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

Raymond H. Chan* Fu-Rong Lin† Wing-Fai Ng‡

Abstract

We present a fast algorithm based on polynomial interpolation to approximate matrices arising from the 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 [6] 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 [7]), 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 [8, 11, 2, 1]. The fast multipole method proposed in [8] 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 [11], the integral equation is discretized at Chebyshev points

*Department of Mathematics, Chinese University of Hong Kong, Shatin, Hong Kong. Research supported in part by Hong Kong Research Grant Council grants no. CUHK178/93E.

†Department of Mathematics, Shantou University, Shantou 515063, Guangdong, People's Republic of China.

‡Department of Mathematics, Chinese University of Hong Kong, Shatin, Hong Kong.

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, if the kernel function is not smooth enough, such as the kernel functions discussed in that paper, 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 [8, 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 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. Finally in §6, we will give concluding remarks.

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 [10, 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 [7, 3]. They converge to the solution in a linear rate, cf. [9] and Table 2 in §5.

To find the solution numerically, we discretize (2) by Nyström's method (see [6]) 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 $\mathbf{I}$ is the identity matrix, $\mathbf{g}$ is a given vector and $\mathbf{f}$ is the unknown vector. As in [1], we define the entries of the discretization matrix $\mathbf{A}$ to be

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

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 $\mathbf{A}\mathbf{y}$ should be done fast for any vector $\mathbf{y}$. For $\mathbf{A}$ defined in (4), the multiplication requires $O(n^2)$ operations. In §3, we will find an approximation $\tilde{\mathbf{A}}$ of $\mathbf{A}$, 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 $\mathbf{A}$ 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 $\mathbf{A}$ or submatrices of $\mathbf{A}$ 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 $\mathcal{DA}$ will be a compact operator. This is because product of bounded operator and compact operator is still compact, see for instance [12, Theorem 4.18]. Hence $\mathcal{I}-\mathcal{DA}$ 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

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

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^uk-by-2^uk. 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

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

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

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

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.

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

   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.

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

   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 [7]), then there is no need to explicitly form the approximation $\tilde{\mathbf{A}}^{(u,v)}$, and we only have to store $\tilde{\mathbf{\Lambda}}^{(u,v)}$ 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], polynomial bases functions are used in the interpolation without mapping to the domain $[-1,1]^2$ first. The resulting basis function matrices are then shifted and scaled by methods different from ours to make them more stable. Their condition numbers, which vary with different blocks and depends on n, are about the same order as that of our $\mathbf{P}_k$.

Another important advantage of having this v independence in the basis function matrices is that we can use the same $\mathbf{P}^{(u)}$ for all $\tilde{\mathbf{A}}^{(u,v)}$. 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)):

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

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 $\mathbf{A}\mathbf{x}$ 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.

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

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

n	$\tilde{\mathbf{A}}\mathbf{x}$	*ratio*	$\mathbf{A}\mathbf{x}$	*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 $\mathbf{A}\mathbf{x}$ for $k = 11$.

n	$\tilde{\mathbf{A}}\mathbf{x}$	*ratio*	$\mathbf{A}\mathbf{x}$	*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 $\mathbf{A}\mathbf{x}$ for $k = 14$.

6 Concluding Remarks

In this paper, we have discussed the fast solution of second-kind integral equation 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. We have shown that our approximation coefficient matrix $\tilde{\mathbf{A}}$ can be constructed in $O(n)$ operations and requires $O(n)$ storage, and the matrix-vector multiplication of $\tilde{\mathbf{A}}$ requires $O(n \log n)$ operations. The numerical results show that our scheme is stable for high degree polynomial interpolation and to reach a given tolerance, the number of iterations of CGLS is independent of the size n of the discretization system and small. For an application of our scheme, we refer readers to [4, 5], where we discuss fast solution of boundary integral equations.

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] R. Chan, W. Ng and H. Sun, *Fast Construction of Optimal Circulant Preconditioners for Matrices from Fast Dense Matrix Method*, Res. Rept. #96-21, Math. Dept., Chinese University of Hong Kong, submitted.

[5] 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.

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

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

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

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

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

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

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