of these four matrices, one can compute the diagonal sums of $A^{(2)}$, cf. (5). With this example in mind, it is easy to verify the following lemma.

Lemma 3 *For $\mu = 1, 2, \ldots, l-2$, the diagonal sums of $A^{(\mu)}$ can be obtained in $O(k^2\mu 2^{\mu}) + O(k^2 2^{l-\mu}) + O(k^3 2^{\mu})$ operations and $O(k2^{\mu})$ storage.*

Proof: As in the example above, we first have to sum the $\Lambda^{(\mu,\nu)}$ along the four block-diagonals to obtain $\Lambda_1^{(\mu)}$, $\Lambda_2^{(\mu)}$, $\Lambda_3^{(\mu)}$ and $\Lambda_4^{(\mu)}$ (cf (13) and (8)). By (4), there are $6(2^{l-1-\mu}-1)$ sub-blocks of $\Lambda^{(\mu,\nu)}$, which are all k-by-k matrices. Therefore to form $\Lambda_\alpha^{(\mu)}$, $\alpha = 1, 2, 3, 4$, it requires at most $6(2^{l-1-\mu}-1)k^2$ operations and $4k^2$ memory.

Once $\Lambda_\alpha^{(\mu)}$ for $\alpha = 1, 2, 3, 4$ are formed, we compute the diagonal sums of the matrices

$$(P^{(\mu)})^t\Lambda_\alpha^{(\mu)}P^{(\mu)}, \quad \alpha = 1, 2, 3, 4,$$

(cf. (13)). We recall by (6) that $P^{(\mu)}$ are k-by-$2^\mu k$ matrices. Therefore, by Corollary 2, the diagonal sums of these four matrices can be obtained in $O(k^2 2^\mu(\mu + \log k) + k^3 2^\mu)$ operations and $O(k2^\mu)$ storage.

Once these diagonal sums are formed, we can accumulate them together to get the diagonal sums of $A^{(\mu)}$. This step requires no more than $2^{\mu+3}k$ operations since there are only four diagonal sums to accumulate and each of the diagonal sums has no more than $2^{\mu+1}k$ numbers. Combining all the complexity counts above, the lemma follows. $\square$

Next we consider the case for $\mu = 0$.

Lemma 4 *The diagonal sums of $A^{(0)}$ can be obtained in $O(k^2 2^l)$ operations and $O(k)$ storage.*

Proof: For $\mu = 0$, the sub-blocks $A^{(0,\nu)}$ are of size k-by-k and are concentrated on 7 (instead of 4) block-diagonals next to and including the main block-diagonal, see for instance [1, Figure 4]. In other words, $A^{(0)}$ is a band matrix of band-width less than or equal to $8k$. Thus forming the diagonal sums of $A^{(0)}$ requires at most $O(kn) = O(k^2 2^l)$ operations and $8k$ memory. $\square$

Combining Lemmas 2, 3 and 4, we have our main theorem.

Theorem 1 *For A given in (9), the cost of forming $c(A)$ for A is of $O(kn\log n)+ O(k^2 n)$ operations and the storage required is $O(n)$.*

Proof: In view of Lemma 3 and 4, the cost of obtaining the diagonal sums of $\sum_{\mu=0}^{l-2} A^{(\mu)}$ is of the *order* of

$$k^2 2^l + k^2 \sum_{\mu=1}^{l-2} (\mu 2^\mu + 2^{l-\mu} + k 2^\mu) \le k^2 l 2^l + k^3 2^l = kn\log n + k^2 n.$$

The memory requirement is of the *order* of

$$8k + k\sum_{\mu=1}^{l-2} 2^{\mu} = k2^{l-1} + 6k = \frac{n}{2} + 6k.$$

Once the diagonal sums of A are formed, by using Lemma 1, the first column of the circulant matrix $c(A)$ can be obtained in just another $O(n)$ operations.

□

In the next section, we will apply our $c(A)$ to solving systems $A\mathbf{x} = \mathbf{b}$ arising from non-convolution type integral equations.

5 Boundary Integral Equations From Potential Equations

In this section, we consider solutions of potential equations

$$\begin{cases} \Delta w(x) = 0, & x \in \Omega, \\ w(x) = g(x), & x \in \partial\Omega, \end{cases}$$

where $\partial\Omega$ is a smooth close curve in $\mathbb{R}^2$ and Ω is either the bounded interior region with boundary $\partial\Omega$ or the unbounded exterior region with boundary $\partial\Omega$. In the boundary integral equation approach, the solution $w(x)$ is found by solving the density function $u(y)$ in the following Fredholm equation of the first kind:

$$-\frac{1}{2\pi}\int_{\partial\Omega} \log|x-y|u(y)dS_y = g(x), \qquad x \in \partial\Omega, \tag{14}$$

see Chen and Zhou [11, §6.12] or Chan, Sun and Ng [9].

If we define the boundary integral operator $\mathcal{B}$ as

$$(\mathcal{B}u)(x) \equiv -\frac{1}{2\pi}\int_{\partial\Omega} \log|x-y|u(y)dS_y,$$

then (14) can be written as

$$(\mathcal{B}u)(x) = g(x). \tag{15}$$

For simplicity, we parameterize the boundary $\partial\Omega$ by $(x_1(\theta), x_2(\theta))$, $0 \le \theta \le 2\pi$, and thus (15) can be expressed as

$$(\mathcal{B}u)(\theta) = \int_0^{2\pi} b(\theta,\phi)u(\phi)d\phi = g(\theta), \quad 0 \le \theta \le 2\pi, \tag{16}$$

where the kernel function $b(\theta, \phi)$ is given by

$$b(\theta,\phi) = -\frac{1}{4\pi}\log\left\{(x_1(\theta)-x_1(\phi))^2 + (x_2(\theta)-x_2(\phi))^2\right\}. \tag{17}$$

In order to guarantee that the operator $\mathcal{B}$ is invertible, we assume without loss of generality that

$$\text{diam}(\partial\Omega) = \max_{x,y\in\partial\Omega} |x-y| < 1. \tag{18}$$

One can scale down the size of the given boundary if necessary, see Chan and Zhou [11, p.287].

The well-known advantage of the boundary integral equation approach is that the dimension of the problem is reduced by one. However, (14) is a first kind boundary integral equation having a weakly singular kernel. It is well-known that its discrete matrix B of $\mathcal{B}$ will be ill-conditioned and have condition number increases like $O(n)$, where n is the size of the matrix, see for instance Chan, Sun and Ng [9]. Therefore if the system is solved by the conjugate gradient method, the number of iterations required for convergence will be increasing like $O(\sqrt{n})$.

To overcome the ill-conditioned nature of the operator $\mathcal{B}$, optimal circulant integral operators are proposed in Chan, Sun and Ng [9] to precondition (16).

Circulant integral operators are convolution operators with 2π-periodic kernels. The *optimal circulant integral operator* of a given operator $\mathcal{B}$ is defined to be the minimizer of $|||\mathcal{C}-\mathcal{B}|||$ over all circulant integral operators $\mathcal{C}$, where $|||\cdot|||$ is the Hilbert-Schmidt norm, see Gohberg, Hanke and Koltracht [12]. For $\mathcal{B}$ given in (16), the kernel function of its optimal circulant integral preconditioner $\mathcal{M}$ is given by

$$\frac{1}{2\pi}\int_0^{2\pi} b(\theta,\theta-\phi)d\theta, \qquad 0<\phi<2\pi,$$

see Chan, Sun and Ng [9]. Instead of solving (15), we solve the preconditioned equation

$$\mathcal{M}^{-1}\mathcal{B}u=\mathcal{M}^{-1}g. \tag{19}$$

It is proven in [9] that this preconditioned equation is well-conditioned.

Theorem 2 (Chan, Sun and Ng [9, Theorems 3,4]) *Let $\mathcal{B}$ be the integral operator as defined in (16) and (17) and $\mathcal{M}$ be the optimal circulant integral operator for $\mathcal{B}$. Then there exist positive constants $\gamma_2 \ge \gamma_1 > 0$ such that the spectrum of $\mathcal{M}^{-1}\mathcal{B}$ lies in $[\gamma_1,\gamma_2]$. Moreover, if the Galerkin method is used to discretize the operator $\mathcal{M}^{-1}\mathcal{B}$, then the condition number of the discretized system is of $O(1)$ independent of the size of the discretized system.*

Thus if the conjugate gradient method is used to solve the preconditioned system (19), the convergence rate of the method is expected to be linear, see Axelsson and Barker [2, p.26].

In the following, we denote B the discretization matrix of $\mathcal{B}$ using the Galerkin method with the trapezoidal rule and A the approximation matrix to $\mathcal{B}$ using our fast dense matrix method. We note that the optimal circulant preconditioner $c(B)$ of B is equal to the discretization matrix of $\mathcal{M}$ using the rectangular quadrature rule, see Chan, Sun and Ng [9, Theorem 5]. Since A approximates B, we expect from (12) that $c(A)$ is a good approximation to $c(B)$ and hence to $\mathcal{M}$.

We now illustrate the effectiveness of the optimal circulant preconditioners and our approximation scheme by using a problem tested in Chan, Sun and Ng [9]. We refer the readers to [9] for more details. We consider the solution of (16) on regions Ω with boundaries $\partial\Omega$ as depicted in Figure 1. The boundaries are defined in polar coordinates by

$$r = \cos 2\theta + f_\lambda(\theta), \qquad 0 \le \theta \le 2\pi, \tag{20}$$

where $f_\lambda(\theta) = (\lambda^4 - \sin^2 2\theta)^{1/2}$ with $\lambda > 1$. Since $\delta \equiv \operatorname{diam}(\partial\Omega) > 1$, we scale the boundary so that the diameter of the new boundary satisfies $\rho = \operatorname{diam}(\partial\Omega) = 3/4 < 1$, see (18). For such scaled domains, the kernel function (17) becomes

$$\begin{aligned} b(\theta,\phi) &= -\frac{1}{2\pi}\log\frac{\rho}{\delta}|2\sin\frac{\theta-\phi}{2}| \\ &\quad -\frac{1}{4\pi}\log\left\{4\sin^2(\theta+\phi)\cos^2(\frac{\theta-\phi}{2})\left(1+\frac{\cos 2\theta+\cos 2\phi}{f_\lambda(\theta)+f_\lambda(\phi)}\right)^2\right. \\ &\qquad\qquad \left. + (\cos 2\theta + f_\lambda(\theta))(\cos 2\phi + f_\lambda(\phi))\right\} \\ &\equiv b_1(\theta,\phi) + b_2(\theta,\phi). \end{aligned} \tag{21}$$

The right hand side $g(\theta)$ in (16) is chosen to be $g(\theta) = |\cos(\theta)|^{\frac{3}{2}}$, $0 \le \theta \le 2\pi$. All our computations were done in Matlab on an IBM 43P-133 workstation.

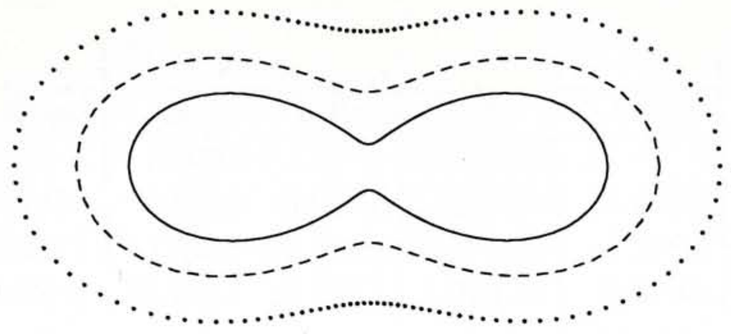

Figure 1. Solid line: $\lambda = 1.1$, dashed line: $\lambda = 1.3$, dotted line: $\lambda = 1.5$

As in Chan, Sun and Ng [9], we discretize $[0, 2\pi]$ by uniform mesh and use the Galerkin method with piecewise constant polynomials as basis functions to discretize the equation. The integral over each element is computed by using trapezoidal rule with 3 points. Since b_1 in (21) is a 2π-periodic convolution kernel, we see that the discretization matrix B of $\mathcal{B}$ can be written as $B = C + B_2$ where C is a circulant matrix corresponding to the integration of b_1 over the elements. Thus C is determined only by its first column. From (21), we also see that

$$b_2(\theta, \phi) = b_2(\phi, \theta) = b_2(2\pi - \theta, 2\pi - \phi), \quad 0 \leq \theta, \phi \leq 2\pi. \tag{22}$$

Therefore, B_2 is a symmetric centro-symmetric matrix. In particular, if B_2 is an n-by-n matrix, it is determined by its upper half entries $[B_2]_{j,l}$, $1 \leq j \leq \lceil n/2 \rceil$, $1 \leq l \leq n$. The bottom half can be obtained by reflecting the upper half entries with respect to the center of the matrix.

It is clear that forming the matrix B_2 (or just its upper half) directly by integration of b_2 over the elements requires $O(n^2)$ operations. Table 1 gives the numbers of floating point operations in thousand (Kflops) required to form the upper half of B_2. We recall that $n = k \cdot 2^l$. Thus the largest matrix size

	k=4		k=8		k=11		k=14	
l	B_2	*ratio*	B_2	*ratio*	B_2	*ratio*	B_2	*ratio*
5	3,780	—	15,095	—	28,528	—	46,200	—
6	15,095	3.9938	60,336	3.9971	114,051	3.9979	184,723	3.9984
7	60,336	3.9971	241,258	3.9986	456,084	3.9990	738,740	3.9992
8	241,258	3.9986	964,861	3.9993	1,824,101	3.9995	2,954,661	3.9996

Table 1: Kflops counts in constructing B_2, where $n = k \cdot 2^l$.

we tried is $n = 14 \cdot 2^8 = 3,584$. We remark that the counts do not depend on the values of λ and $\text{diam}(\partial\Omega)$. In the table, the *ratios* denote the ratios of the operation counts when the size n of the matrix is doubled. We clearly see from the ratios that the cost of constructing B_2 is increasing like $O(n^2)$.

Besides B_2, we also use our fast dense matrix method to approximate the integration of b_2 over the elements. This results in a matrix A_2 which can be obtained in $O(n)$ operations and requires only $O(n)$ storage, and that the matrix-vector product $A_2\mathbf{x}$ for any vector $\mathbf{x}$ can be done in $O(n \log n)$ operations, see Chan, Lin and Ng [7]. By the centro-symmetric property of b_2 (see (22)), we only need to generate the upper half of A_2. More precisely, we only need to apply our method to get the upper left and upper right $n/2$-by-$n/2$ submatrices of A_2 only. The bottom half of A_2 can be obtained by reflecting these two matrices.

Table 2 gives the numbers of Kflops required to form the upper half of A_2. Since $n = k \cdot 2^l$, the largest matrix size we tried is $n = 14 \cdot 2^{12} = 57,344$. We remark that the counts do not depend on the values of λ and $\text{diam}(\partial\Omega)$. In the table, the *ratios* again denote the ratios of the operation counts when

	k=4		k=8		k=11		k=14	
l	A_2	*ratio*	A_2	*ratio*	A_2	*ratio*	A_2	*ratio*
5	1,701	—	6,768	—	12,827	—	20,857	—
6	3,806	2.2373	15,160	2.2400	28,763	2.2424	46,823	2.2449
7	8,109	2.1306	32,326	2.1323	61,372	2.1337	99,972	2.1351
8	16,808	2.0728	67,039	2.0738	127,326	2.0747	207,490	2.0755
9	34,300	2.0407	136,845	2.0413	259,969	2.0418	423,745	2.0422
10	69,376	2.0227	276,839	2.0230	525,993	2.0233	857,473	2.0236
11	139,623	2.0126	557,207	2.0128	1,058,775	2.0129	1,726,149	2.0131
12	280,210	2.0069	1,118,325	2.0070	2,125,075	2.0071	3,464,720	2.0072

Table 2: Kflops counts in constructing A_2, where $n = k \cdot 2^l$.

the size n of the matrix is doubled. We see from the ratios that the cost of constructing A_2 is increasing like $O(n)$. In contrast, the cost for constructing B_2 is $O(n^2)$, see Table 1. We emphasize that there is no need to form B_2 in order to form A_2. We get A_2 by directly approximating b_2 in (21) using our fast dense matrix method.

To illustrate the accuracy of our approximation, the relative errors $\|A_2 - B_2\|_F/\|B_2\|_F$ for different λ are given in Table 3. Because generating B_2 is very expensive, we tried only matrices of size up to $14 \cdot 2^8 = 3,584$. We see from the table that our approximation scheme provides a very accurate approximation A_2 to the matrix B_2 even for small k like 8.

To accelerate the convergence of the conjugate gradient method, we use the optimal circulant preconditioner $c(C+A_2)$ to precondition the system $(C+A_2)$. By the linear property of $c(\cdot)$, we see that

$$c(C + A_2) = c(C) + c(A_2) = C + c(A_2).$$

k	l	$\lambda = 1.1$	$\lambda = 1.3$	$\lambda = 1.5$	k	l	$\lambda = 1.1$	$\lambda = 1.3$	$\lambda = 1.5$
4	5	3.89E-03	1.20E-03	4.37E-04	11	5	3.40E-06	9.44E-08	6.61E-09
4	6	4.58E-03	1.39E-03	5.03E-04	11	6	3.98E-06	1.09E-07	7.61E-09
4	7	4.94E-03	1.48E-03	5.38E-04	11	7	4.27E-06	1.16E-07	8.12E-09
4	8	5.13E-03	1.53E-03	5.57E-04	11	8	4.42E-06	1.19E-07	8.38E-09
8	5	5.16E-05	3.21E-06	4.39E-07	14	5	2.91E-07	3.23E-09	1.29E-10
8	6	6.00E-05	3.66E-06	5.03E-07	14	6	3.37E-07	3.70E-09	1.49E-10
8	7	6.43E-05	3.90E-06	5.36E-07	14	7	3.60E-07	3.94E-09	1.59E-10
8	8	6.65E-05	4.03E-06	5.54E-07	14	8	3.72E-07	4.06E-09	1.64E-10

Table 3: $\|B_2 - A_2\|_F / \|B_2\|_F$ for different kernels, where $n = k \cdot 2^l$.

In constructing $c(A_2)$, we have also made use of the centro-symmetric property of the matrix A_2, i.e. we only need to compute the diagonal sums of the upper half of A_2. Table 4 gives the numbers of Kflops required to form $c(A_2)$. We remark again that the counts do not depends on the values of λ and $\mathrm{diam}(\partial\Omega)$. In the table, the *ratios* again denote the ratios of the operation counts when the size n of the matrix is doubled. The largest matrix size we tried here is also $n = 14 \cdot 2^{12} = 57,344$. We see from the ratios that the cost of constructing $c(A_2)$ is increasing like $O(nkl) = O(n \log n)$. In contrast, the cost for constructing $c(B_2)$ using (1) is $O(n^2)$ operations.

Next we test the efficiency and accuracy of solving (19) using the approximation A_2 for B_2 and the optimal circulant preconditioner $C + c(A_2)$. Using the conjugate gradient method, we solve for the vector $\mathbf{x}$ in the non-preconditioned system (cf (16))

$$(C + B_2)\mathbf{x} = \mathbf{g}, \tag{23}$$

	k=4		k=8		k=11		k=14	
l	$c(A_2)$	*ratio*	$c(A_2)$	*ratio*	$c(A_2)$	*ratio*	$c(A_2)$	*ratio*
5	126	—	551	—	1,594	—	2,139	—
6	309	2.4546	1,357	2.4620	3,934	2.4676	5,262	2.4599
7	704	2.2808	3,092	2.2792	8,964	2.2789	11,957	2.2721
8	1,556	2.2110	6,818	2.2051	19,741	2.2021	26,256	2.1959
9	3,388	2.1773	14,790	2.1692	42,738	2.1650	56,694	2.1593
10	7,312	2.1581	31,784	2.1490	91,640	2.1442	121,270	2.1390
11	15,685	2.1450	67,885	2.1358	195,276	2.1309	257,845	2.1262
12	33,488	2.1350	144,329	2.1261	414,234	2.1213	545,870	2.1170

Table 4: Kflops counts in constructing $c(A_2)$, where $n = k \cdot 2^l$.

and for the vector $\mathbf{y}$ in the preconditioned system (cf (19))

$$c(C + A_2)^{-1}(C + A_2)\mathbf{y} = c(C + A_2)^{-1}\mathbf{g}. \tag{24}$$

We note that $c(C+A_2) = C+c(A_2)$ is a circulant matrix. Hence its inverse can be found efficiently in $O(n \log n)$ operations by using fast Fourier transforms, see Chan and Ng [8]. Thus the cost per iteration of solving (23) and (24) by conjugate gradient method is $O(n^2)$ and $O(n \log n)$ operations respectively.

For both systems (23) and (24), we choose the zero vector as the initial guess and the stopping criterion is $||\mathbf{r}_q||_2/||\mathbf{r}_0||_2 < 10^{-10}$, where $\mathbf{r}_q$ is the residual vector at the qth iteration. The numbers of iterations required for convergence for different λ are given in Table 5, where the symbols C and I indicate if circulant preconditioning is used or not. From the table, we see that the numbers of iterations of the preconditioned systems are smaller than that of the non-preconditioned ones considerably. Notice that the iteration numbers of the preconditioned systems are uniformly bounded whereas those of the

$\rho = 3/4$		$\lambda = 1.1$			$\lambda = 1.3$			$\lambda = 1.5$		
k	l	I	C	e_n	I	C	e_n	I	C	e_n
4	5	29	9	5.90E-03	30	8	1.60E-03	31	7	1.52E-03
4	6	40	9	6.41E-03	41	7	1.75E-03	41	7	1.62E-03
4	7	54	9	6.83E-03	54	7	1.81E-03	53	7	1.65E-03
4	8	70	9	7.52E-03	71	7	1.86E-03	74	7	1.67E-03
8	5	40	9	1.91E-04	41	7	2.41E-05	41	7	4.33E-06
8	6	54	9	2.03E-04	54	7	2.59E-05	53	7	4.63E-06
8	7	70	9	2.06E-04	71	7	2.61E-05	74	7	4.70E-06
8	8	94	9	2.14E-04	94	7	2.61E-05	95	7	4.77E-06
11	5	51	9	3.76E-05	49	7	1.03E-06	49	7	9.09E-08
11	6	63	9	4.07E-05	63	7	1.14E-06	64	7	9.90E-08
11	7	86	9	4.14E-05	85	7	1.16E-06	85	7	1.01E-07
11	8	117	9	4.26E-05	118	7	1.18E-06	118	7	1.02E-07
14	5	53	9	5.83E-06	53	7	4.81E-08	54	7	5.57E-09
14	6	73	9	6.47E-06	74	7	5.36E-08	74	7	9.01E-09
14	7	93	9	6.51E-06	95	7	5.51E-08	91	7	2.19E-08
14	8	121	9	6.44E-06	123	7	6.11E-08	122	7	3.03E-08

Table 5: Numbers of Iterations and Relative Errors, where $n = k \cdot 2^l$.

original systems are increasing with n as expected.

Finally, we compare the accuracy of the solution $\mathbf{y}$ of the approximate system (24) with the solution $\mathbf{x}$ of (23). We give the relative errors $\|\mathbf{x} - \mathbf{y}\|_2/\|\mathbf{x}\|_2$ for different λ in Table 5 under the column e_n. We see that the solution $\mathbf{y}$ provides a very accurate approximation to the solution $\mathbf{x}$ even for small k.

6 Concluding Remarks

In this paper, we have developed a fast algorithm for computing optimal circulant preconditioners $c(A)$ for dense matrices A constructed from the fast dense matrix method proposed in Chan, Lin and Ng [7]. We remark that besides the fast dense matrix method, there are many other fast multiplication schemes that can provide good approximations to a given integral equation or its dense discretization matrix B, see for instance [1, 3, 14]. For example, using wavelet transforms W, the matrix WBW^t can be approximated by a sparse matrix S accurately. We note that though $c(S)$ can be computed fast using (1) (because of its sparsity) and gives a good approximation to $c(WBW^t)$ (because of (12)), it is however not close to $W \cdot c(B) \cdot W^t$. Hence in general, $c(S)$ will not be a good preconditioner for S. In contrast, $c(A)$ constructed by our fast dense matrix method will provide a good approximation to $c(B)$ and a good preconditioner for A and B.

References

[1] B. Alpert, G. Beylkin, R. Coifman and V. Rokhlin, *Wavelets for the Fast Solution of Second-Kind Integral Equations*, SIAM J. Sci. Comput., 14 (1993), 159–184.

[2] O. Axelsson and V. Barker, *Finite Element Solution of Boundary Value Problems*, Academic Press, Orlando, 1984.

[3] G. Beylkin, R. Coifman and V. Rokhlin, *Fast Wavelet Transforms and Numerical Algorithms I*, Comm. Pure Appl. Math., 46 (1991), 141–183.

[4] R. Chan, X. Jin and M. Ng, *Circulant Integral Operators as Preconditioners for Wiener-Hopf Equations*, Integr. Equat. Oper. Theory, 21 (1995), 12–23.

[5] R. Chan, X. Jin and M. Yeung, *The Circulant Operator in the Banach Algebra of Matrices*, Lin. Algebra Appls., 149 (1991), 41–53.

[6] R. Chan and F. Lin, *Preconditioned Conjugate Gradient Methods for Integral Equations of the Second Kind Defined on the Half-Line*, J. Comput. Math., to appear.

[7] R. Chan, F. Lin and W. Ng, *Fast Dense Matrix Method for the Solution of Integral Equations of the Second Kind*, Res. Rept. #96-05, Math. Dept., Chinese University of Hong Kong, submitted.

[8] R. Chan and M. Ng, *Conjugate Gradient Methods for Toeplitz Systems*, SIAM Review, Sept. 1996.

[9] R. Chan, H. Sun and W. Ng, *Circulant Preconditioners for Ill-Conditioned Boundary Integral Equations from Potential Equations*, Res. Rept. #96-20, Math. Dept., Chinese University of Hong Kong, submitted.

[10] T. Chan, *An Optimal Circulant Preconditioner for Toeplitz Systems*, SIAM J. Sci. Statist. Comput., 9 (1988), 766–771.

[11] G. Chen and J. Zhou, *Boundary Element Methods*, Academic Press, London, 1992.

[12] I. Gohberg, M. Hanke and I. Koltracht, *Fast Preconditioned Conjugate Gradient Algorithms for Wiener-Hopf Integral Equations*, SIAM J. Numer. Anal., 31 (1994), 429–443.

[13] L. Greengard and V. Rokhlin, *A Fast Algorithm for Particle Simulations*, J. Comput. Phys., 73 (1987), 325–348.

[14] L. Reichel, *Fast Solution Methods for Fredholm Integral Equations of the Second Kind*, Numer. Math., 57 (1989), 719–736.

[15] E. Tyrtyshnikov, *Optimal and Super-Optimal Circulant Preconditioners*, SIAM Matrix Anal. Appl., 13 (1992), 459–473.