NUMERICAL SOLUTION OF INTEGRAL EQUATION OF THE SECOND KIND

by

Chi-Fai CHAN

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

July, 1998
Abstract of thesis entitled

'NUMERICAL SOLUTION OF INTEGRAL EQUATION OF THE SECOND KIND'

submitted by

Chi-Fai CHAN

for the degree of Master of Philosophy

at The Chinese University of Hong Kong in June, 1998

In this thesis, we consider solutions of Fredholm integral equations of the second kind where the kernel functions are asymptotically smooth or a product of such function with a highly oscillatory coefficient function. We present an approximation scheme based on polynomial interpolation to approximate the discretization matrix $A$ which arise from the discretization of these integral operators. The idea of the scheme is to take advantage of the smoothness of the kernel function away from the singularity where we can use low degree polynomials to approximate the kernel function quite accurately. As $A$ possesses the same smoothness properties of the kernel function, we can use low rank matrices obtained by using polynomial interpolation to approximate $A$.

We start our approximation scheme with the same approach as in [1] and [2]. Our approximation matrix $B$ is obtained by partitioning the domain $[0, 1]^2$ into subdomains of different sizes and approximate the kernel function at each subdomain by using the Lagrange interpolation polynomials at the Chebyshev points. As the Chebyshev points are optimal interpolation points, see [7, pp.284–287], our approximation scheme can get more accurate numerical results than that in [2]. Although our approximation matrix $B$ is dense, it can be constructed in only $O(n)$ operations and requires only $O(n)$ storage and
the cost of matrix-vector multiplication $By$ is of order $O(n \log n)$ where $n$ is the number of quadrature points used in the discretization. We prove that for a given accuracy $\epsilon$ (in Frobenius matrix norm), the degree of the interpolation polynomials is of order $O(\log \epsilon^{-1})$ in order that $\|A - B\|_F \leq \epsilon$ where $\|\cdot\|_F$ is the Frobenius norm. With this result, we can show that the memory requirement of our approximation method is of the same order as that in [11]. Besides, it follows that by using conjugate gradient type methods, the numerical solution of the second kind integral equations requires $O(n \log(n/\epsilon))$ operations.
摘要

本篇論文主要討論第二種 Fredholm 積分方程的解。我們所討論的積分方程中的核函數全部都是漸近平滑的或是一個漸近平滑的函數和一個高度振盪的係數函數的積。我們會利用一個建基於多項式插值法的逼近方法去逼近一個離散矩陣 A，而該離散矩陣 A 是將我們所考慮的積分算子離散化後所獲取的。這個方法的好處在於利用了核函數遠離奇點時的光滑性。這性質能令我們用一低階的多項式較準確地去逼近該核函數。由於矩陣 A 擁有和核函數相同的光滑性，因此我們可用多項式插值法去獲取一個低秩的矩陣來逼近 A。

我們的逼近方法先將定義域 [0,1]^2 分割成不同大小的子域，然後我們在每個子域上用建立於 Chebyshev 點上的 Lagrange 多項式插值法去逼近核函數，從而獲得我們的逼近矩陣 B。由於 Chebyshev 點是最優化的插値點。因此我們可獲取較佳的數值結果。雖然矩陣 B 是稠密的，但它的建立和儲存只需 O(n) 的運算步驟。除此之外，一支任意向量和它的積亦只需 O(n \log n) 的運算步驟。其中 n 是插値點的數目。

除此之外，我們亦可證明當我們給出一個任意的 \varepsilon 並要求 ||A - B||_F \leq \varepsilon (\|\cdot\|_F 是 Frobenius 矩陣範數) 時，我們所用的多項式階數是 O(\log \varepsilon^{-1})。由此結果，我們可以知道當我們利用共軛梯度式方法時，我們可在 O(n \log (n/\varepsilon)) 的運算步驟內獲得所需的數值結果。
DECLARATION

The author declares that this thesis represents his own work based on the ideas suggested by Prof. Raymond H.F. Chan, the author's supervisor. All the work is done under the supervision of Prof. Raymond H.F. Chan during the period 1996-1998 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 or report submitted to any institution for a degree, diploma or other qualification.

________________________________________
Chi-Fai CHAN
To
My parents
ACKNOWLEDGMENT

I wish to express my sincere gratitude to my supervisor, Prof. Raymond H.F. Chan, for his inspired guidance, constant encouragement and help throughout the period of my M.Phil. studies and in the preparation of this thesis. I would like to thank Drs. X.Q. Jin, F.R. Lin, M. Ng, H.W. Sun, J. Zou, my colleagues Mr. C.P. Cheung, H.C. Chan, W.K. Ching, K.W. Mak, W.F. Ng, W.C. Tang, T.M. Tso, C.K. Wong, H.M. Zhou and L.L. Heung for their many helpful discussions.
CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>§1.1 Polynomial Interpolation</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>§1.2 Conjugate Gradient Type Methods</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>§1.3 Outline of the Thesis</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>INTEGRAL EQUATIONS</td>
<td>11</td>
</tr>
<tr>
<td>§2.1 Integral Equations</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>§2.2 Numerical Treatments of Second Kind Integral Equations</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Chapter 3</td>
<td>FAST ALGORITHM FOR SECOND KIND INTEGRAL EQUATIONS</td>
<td>20</td>
</tr>
<tr>
<td>§3.1 Introduction</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>§3.2 The Approximation</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>§3.3 Error Analysis</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>§3.4 Numerical Examples</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>§3.5 Concluding Remarks</td>
<td></td>
<td>51</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>53</td>
</tr>
</tbody>
</table>
In this thesis, we present a fast matrix-vector multiplication scheme to solve the Fredholm integral equations of the second kind where the kernel functions are asymptotically smooth or a product of such function with a highly oscillatory coefficient function. We construct the fast algorithm by using an approximation matrix $B$ to approximate the discretization matrix $A$ which arises from the discretization of these integral operators. We use the techniques of polynomial interpolation to construct our approximation scheme. Therefore, we present the mathematical background about the polynomial interpolation in this chapter.

In this chapter, we also discuss the conjugate gradient method which is an iterative method that we use to solve the matrix equation which arises from our approximation scheme.

1.1 Polynomial Interpolation

In this section, we present two forms of interpolation polynomials. One is the Newton form and the other is the Lagrange form. Before we present these two forms, we present the basic idea of polynomial interpolation.

Here, we give a simple example to illustrate the idea of polynomial interpolation. Suppose that we have three distinct points $x_0$, $x_1$, $x_2$ and their corresponding function values are $y_0$, $y_1$ and $y_2$. We want to know that if we can find a polynomial $p(x)$ such that $p(x_i) = y_i$ for $i = 0, 1$ and $2$. If we find such $p(x)$ satisfying the above interpolation conditions, then we say the
polynomial is interpolating the data. More precisely, let

\[ p_2(x) = a_2x^2 + a_1x + a_0 \]

where \(a_0, a_1\) and \(a_2\) are to be determined such that \(p_2(x_i) = y_i\) for \(i = 0, 1\) and \(2\). Then, we have a linear system of three equations with three unknowns. If we use matrix form to represent them, we get the following matrix equation:

\[
\begin{pmatrix}
 x_0^2 & x_0 & 1 \\
 x_1^2 & x_1 & 1 \\
 x_2^2 & x_2 & 1
\end{pmatrix}
\begin{pmatrix}
 a_2 \\
 a_1 \\
 a_0
\end{pmatrix}
=
\begin{pmatrix}
 y_0 \\
 y_1 \\
 y_2
\end{pmatrix}
\]

In the above matrix equation, the 3-by-3 matrix \(V\) is called the Vandermonde matrix. Since \(x_0, x_1\) and \(x_2\) are distinct, \(V\) is non-singular and hence there is a unique solution. This analysis can be extended to polynomial interpolation on \(n + 1\) distinct points which leads to the following theorem, see [7, p. 278].

**Theorem 1** If \(x_0, x_1, \ldots, x_n\) are distinct real numbers, then for arbitrary values \(y_0, y_1, \ldots, y_n\), there is a unique polynomial \(p_n\) of degree at most \(n\) such that

\[ p_n(x_i) = y_i \quad (0 \leq i \leq n). \]

In the following, we present the two different forms of interpolation polynomials. One is called the Newton form of the interpolation polynomial and the other is called the Lagrange form of the interpolation polynomial. In the following two subsections, we will use the same notations as in Theorem 1, i.e. \(p_n(x)\) denotes an interpolation polynomial of degree \(n\) with distinct nodes \(x_0, x_1, \ldots, x_n\).

### 1.1.1 Newton Form of the Interpolation Polynomial

The main idea of the Newton form is that we construct a degree \(n\) interpolation polynomial \(p_n(x)\) from a known degree \(n - 1\) interpolation polynomial \(p_{n-1}(x)\)
and known additional point $x_n$. Let $r(x)$ be such additional term of degree $n$ such that

$$p_n(x) = p_{n-1}(x) + r(x).$$

Since we want $p_n(x_i) = p_{n-1}(x_i) = y_i$, we have $r(x_i) = 0$ for $i = 0, \ldots, n-1$ and $r(x)$ must be of the form

$$r(x) = a_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}).$$

Hence, the interpolation polynomial $p_n(x)$ can be written as

$$p_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0) \cdots (x - x_{n-1}) \quad (1.1)$$

where $a_0, \ldots, a_n$ are constants to be determined.

Now, we see how to determine the coefficients $a_0, \ldots, a_n$. From (1.1), we observe that $y_0 = p_n(x_0) = a_0$ and $p_n(x_1) = a_0 + a_1(x_1 - x_0) = y_0 + a_1(x_1 - x_0) = y_1$. Therefore,

$$a_1 = \frac{y_1 - y_0}{x_1 - x_0}.$$ 

Since the numerator and denominator are differences, we define the first divided difference as follows

$$f[x_1, x_0] = \frac{p_n(x_1) - p_n(x_0)}{x_1 - x_0}.$$ 

It is easily to show that $f[x_1, x_0] = f[x_0, x_1]$ and $a_1 = f[x_1, x_0]$. Similarly, we can further show that

$$a_2 = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0}.$$

In summary, if we define

$$f[x_2, x_1, x_0] = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0},$$

then we have $a_2 = f[x_2, x_1, x_0]$ and the coefficient $a_i$ is the $i$th order divided difference which is defined by

$$f[x_i, x_{i-1}, \ldots, x_0] = \frac{f[x_i, \ldots, x_1] - f[x_{i-1}, \ldots, x_0]}{x_i - x_0}.$$
Therefore, the interpolation polynomial in the Newton form can be written as
\[ p_n(x) = y_0 + \sum_{i=1}^{n} f[x_i, x_{i-1}, \ldots, x_0](x - x_0) \cdots (x - x_{i-1}). \quad (1.2) \]

### 1.1.2 Lagrange Form of the Interpolation Polynomial

From the above subsection, we know that if we use the Newton form, we need to determine the coefficients \( a_i \) of the interpolation polynomial \( p_n(x) \). Suppose that we use the values \( y_0, \ldots, y_n \) to act as the coefficients of \( p_n(x) \), i.e.
\[ p_n(x) = y_0l_0(x) + y_1l_1(x) + \cdots + y_nl_n(x) = \sum_{i=0}^{n} y_il_i(x). \quad (1.3) \]
where \( l_0, \ldots, l_n \) are polynomials that depend on the nodes \( x_0, \ldots, x_n \) but not on the ordinates \( y_0, \ldots, y_n \). Since \( p_n(x_i) = y_i \) for all \( i = 0, \ldots, n \), we have
\[ \delta_{ij} = l_i(x_j) \]
(Recall that \( \delta_{ij} = 1 \) if \( i = j \) and \( \delta_{ij} = 0 \) if \( i \neq j \).) Therefore, \( l_i(x) \) must be of the following form
\[ l_i(x) = c_i(x - x_0) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n) \]
where \( c_i \) is a constant to be determined. Since \( l_i(x_i) = 1 = c_i(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n) \), we can deduce that
\[ c_i = \frac{1}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)}. \]
Therefore,
\[ l_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}, \quad 0 \leq i \leq n \quad (1.4) \]
and the Lagrange form of the interpolation polynomial is as follows
\[ p_n(x) = \sum_{i=0}^{n} y_il_i(x). \quad (1.5) \]

In Chapter 3, we will use the Lagrange interpolation polynomial presented here in our approximation scheme.
1.1.3 The Error in Polynomial Interpolation

After presenting two algorithms to find the interpolation polynomials, we present the error between a function and a polynomial interpolant to it. Two theorems presented in the following will be used in Chapter 3. The general error estimate is given in the following theorem, see [7, p. 284].

**Theorem 2** Let \( f \) be a function in \( C^{n+1}[a,b] \) and \( p \) be the polynomial of degree \( \leq n \) that interpolates the function \( f \) at \( n+1 \) points \( x_0, x_1, \ldots, x_n \) in the interval \( [a,b] \). To each \( x \) in \( [a,b] \) there corresponds a point \( \xi_x \) in \( (a,b) \) such that

\[
f(x) - p(x) = \frac{1}{(n+1)!}f^{(n+1)}(\xi_x) \prod_{i=0}^{n}(x - x_i). \tag{1.6}
\]

Theorem 2 gives us a general error estimate of polynomial interpolation. For example, let \( f(x) = \cos(x) \) and we use a degree nine polynomial that interpolates \( f \) at ten points in \([0,1]\). By using (1.6), we see that the error is about \( 10^{-7} \).

After getting this general result, we start to think about that if we can do something to minimize the error due to polynomial interpolation. In general, we do not know \( f^{(n+1)}(\xi_x) \). Hence, we try to select \( x_0, x_1, \ldots, x_n \) such that

\[
\max_{0 \leq x \leq b} \left| \prod_{i=0}^{n}(x - x_i) \right| \]

is minimized. This question is solved by introducing the Chebyshev polynomials. For \( x \) in \([-1,1]\), the Chebyshev polynomial of degree \( n \) is defined as

\[
T_n(x) = \cos(n \cos^{-1}(x)) \quad (n \geq 0).
\]

and the roots of \( T_n(x) \) are given by

\[
x_i = \cos\left(\frac{2i - 1}{2n} \pi\right), \quad 1 \leq i \leq n. \tag{1.7}
\]

If the nodes \( x_i \) are the roots of the Chebyshev polynomial \( T_{n+1} \), we can prove that the minimum value of

\[
\max_{|x| \leq 1} \left| \prod_{i=0}^{n}(x - x_i) \right|
\]
is $1/2^n$, see [7, p. 287]. Therefore, we can get the following theorem, see [7, p. 287].

**Theorem 3** If the nodes $x_i$ are the roots of the Chebyshev polynomial $T_{n+1}$, then the error formula in Theorem 2 yields (for $|x| \leq 1$)

$$|f(x) - P(x)| \leq \frac{1}{2^n(n+1)!} \max_{|t| \leq 1} f^{(n+1)}(t).$$

We will combine the results of Theorems 2 and 3 in Chapter 3 to carry out the error analysis of our approximation scheme.

### 1.2 Conjugate Gradient Type Methods

There are two main types of methods for solving the following general linear system,

$$Ax = b,$$  \hspace{1cm} (1.8)

where $A$ is an $n$-by-$n$ matrix. One is the direct methods such as the Gaussian elimination and the other is the iterative methods such as Jacobi, SOR and Gauss-Seidel method. For large systems, iterative methods are more efficient than direct methods. Here, we introduce the Conjugate Gradient Method which is one of the iterative methods and is suitable for solving symmetric positive definite linear systems. According to G. Golub and C. Van Loan[12, p. 523], we have

**Algorithm 1** [Conjugate Gradients] If $A \in R^{n \times n}$ is symmetric positive definite and $b \in R^{n \times n}$, then the following algorithm computes $x \in R^n$ so $Ax = b$.  

$k = 0$; $x_0 = 0$; $r_0 = b$

while $r_k \neq 0$

1. $k = k + 1$
2. if $k = 1$
   1. $p_1 = r_0$
3. else
\begin{align*}
\beta_k & = r_{k-1}T r_{k-1}^T r_{k-2}T r_{k-2} \\
p_k & = r_{k-1} + \beta_k p_{k-1} \\
& \quad \text{end} \\
\alpha_k & = r_{k-1}^T r_{k-1} / p_k^T A p_k \\
x_k & = x_{k-1} + \alpha_k p_k \\
r_k & = r_{k-1} - \alpha_k A p_k \\
& \quad \text{end} \\
x & = x_k
\end{align*}

If \( A \) is nonsymmetric and nonsingular, then we can solve the equivalent system

\[ A^T A x = A^T b \]  \hspace{1cm} (1.9)

which is also symmetric positive definite. This system is known as the system of normal equation. We can apply the conjugate gradient method to the linear system (1.9) and get the following CGNR method, see [4].

**Algorithm 2** [CGNR] Let \( x_0 \) be an initial guess to the linear system \( A x = b \) where \( A \) is nonsymmetric.

\begin{align*}
\rho_0 & = b - A x_0 \\
p_0 & = s_0 = A^T r_0 \\
\gamma_0 & = ||s_0||^2 \\
& \quad \text{for } k = 0, 1, 2, \ldots \\
q_k & = A p_k \\
\alpha_k & = \gamma_k / ||q_k||^2 \\
x_{k+1} & = x_k + \alpha_k p_k \\
r_{k+1} & = r_k - \alpha_k p_k \\
s_{k+1} & = A^T r_{k+1} \\
\gamma_{k+1} & = ||s_{k+1}||^2 \\
\beta_k & = \gamma_{k+1} / \gamma_k \\
p_{k+1} & = s_{k+1} + \beta_k p_k
\end{align*}
We see from the above two algorithms that the main cost comes from the matrix-vector multiplication as the cost is $O(n^2)$ for non-structural $A$. In practical, $n$ is usually very large, so it may seem that the algorithm is not applicable for a large scale problem. Besides, we actually can not find the exact solution due to the rounding errors. Thus, we need to discuss the error estimate of the approximation solution of these two algorithms after some iterations. Before we state the theorem of the convergence properties of the conjugate gradient method, we first recall that $||x||_A$ denotes the norm defined by

$$||x||_A = (Ax, x)^{\frac{1}{2}}.$$ 

The error estimate is shown as follows, see Axelsson and Barker [13, p.26].

**Theorem 4** Let $T_k$ be the $k$th degree Chebyshev polynomial, $x_k$ be the $k$th iterant of the conjugate gradient method applying to the linear system (1.8) and $y$ to be the true solution of the linear system (1.8). For the conjugate gradient method, we have

$$||y - x_k||_A \leq \left( T_k \left( \frac{\lambda_{\text{max}} + \lambda_{\text{min}}}{\lambda_{\text{max}} - \lambda_{\text{min}}} \right) \right)^{-1} \ ||y - x_0||_A,$$

where $\lambda_{\text{max}}$ and $\lambda_{\text{min}}$ are the largest and smallest eigenvalues of $A$ respectively. Moreover, if $k(\epsilon)$ is defined for any $\epsilon > 0$ to be the smallest integer $k$ such that

$$\frac{||y - x_k||_A}{||y - x_0||_A} \leq \epsilon,$$

then

$$k(\epsilon) \leq \frac{1}{2} \sqrt{\kappa(A)} \ln(\frac{2}{\epsilon}) + 1.$$ 

From the above theorem, we know that the convergence rate of the conjugate gradient method is linear if the spectral condition number $\kappa(A)$ is uniformly bounded. If in particular that the eigenvalues of $A$ are clustered around 1, then the convergence rate of the conjugate gradient method will become superlinear by the following theorem, see [14].
**Theorem 5** Let \( x_k \) be the \( k \)th iterant of the conjugate gradient method applied to the linear system (1.8) and \( y \) be the true solution of (1.8). If the eigenvalues \( \lambda_j \) \( (j = 1, 2, \ldots, n) \) of \( A \) are ordered such that

\[
0 < \lambda_1 \leq \cdots \leq \lambda_p \leq b_1 \leq \lambda_{p+1} \leq \cdots \leq \lambda_{n-q} \leq b_2 \leq \lambda_{n-q+1} \leq \cdots \leq \lambda_n
\]

then

\[
\frac{\|y - x_k\|_A}{\|y - x_0\|_A} \leq \left( \frac{\sqrt{b_2}}{\sqrt{b_1}} - 1 \right) \frac{2}{\sqrt{b_2}} \leq \max_{\lambda \in [b_1, b_2]} \prod_{i=1}^{p} \left( \frac{\lambda - \lambda_i}{\lambda_i} \right).
\]

In many cases, the spectra of many matrices are not clustered around a certain point. So, we hope to transform the linear system in the following form

\[
C_n^{-1}Ax = C_n^{-1}b.
\] (1.10)

We hope that the spectra of \( C_n^{-1}A \) will be clustered around a certain point and we call \( C_n \) the preconditioner. The preconditioned linear system (1.10) can be solved by the preconditioned conjugate gradient method, see [12, p. 529].

**Algorithm 3** [Preconditioned Conjugate Gradient] Given a symmetric positive definite \( A \in \mathbb{R}^{n \times n} \) and \( b \in \mathbb{R}^{n \times n} \), then the following algorithm solves \( Ax = b \) using the method of conjugate gradients with preconditioner \( C \in \mathbb{R}^{n \times n} \).

\[
k = 0; x_0 = 0; r_0 = b
\]

while \( r_k \neq 0 \)

- Solve \( Cz_k = r_k \)
- \( k = k + 1 \)
- if \( k = 1 \)
  - \( p_1 = z_0 \)
- else
  - \( \beta_k = r_k^T z_{k-1} / r_{k-2}^T r_{k-2} \)
  - \( p_k = z_{k-1} + \beta_k p_{k-1} \)
- end
- \( \alpha_k = r_k^T z_{k-1} / p_k^T A p_k \)
- \( x_k = x_{k-1} + \alpha_k p_k \)
\[ r_k = r_{k-1} - \alpha_k A p_k \]
end
\[ x = x_k \]

In order to make the above algorithm to be an effective technique, we need a fast algorithm to solve \( C z_k = r_k \). From [15], we know that if \( C \) is a circulant matrix, then \( C z_k = r_k \) can be solved in \( O(n \log n) \) operations. Hence, there are many circulant preconditioner proposed, see G. Strang [18], R. Chan [16] and T. Chan [17].

### 1.3 Outline of the Thesis

After an introduction of the mathematical background about the polynomial interpolation and the conjugate gradient type method, we will see how to apply them in our fast matrix-vector multiplication scheme to solve the second kind integral equations in Chapter 3. Since our target is to introduce a fast algorithm to solve the second kind integral equations, we will introduce the general idea of integral equations in the next chapter. In Chapter 2, we will first give out the classification of the integral equations. Then, we will discuss about some of the numerical treatments of the second kind integral equations.

In Chapter 3, we will present our fast matrix-vector multiplication scheme for solving the second kind integral equation. The scheme is based on polynomial interpolation to approximate the kernel function of the integral equations. Numerical results are also given in Chapter 3. We will use them to show that our method is accurate and stable. Chapter 3 is mainly an extract from the paper "Polynomial Interpolation for Fast Solutions of the Second Kind Integral Equations" by R. Chan, F. Lin and the author of this thesis.
Chapter 2

INTEGRAL EQUATIONS

In this chapter, we present the subjects of integral equations and the numerical treatments of Fredholm equations of the second kind. The integral equation formulation of physical problems proves to be more elegant and compact than the differential equation formulation, since the auxiliary initial and boundary conditions are automatically satisfied in the integral equation formulation. Besides, the form of the solution of an integral equation is often more suitable for today's extremely fast machine computation. Therefore, it is a good mathematical tool to solve different kinds of problems such as population dynamics, control problems, mechanics problems and etc.

In §2.1, we present how to reduce an initial value problem to Volterra integral equation and how to reduce a boundary value problem to Fredholm integral equation. In the other words, we introduce the main types of integral equations first. Then, we give a more detailed classification of the integral equations. In §2.2, we mainly present the numerical treatments of Fredholm equations of the second kind. Nyström's method and the Galerkin method are introduced in this section.

2.1 Integral Equations

An integral equation is an equation in which the unknown function $f(x)$ appears under an integral sign. A general example of an integral equation in $f(x)$ is

$$
\gamma f(x) - \int_{a}^{b(x)} K(x, t) f(t) dt = g(x).
$$

(2.1)
where $K(x, t)$ is called the kernel function. Some problems have their mathematical representation appear directly in terms of integral equations. On the other hand, problems which have direct representation in terms of differential equations and their auxiliary conditions may also be reduced to integral equations. We will give two examples in the following paragraphs to illustrate how to reduce the initial value problem and the boundary value problem to integral equation.

Consider the following initial value problem,

$$\frac{d^2 y}{dx^2} = \lambda y(x) + g(x),$$

\[ y(0) = 1 \]
\[ y'(0) = 0. \]

Let

$$\frac{d^2 y}{dx^2} = f(x),$$

and integrate twice with respect to $x$, we get

$$y(x) = \int_0^x (x-t)f(t)dt + c_1 x + c_2. \quad (2.2)$$

By using $y(0) = 1$ and $y'(0) = 0$, we can deduce that $c_1 = 0$ and $c_2 = 1$. After substituting $f(x) = \frac{d^2 y}{dx^2} = \lambda y(x) + g(x)$ into (2.2), we have

$$y(x) - \lambda \int_0^x (x-t)y(t)dt = 1 + \int_0^x (x-t)g(t)dt \quad (2.3)$$

which is an integral equation with the same format as (2.1).

Next, we consider a boundary value problem of the following form.

$$\frac{d^2 y}{dx^2} = \lambda y(x), \quad a < x < b, \quad (2.4)$$

\[ y(a) = 0, \]
\[ y(b) = 0. \]
If we integrate (2.4) twice with respect to $x$, we can get

$$y(x) = \lambda \int_a^x (x-t)y(t)dt + c_1 x + c_2. \quad (2.5)$$

By using the boundary conditions $y(a) = 0 = y(b)$, we have

$$c_1 = \frac{\lambda}{a-b} \int_a^b (b-t)y(t)dt$$

and

$$c_2 = -\frac{a\lambda}{a-b} \int_a^b (b-t)y(t)dt.$$

Therefore, we finally obtain

$$y(x) = \lambda \frac{x-a}{b-a} \int_a^b (t-b)y(t)dt + \lambda \int_a^x (x-t)y(t)dt. \quad (2.6)$$

Now, we do some modification in (2.6) in order to get our required form. This is done by writing the first integral in (2.6) as two parts on the intervals $[a, x]$ and $[x, b]$ where the first part can then be combined with the second integral in (2.6).

$$y(x) = \lambda \frac{x-a}{b-a} \int_a^x (t-b)y(t)dt + \lambda \int_a^b (t-b)y(t)dt$$

$$+ \lambda \int_a^x (x-t)y(t)dt$$

$$= \lambda \int_a^x \frac{(x-b)(t-a)}{b-a} y(t)dt + \lambda \int_x^b \frac{(x-a)(t-b)}{b-a} y(t)dt \quad (2.7)$$

If we define the kernel function $K(x, t)$ as

$$K(x, t) = \begin{cases} 
\frac{(x-b)(t-a)}{(b-a)}, & a \leq t \leq x \\
\frac{(x-a)(t-b)}{(b-a)}, & x \leq t \leq b.
\end{cases}$$

Then, (2.7) can be reduced to the following integral equation

$$y(x) = \lambda \int_a^b K(x, t)y(t)dt. \quad (2.8)$$

From the above two examples we see that there is a close connection between the integral and differential formulations of a given problem. Therefore, we have the following classification of integral equations:
1. If \( b(x) = x \) in (2.1), the integral equation is called a Volterra equation. Moreover, if
   
   (a) \( \gamma = 0 \) in (2.1), the integral equation is called a Volterra equation of the first kind.
   
   (b) \( \gamma = 1 \) in (2.1), the integral equation is called a Volterra equation of the second kind.

2. If \( b(x) = b \) in (2.1), the integral equation is called a Fredholm equation. Moreover, if
   
   (a) \( \gamma = 0 \) in (2.1), the integral equation is called a Fredholm equation of the first kind.
   
   (b) \( \gamma = 1 \) in (2.1), the integral equation is called a Fredholm equation of the second kind.
   
   (c) \( \gamma = 1 \) and \( g(x) = 0 \) in (2.1), we refer the homogeneous equation as an eigenvalue equation or a Fredholm equation of the third kind.

According to the classification of the integral equations, we can see that (2.3) is a Volterra equation of the second kind and (2.8) is the Fredholm equation of the third kind.

Like other equations, we distinguish linear integral equations from nonlinear integral equations depending on whether the equation is linear with respect to the unknown function \( f(x) \). Thus, the following integral equation

\[
    f(x) - \int_a^b K(x, t)f(t)dt = g(x)
\]

is a nonlinear Fredholm equation of the second kind.

Although the Volterra and Fredholm equations are very similar except for the limits in the integral, we can see from the above two examples that they come from different origins and hence have different methods of solution. In the following section, we will discuss mainly on the numerical solutions of the Fredholm equation of the second kind.
2.2 Numerical Treatments of Second Kind Integral Equations

In this section, we present two methods to get the numerical solution of the Fredholm equation of the second kind:

$$f(x) - \int_a^b K(x, t)f(t)dt = g(x). \quad (2.9)$$

One is Nyström's method (or quadrature method) and the other is the Galerkin method. Before introducing these two methods, we make some assumptions on $K(x, t)$ and $g(x)$ first. We assume that $K(x, t)$ is continuous for $a \leq x, t \leq b$ and $g(x)$ is continuous for $a \leq x \leq b$ and we find the solution $f(x)$ for $a \leq x \leq b$.

2.2.1 Nyström's Method (Quadrature method)

Before presenting Nyström's method, we first discuss about what is a quadrature rule. Quadrature rule is any numerical method for evaluating an approximation to an integral $\int_a^b f(t)dt$ of $f(t)$. Therefore, the general form of a quadrature rule is as follows:

$$Qf = \sum_{j=0}^{n} w_j f(t_j) = \int_a^b f(t)dt - \mathcal{E}f. \quad (2.10)$$

where the weights $w_j$ and the quadrature points $t_j$ depend on the interval of integration $[a, b]$. In (2.10), $\mathcal{E}f$ denotes the error of the quadrature rule.

We start to introduce Nyström's method now. By discarding the error term and introducing (2.10) for the interval $[a, b]$ into (2.9), we get:

$$\tilde{f}(x) - \sum_{j=0}^{n} w_j K(x, t_j)\tilde{f}(t_j) = g(x), \quad a \leq x \leq b, \quad (2.11)$$

where $\tilde{f}(x)$ is regarded as the approximation of $f(x)$ in (2.9). The solution of (2.11) can be founded by setting $x = t_i$ for $1 \leq i \leq n$ and (2.11) becomes:

$$\tilde{f}(t_i) - \sum_{j=0}^{n} w_j K(t_i, t_j)\tilde{f}(t_j) = g(t_i), \quad 0 \leq i \leq n. \quad (2.12)$$
If we can find \( f(t_i) \) for \( 0 \leq i \leq n \) that satisfy (2.12), then the solution of (2.11) on the interval \([a, b]\) can be computed by

\[
fr(x) = g(x) + \sum_{j=0}^{n} w_j K(x, t_j) f(t_j)
\]

(2.13)

where \( f_r(t_i) = \tilde{f}(t_i) \) for \( 1 \leq i \leq n \). In other words, \( f_r(t_i) \) satisfies

\[
(I - K) f_r = g
\]

(2.14)

where \( I \) is the identity matrix, \( K_{ij} = w_j a(t_i, t_j) \) and \( g = [g(t_0), \ldots, g(t_n)]^T \).

Here, we call (2.14) the Nyström equation. In Chapter 3, we will use Nyström's method to discretize the integral operators.

After discussing the idea of the Nyström method, we present the error between \( f_r \) in (2.14) and the exact solution \( f \) of (2.9). We let \( e_r(x) = f(x) - f_r(x) \). According to the deviation by Delves and Mohamed, [19], we have

\[
f_r(x) - \int_a^b K(x, t) f_r(t) dt = g(x) - \mathcal{E}(K(x, t) f_r(t))
\]

where \( \mathcal{E} \) is the error operator of the quadrature rule. If we subtract the above equation from (2.9), we get

\[
e_r(x) - \int_a^b K(x, t) e_r(t) dt = \mathcal{E}(K(x, t) f_r(t)).
\]

Therefore, the error satisfies a Fredholm equation of the second kind and has the solution

\[
e_r(x) = [I - K]^{-1} \mathcal{E}(K(x, t) f_r(t)).
\]

(2.15)

From (2.15), we see that the error \( e_r(x) \) depends on the conditioning of the original integral equation which is suggested by the term \([I - A]^{-1}\). This factor reflects the sensitivity of the solution to the small change in the integral operator. Moreover, the error also depends on the approximation error of the quadrature rule used. Therefore, we always want to choose a suitable quadrature rule to reduce the error in integrating the function \( K(x, t) f_r(t) \) over the variable \( t \) for every fixed \( x \).
2.2.2 The Galerkin Method

In this subsection, we present the Galerkin method. Suppose that we seek an approximation solution \( \tilde{f}(x) \) for (2.9)

\[
\tilde{f}(x) = \sum_{i=1}^{n} a_i \phi_i(x),
\]

(2.16)

where \( \phi_i(x) \) are linearly independent functions defined on \([a, b]\).

According to Delves and Mohamed, [19], if we require further that the set \( \{\phi_i(x)\} \) is complete in \( L^2(a, b) \), then for sufficiently large \( n \), we can choose \( a_i, \ i = 1, \ldots, n \) such that the approximation \( \tilde{f} \) is as close to \( f \) as we want.

By substituting (2.16) into (2.9), we get

\[
r(x) = g(x) - \left( \tilde{f}(x) - \int_{a}^{b} K(x, t) \tilde{f}(t) dt \right).
\]

The main aim of the Galerkin method is to seek \( a_i \) to minimize the norm of the residual function \( r(x) \). This is equivalent to

\[
\int_{a}^{b} \phi_i^*(x) r(x) dx = 0, \quad i = 1, \ldots, n.
\]

\[
\int_{a}^{b} \phi_i^*(x) \left[ g(x) - \left( \tilde{f}(x) - \int_{a}^{b} K(x, t) \tilde{f}(t) dt \right) \right] dx = 0, \quad i = 1, \ldots, n. \tag{2.17}
\]

If we substitute (2.16) into (2.17), we can get the following matrix equation,

\[
La = g, \tag{2.18}
\]

where

\[
L_{ij} = \int_{a}^{b} \phi_i^*(x) \phi_j(x) dx - \int_{a}^{b} \int_{a}^{b} \phi_i^*(x) K(x, t) \phi_j(t) dt dx
\]

\[
= \delta_{ij} - \int_{a}^{b} \int_{a}^{b} \phi_i^*(x) K(x, t) \phi_j(t) dt dx,
\]

\[
g_i = \int_{a}^{b} \phi_i^*(x) g(x) dx,
\]

and

\[
a = [a_1, \ldots, a_n]^T, \quad i, j = 1, \ldots, n.
\]
Hence, the matrix equation (2.18) can be rewritten as

\[(I - K)a = g\]  

(2.19)

where

\[K_{ij} = \int_a^b \int_a^b \phi_i^*(x)K(x, t)\phi_j(t)dtdx.\]

After introducing the idea of the Galerkin method, we discuss its error estimates. According to [19], we first write down the exact expression of \(f(x)\). We have

\[f(x) = \sum_{i=1}^{\infty} b_i \phi_i(x)\]

Then, we rewrite (2.19) in the following form

\[(I - (\bar{K} + \mathcal{E}(K)))(\bar{a} + \mathcal{E}(a)) = \bar{g} + \mathcal{E}(g)\]  

(2.20)

where \(\mathcal{E}(K)\) and \(\mathcal{E}(g)\) represent the quadrature errors. Finally, the error in the computed solution is

\[e_n(x) = f(x) - \tilde{f}(x)\]

\[= \sum_{i=1}^{n} (b_i - a_i)\phi_i(x) + \sum_{i=n+1}^{\infty} b_i \phi_i(x)\]

\[= \sum_{i=1}^{n} (b_i - \bar{a}_i)\phi_i(x) + \sum_{i=1}^{n} (\bar{a}_i - a_i)\phi_i(x) + \sum_{i=n+1}^{\infty} b_i \phi_i(x)\]

\[= e_1(x) + e_2(x) + e_3(x),\]

where \(\tilde{f}(x)\) is defined in (2.16). As we assume that \(\{\phi_i(x)\}\) is orthonormal, we can further get

\[||e_n(x)|| \leq ||e_1(x)|| + ||e_2(x)|| + ||e_3(x)||.\]  

(2.21)

Here

\[||e_1(x)||^2 = \sum_{i=1}^{n} (b_i - \bar{a}_i)^2,\]

\[||e_2(x)||^2 = \sum_{i=1}^{n} (\bar{a}_i - a_i)^2 = ||\mathcal{E}(a)||^2,\]

\[||e_3(x)||^2 = \sum_{i=n+1}^{\infty} b_i^2.\]
From the above expressions, we know that $e_1(x)$ is the error due to approximate $b_i$ with the Galerkin method, $e_2(x)$ is the error due to the quadrature rule and $e_3(x)$ is the error due to the truncation of the series representing $f(x)$. Therefore, we know that the error of the solution is related to these three factors.

In the next chapter, we will use Nyström's method to discretize the integral operators of second kind integral equations.
Chapter 3

Fast Algorithm for Second Kind Integral Equations

In this chapter, we consider solutions of Fredholm integral equations of the second kind where the kernel functions are asymptotically smooth or a product of such function with a highly oscillatory coefficient function. We present a scheme based on polynomial interpolation to approximate the discretization matrices $A$ which arise from the discretization of these integral operators. Our approximation matrix $B$ is obtained by partitioning the domain $[0, 1]^2$ into subdomains of different sizes and approximate the kernel function at each subdomain by using interpolation polynomial at the Chebyshev points. Although $B$ is dense, it can be constructed in $O(n)$ operations and requires $O(n)$ storage and the cost of matrix-vector multiplication $By$ is of order $O(n \log n)$ where $n$ is the number of quadrature points used in the discretization. We prove that for a given accuracy $\epsilon$, the degree of the interpolation polynomials is of order $O(\log \epsilon^{-1})$ in order that $\|A - B\|_F \leq \epsilon$ where $\| \cdot \|_F$ is the Frobenius norm. It follows that by using conjugate gradient type methods, the numerical solution of second kind integral equations requires $O(n \log(n/\epsilon))$ operations.

3.1 Introduction

In this chapter, we consider the fast solutions of Fredholm integral equations of the second kind:

\[ f(x) - d(x) \int_0^1 a(x, t)f(t)dt = g(x), \quad x \in [0, 1], \tag{3.1} \]

where the kernel function $a(x, t)$ is in $L^2[0, 1]^2$ and is analytic except at $x = t$, and the unknown function $f(x)$ and the right hand side function $g(x)$ are in
$L^2[0,1]$. We assume that the coefficient function $d(x)$ can be oscillatory but bounded. These equations lie between the equations with smooth kernels and those with arbitrary oscillatory kernels.

It is well-known that an integral operator with weakly singular kernel is a compact operator, see for instance [8, Theorem 2.21] and that the product of a bounded operator and a compact operator is still compact, see for instance [10, Theorem 4.18]. Therefore the integral $d(x) \int_0^1 a(x, t)f(t)dt$ in (3.1) defines a compact operator. Hence for integral equations of the second kind, they are either singular or well-conditioned. After discretization, (3.1) becomes a linear system

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

where $I$ is the identity matrix, $D$ is a diagonal matrix and $A$ is a dense matrix corresponding to the discretization of the integral in (3.1).

Various direct and iterative methods have been proposed for solving (3.2), see [5] for instance. However, one overriding drawback of these methods is the high cost of working with the associated dense matrices $A$. For problems discretized with $n$ quadrature points, 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 type methods, each iteration requires matrix-vector multiplications of the form $A\mathbf{y}$, see [12]. Therefore even for well-conditioned problems, such as the second kind equations, the methods require $O(n^2)$ operations, which for large-scale problems is often prohibitive.

In recent years, a number of fast algorithms for (3.2) have been developed, see for instance [6, 9, 3, 1, 2]. The fast multipole method proposed in [6] combines the use of low-order polynomial interpolation of the kernel functions 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 the 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 [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 are then inverted by the Schulz method. The complexity of the resulting algorithm is bounded by $O(n \log^2 n)$.

In [2], a fast multiplication scheme is proposed for doing the matrix-vector multiplications. By starting with the same approach as in [1], the matrix $A$ is written as the sum of a sequence of block matrices where the size of the blocks are increasing. Then, polynomial interpolation is used as in [1] and [6] for each of the block matrices to obtain an approximation matrix $B$. More precisely, the domain on which the block matrix is defined is first mapped into a fixed domain $[-1, 1]^2$. Then, $k^2$ entries (samples) are taken on the domain $[-1, 1]^2$. A degree $k$ interpolation polynomial is then used to interpolate the kernel function in the domain $[-1, 1]^2$. Finally, the interpolation polynomial is mapped back to the domain on which the block matrix is defined.

By mapping the domain on which the block matrix is defined into a fixed domain $[-1, 1]^2$, the basis function matrix is stable. Moreover, it can be factored out during the construction of $B$ and therefore lead to the fast multiplication scheme. However, wavelet-like bases are not used as in [1] to further approximate the matrix to get a sparse representation. By exploiting the special structure of $B$, all the entries of $B$ are kept and yet $B$ can be constructed in $O(nk)$ operations with $O(nk)$ storage requirement and that the matrix-vector multiplication $By$ can be done in $O(nk \log n)$ operations. Thus well-conditioned problems can be solved by conjugate gradient type methods in only $O(nk \log n)$ operations. Since the wavelet-like transformation is not needed as in [1], therefore the coefficient functions $d(x)$ in (3.1) need not be
positive as required by the algorithm in [1]. Also since the small entries are not needed to throw away to get a sparse matrix, the approximation is more accurate.

In this chapter, we start with the same approach as in [1] and [2]. We partition the discretization matrix $A$ into sub-blocks of different sizes. Then, a degree $k$ interpolation polynomial is used for each of the sub-blocks to obtain an approximation matrix $B$. In the approximation, the Chebyshev points and the Lagrange interpolation polynomials are chosen as interpolation points and basis functions respectively. Unlike [2], we do not map the subdomain on which the sub-block is defined into the domain $[-1,1]^2$ and we apply the Lagrange polynomial interpolation functions directly onto each subdomain. Our approximation scheme is therefore simpler than the approximation scheme in [2]. The resulting basis function matrices are still as stable as that of in [2] and can also be factored out during the construction of $B$. Since the Chebyshev points are the optimal interpolation points, see [7, pp.284-287], our approximation scheme is more accurate, see Tables 2 and 3 in §4.

By using our scheme, the approximation matrix $B$ can be constructed in $O(nk)$ operations and only require $O(nk)$ storage to represent, where $n$ is the number of quadrature points used in the discretization. By the structure of $B$, for any vector $y$, the product $By$ can also be done in only $O(nk \log n)$ operations. Therefore, as the second kind integral equation is in general a well-posed problem, the approximated system can be solved by the conjugate gradient type method in only $O(nk \log n)$ operations.

Since the Chebyshev points used in our approximation are the optimal interpolation points, we are able to analyze the accuracy of the matrix $B$ under the assumption that the kernel functions are asymptotically smooth, i.e.,

$$|\mathcal{D}^m a(x, t)| \leq \rho m^\alpha m!|x - t|^{\delta - m}, \quad \text{for } m > 0,$$

where $\alpha > 0$, $\rho > 0$, see [11]. Here $\mathcal{D}^m$ is the $m$th-order partial derivative of $x$ or
Chapter 3  Fast Algorithm for Second Kind Integral Equations

24

t or both. We will show that if the kernel function is smooth ($\delta \geq 0$), then the degree $k$ of the interpolation polynomials used in our scheme is independent of the matrix size $n$ and for $\|A - B\|_F \leq \epsilon$, it can be set to $O(\log \epsilon^{-1})$. If the kernel function is not smooth ($\delta < 0$), then in order to acquire a given accuracy, we show that $k$ increases slowly like $O(\log n)$ as the matrix size $n$ increases. In fact, we will show in §3 that $k$ is given by

$$k = O(\log \epsilon^{-1}) + \max(0, -\delta) \log n.$$  (3.3)

In [11], a method is proposed to find an approximation matrix $C$ to approximate $A$ which requires $O(n \log n \log \epsilon^{-1})$ memory in order to get $\|A - C\|_F \leq \epsilon$ for any $\epsilon > 0$. Since the storage requirement of our approximation matrix $B$ is of order $O(nk)$, according to (3.3), it becomes $O(n \log \epsilon^{-1}) + O(n \max(0, -\delta) \log n)$ in order that $\|A - B\|_F \leq \epsilon$. Thus, our storage requirement is of the same order as that in [11].

The outline of this chapter is as follows. In §2, we derive our fast multiplication scheme and give its complexity analysis. We will show that the approximation matrix $B$ can be obtained in $O(nk)$ operations and requires $O(nk)$ storage. Moreover, the cost for matrix-vector multiplication $Bx$ is just $O(nk \log n)$. In §3, we will analyze the accuracy of the approximation matrix $B$ and give a proof of (3.3). We will give the numerical results in §4 to illustrate the efficiency, accuracy and stability of our approximation scheme. Finally, concluding remarks are given in §5.

3.2 The Approximation

In this section, we present the main idea of our fast matrix-vector multiplication scheme for the discretization matrix $A$. We show in this section that by using this fast matrix-vector multiplication scheme, the cost of matrix-vector multiplication is of order $O(nk \log n)$ and the storage requirement is of order $O(nk)$. Here $k$ is the degree of the interpolation polynomial used in our
approximation scheme. The idea of the scheme is to take advantage of the smoothness of the kernel function $a(x, t)$ away from the singularity where we can use polynomials to approximate the function quite accurately. 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. As $A$ possesses the same smoothness properties of $a(x, t)$, we use low rank matrices obtained by polynomial interpolation to approximate $A$. Clearly, it is not good to approximate the whole matrix $A$ by one low-rank matrix. Therefore, we partition $A$ into block matrices with different sizes. Then, the block matrices are approximated by some low rank matrices. In other words, we partition the domain $[0, 1]^2$ on which the discretization matrix $A$ is defined into subdomains of different sizes and approximate the kernel function $a(x, t)$ on each subdomain by a degree $k$ polynomial.

3.2.1 The Partition

Our partition is the same as that in [1] and [2]. Let the size of $A$ be $n = 2^k$. Here $k$ is a fixed small integer that depends on the smoothness of $a(\cdot, \cdot)$. As remarked above, for $a(x, t) = \log |x - t|$, a small $k$ like 7 will give 6 digits of accuracy, see Table 2 in §4 for instance. Also, Theorem 2 in §3 will give an estimate of $k$ for a given accuracy $\epsilon$. We partition $A$ into blocks of different sizes as shown in Figure 1. The blocks near the diagonal will have size $k$-by-$k$, those next remote are of size $2k$-by-$2k$ and up to the largest size $2^{l-2}k$-by-$2^{l-2}k$. 
Chapter 3 Fast Algorithm for Second Kind Integral Equations

26

4^12\lambda

\int

22

\int

23

24

\int

25

26

27

28

\int

29

30

\int

31

\int

32

33

\int

34

\int

35

36

\int

37

\int

38

39

\int

40

41

42

\int

\int

\int

\int

\int

\int

\int

\int

\int

\int

Figure 1: Partition of the matrix $A$.

If we group the block matrices with the same size into one matrix, we see that $A$ can be written as the sum of a sequence of block matrices

$$
A = A^{(0)} + A^{(1)} + \ldots + A^{(l-2)}.
$$

(3.4)

Here $A^{(u)}$ contains only size $2^u k$-by-$2^u k$ block matrices where $u = 0, \ldots, l - 2$. The numbered blocks in Figure 1 give the non-zero blocks in $A^{(1)}$ for $l = 5$ and each numbered block $A^{(1,v)}$ is a $2k$-by-$2k$ matrix where $v = 1, \ldots, 42$. We can easily see from Figure 1 that the number of nonzero blocks in $A^{(u)}$ is given by

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

(3.5)

We will denote these nonzero blocks by $A^{(u,v)}$, $v = 1, \ldots, v_u$.

To carry out the error analysis of the approximation matrix $B$ in §3, we need to partition the domain $[0, 1]^2$ on which our discretization matrix $A$ is
defined. Since we have defined the partition of $A$, the corresponding partition of the domain $[0, 1]^2$ is defined similarly in Figure 2.

As each of the nonzero blocks in $A^{(u)}$ is called $A^{(u,v)}$, we call the subdomain on which the matrix $A^{(u,v)}$ is defined a $(u, v)$-subdomain. As an illustration, for $l = 5$, the numbered subdomains in Figure 2 are the $(1, v)$-subdomains where $v = 1, \ldots, 42$.

We know that all the block matrices $A^{(u,v)}$ in $A^{(u)}$ are of the same size. Therefore, the lengths of the edge of the corresponding $(u, v)$-subdomains are also the same and are given by

$$d_u = \frac{1}{2^{l-u}},$$

where $u = 0, \ldots, l - 2$. From Figure 2, it is also easy to observe that all the
points \((x, t)\) in the \((1, 1)\)-subdomain satisfy
\[
|x - t| \geq |2d_1 - d_1| = d_1.
\]
Similarly, it is not difficult to see that each point \((x, t)\) in any \((u, v)\)-subdomain satisfies
\[
|x - t| \geq d_u. \tag{3.7}
\]

The approximation matrix \(B\) of \(A\) is constructed by approximating each block \(A^{(u,v)}\) in \(A^{(u)}\) by a rank \(k\) matrix \(B^{(u,v)}\). Let the entries of \(A^{(u,v)}\) be defined as
\[
[A^{(u,v)}]_{i,j} = \frac{1}{n-1} a(x_0^{(u,v)} + (i-1)h, t_0^{(u,v)} + (j-1)h), \quad 1 \leq i, j \leq 2^u k, \tag{3.8}
\]
where \(h = \frac{1}{n-1}\). (For simplicity, we use the Nyström method with uniformly-spaced points here to discretize (3.1) but we can see later that if other quadrature rules are used, our approximation scheme still works.) The matrix \(B^{(u,v)}\) is obtained by taking \(k^2\) entries (samples) in the \((u, v)\)-subdomain at the Chebyshev points and then interpolating them by Lagrange polynomials to get the \(2^u k\)-by-\(2^u k\) matrix. The advantage for this approach is that the basis function matrix \(L^{(u)}\) is the same for all \(v\). More precisely, we will see in the next subsection that \(L^{(u)}\) depends only on the size of the matrix \(A^{(u,v)}\). Therefore, as the size for all \(A^{(u,v)}\) is the same for same \(u\), \(L^{(u)}\) can be factored out and hence lead to our fast multiplication scheme.

### 3.2.2 The Sampling and the Interpolation

We now consider how to construct \(B^{(u,v)}\). We will generate the entries of \(B^{(u,v)}\) by calculating the Lagrange interpolation polynomials of \(a(x, t)\) at the Chebyshev points on the \((u, v)\)-subdomains. We define the required Chebyshev points as follows:
\[
\begin{cases}
  x_r^{(u,v)} = x_0^{(u,v)} + \frac{(2^u k - 1)h}{2} (1 + c_r), \\
  t_s^{(u,v)} = t_0^{(u,v)} + \frac{(2^u k - 1)h}{2} (1 + c_s),
\end{cases} \tag{3.9}
\]
where

$$c_r = \cos \left( \frac{(2r - 1)\pi}{2k} \right), \quad r = 1, \ldots, k$$  \hspace{1cm} (3.10)

are roots of the degree $k$ Chebyshev polynomial on the interval $[-1, 1]$.

Using the Lagrange polynomials as basis functions, for $x_0^{(u,v)} \leq x \leq x_0^{(u,v)} + (2^k - 1)h$, $t_0^{(u,v)} \leq t \leq t_0^{(u,v)} + (2^k - 1)h$ and by using the Chebyshev points $c_r$ defined in (3.10), we have

$$a(x, t) \approx \sum_{r=1}^{k} \sum_{s=1}^{k} a(x_r^{(u,v)}, t_s^{(u,v)}) p_r(x, x_0^{(u,v)}) p_s(t, t_0^{(u,v)}),$$  \hspace{1cm} (3.11)

where the Lagrange polynomial $p_r(x, x_0^{(u,v)})$ is defined as

$$p_r(x, x_0^{(u,v)}) = \prod_{s=1, s \neq r}^{k} \frac{(x - x_s^{(u,v)})}{(x_r^{(u,v)} - x_s^{(u,v)})} = \prod_{s=1, s \neq r}^{k} \frac{(x - x_0^{(u,v)} - (2^k - 1)h) (1 + c_s)}{(2^k - 1)h (c_r - c_s)}$$

$$= \prod_{s=1, s \neq r}^{k} \left( -1 + \frac{2(x - x_0^{(u,v)})}{(2^k - 1)h} - c_s \right) = \omega_r(-1 + \frac{2(x - x_0^{(u,v)})}{(2^k - 1)h}),$$  \hspace{1cm} (3.12)

the Lagrange polynomial $p_s(t, t_0^{(u,v)})$ is defined similarly as $p_r(x, x_0^{(u,v)})$ and

$$\omega_r(x) = \prod_{s=1, s \neq r}^{k} (x - c_s).$$  \hspace{1cm} (3.13)

Combining (3.8), (3.11) and (3.12), we get the approximation of $A^{(u,v)}$ as follows

$$[A^{(u,v)}]_{i,j} = \frac{1}{n - 1} a(x_0^{(u,v)} + (i - 1)h, t_0^{(u,v)} + (j - 1)h)$$

$$\approx \frac{1}{n - 1} \sum_{r=1}^{k} \sum_{s=1}^{k} a(x_r^{(u,v)}, t_s^{(u,v)}) p_r(x_0^{(u,v)} + (i - 1)h, x_0^{(u,v)}) p_s(t_0^{(u,v)} + (j - 1)h, t_0^{(u,v)})$$

$$= [B^{(u,v)}]_{i,j}, \quad 1 \leq i, j \leq 2^k.$$  \hspace{1cm} (3.14)

In matrix terms, the approximation matrix $B^{(u,v)}$ of $A^{(u,v)}$ is given by

$$B^{(u,v)} = L^{(u)} S^{(u,v)} (L^{(u)})^T,$$
where the $k$-by-$k$ matrix $S^{(u,v)}$ and the $2^u k$-by-$k$ matrix $L^{(u)}$ are defined by

$$
[S^{(u,v)}]_{r,s} = \frac{1}{n-1} a(x^{(u,v)}_r, t^{(u,v)}_s), \quad 1 \leq r, s \leq k \tag{3.15}
$$

and

$$
[L^{(u)}]_{i,j} = p_j(x_0^{(u,v)} + (i-1)h, x_0^{(u,v)}) = \frac{\omega_j(-1 + \frac{2(i-1)}{(2^u k-1)h})}{\omega_j(c_j)} \tag{3.16}
$$

where $1 \leq i \leq 2^u k$ and $1 \leq j \leq k$.

To conclude, we see that we apply the Lagrange interpolation functions directly on each $(u,v)$-subdomain and use the Chebyshev points as our interpolation points. In [2], the $(u,v)$-subdomains are first mapped to $[-1,1]^2$ and the uniformly-spaced points are used as the interpolation points. Thus, our approximation scheme is simpler than the approximation scheme in [2]. Besides, our scheme is more accurate as the Chebyshev points are optimal interpolation points, see [7, pp.284-287]. The results shown in Tables 2 and 3 in §4 also illustrate that our scheme is more accurate.

Here, we emphasize that we only need a formula $a(\cdot, \cdot)$ of the kernel function and the $2^u k$-by-$2^u k$ matrix $A^{(u,v)}$ is not needed to form during the construction of the approximation matrix $B^{(u,v)}$. We see from the derivation that any quadrature rules can be used to discretize $a(\cdot, \cdot)$ in order to get the sample matrix $S^{(u,v)}$ in (3.15). Furthermore, we also have no need to form the $2^u k$-by-$2^u k$ matrix $B^{(u,v)}$ explicitly. By (3.14), only storing the factors $L^{(u)}$ and $S^{(u,v)}$ of $B^{(u,v)}$ is enough for us to carry out the matrix-vector multiplications.

### 3.2.3 The Approximation Matrix $B$

From the above discussion, we know that the matrix $L^{(u)}$ is independent of the index $v$. Our fast multiplication scheme is based on this fact. By this property, we observe from Figure 1 and (3.14) that the approximation matrix $B^{(u)}$ is of the form

$$
B^{(u)} = [I_{2^u} \otimes L^{(u)}] \cdot S^{(u)} \cdot [I_{2^u} \otimes (L^{(u)})^T], \quad u = 1, \ldots, l-2, \tag{3.17}
$$
where $I_{2^l \times 2^l}$ is the identity matrix of size $2^l \times 2^l$, $\otimes$ is the Kronecker tensor product. Here the matrix $S^{(u)}$ has the same block structure as $A^{(u)}$ but each block in $S^{(u)}$ is of size $k$-by-$k$. For example, the matrix $B^{(1)}$ for $l = 5$ has the following structure,

\[
B^{(1)} = \left[ I_{16} \otimes \left( L^{(1)} \right) \right],
\]

where each numbered block $S^{(l,v)}$ is a $k$-by-$k$ matrix for $v = 1, \ldots, 42$.

After we have defined the approximation matrix $B^{(u)}$ for each $A^{(u)}$, $u = 1, \ldots, l - 2$, we can get the approximation matrix $B$ of $A$ by,

\[
B = A^{(0)} + B^{(1)} + B^{(2)} + \ldots + B^{(l-2)}. \tag{3.19}
\]

From (3.17) and (3.19), we see that the approximation matrix $B$ is not needed to form explicitly in order to perform the matrix-vector multiplication. We only need $A^{(0)}$ and the factors $S^{(u,v)}$ and $L^{(u)}$. In the next subsection, we study the storage requirement and the cost of matrix-vector multiplications $B \mathbf{y}$ by using the structure (3.17) and (3.19).

### 3.2.4 Complexity Analysis

In this subsection, we show that the approximation matrix $B$ can be built up by our approximation scheme in $O(nk)$ operations and requires only $O(nk)$
storage. Here \( n \) is the number of quadrature points used in the discretization and \( k \) is the degree of the interpolation polynomial. Moreover, by using (3.17) and (3.19), the product \( By \) can be done in \( O(nk \log n) \) operations. In the following complexity analysis, we only count the number of multiplications in the operation counts as the number of additions is of the same order.

**Theorem 6** Let \( B \) be defined as in (3.17) and (3.19), where \( A^{(0)}, L^{(u)} \) and \( S^{(u)} \) are defined in (3.4), (3.16) and (3.15) respectively. We have

(i) The storage requirement for representing \( B \) is less than \( 10nk \).

(ii) \( A^{(0)}, L^{(u)} \) and \( S^{(u)} \) can be constructed in \( 7nk \) multiplications and \( 9nk \) function evaluations.

(iii) For any vector \( y \), the product \( By \) can be obtained in \( (2 \log n + 5)nk \) operations.

**Proof:**  
(i) By using (3.17) and (3.19) to represent \( B \), we only need to store \( A^{(0)}, L^{(u)} \) and \( S^{(u)} \), \( u = 1, \ldots, l-2 \). Their storage requirements are summarized in Table 1.

Thus the total storage requirement is

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

(ii) From (3.5), we see that \( A^{(0)} \) consists of \( 6 \cdot 2^l - 8 \) blocks and each of them is of size \( k \)-by-\( k \). Therefore, we need \( 6 \cdot 2^l - 8 \) \( k^2 \) function evaluations to construct \( A^{(0)} \).

Before we start to construct \( S^{(u)} \) and \( L^{(u)} \), from (3.9), (3.15) and (3.16), we know that it is required to form the \( k \) Chebyshev points in order to construct \( S^{(u)} \) and \( L^{(u)} \). Therefore, we use \( k \) function evaluations to evaluate the \( k \) Chebyshev points \( c_r \) only once, see (3.10). After storing the points \( c_r \), we can use them in the construction of \( S^{(u)} \) and \( L^{(u)} \).
Chapter 3  Fast Algorithm for Second Kind Integral Equations

Forming | Storage | Explanation
--- | --- | ---
$A^{(0)}$ | $(6 \cdot 2^l - 8)k^2$ | $A^{(0)}$ consists of $(6 \cdot 2^l - 8)$ blocks and each of them is of size $k$-by-$k$, see (3.5).

$S^{(u)}$ | $6(2^{l-1-u} - 1)k^2$ | $S^{(u)}$ is a block matrix with $6(2^{l-1-u} - 1)$ nonzero blocks and each nonzero block $S^{(u,v)}$ is of size $k$-by-$k$, see (3.5) and (3.15).

$L^{(u)}$ | $2^u k^2$ | $L^{(u)}$ is a $2^u k$-by-$k$ matrix, see (3.16).

Table 1. Storage requirement.

From (3.15), we know that we need to form the mesh points defined in (3.9) before we can evaluate the $k^2$ entries of $S^{(u,v)}$. Let $h' = (2^u k - 1)h/2$, then each of the mesh points in (3.9) requires one multiplication. Therefore, it requires $2 \cdot k$ multiplications and $k^2$ function evaluations to construct $S^{(u,v)}$. Thus, the construction of $S^{(u)}$ requires $2 \cdot 6(2^{l-1-u} - 1)k$ multiplications and $6(2^{l-1-u} - 1)k^2$ function evaluations as there are $6(2^{l-1-u} - 1)$ nonzero blocks $S^{(u,v)}$ in $S^{(u)}$, see (3.5).

Next, we discuss the cost of the construction of $L^{(u)}$, cf. (3.13) and (3.16). It can be done by the following steps.

(a) From (3.16), we see that we need to evaluate the uniformly-spaced points

$$x_i = -1 + \frac{2(i - 1)}{2^u k - 1}, \quad i = 1, \ldots, 2^u k$$

and $c_j$ defined in (3.10). Let $h'' = 2/(2^u k - 1)$, then $x_{i+1} = x_i + h''$. Therefore, we need only one multiplication and one division to get all $x_i$ for a fixed $u$. We emphasize again that we need not form $c_j$ again as we have already formed them before we construct $L^{(u)}$. 

(b) We then calculate the denominator $\omega_j(c_j)$ in (3.16). Evaluating each $\omega_j(c_j)$ requires $k-2$ multiplications as there are $k-1$ brackets in each $\omega_j(c_j)$, see (3.13). Since we need to form $\omega_j(c_j)$ for $1 \leq j \leq k$, totally $k(k-2)$ operations are required.

(c) Now, we discuss how to construct the numerators of $L^{(u)}$ in (3.16). From (3.13) and (3.16), we know that the numerators in the first column of $L^{(u)}$ consists only $\omega_1(x_i)$ where $x_i$, $i = 1, \ldots, 2^u k$ are the uniformly-spaced points formed in (a). Since each $\omega_1(x_i)$ requires $(k-2)$ operations, the numerators in the first column can be computed in $2^u k (k-2)$ operations. For the numerators in the other columns, we observe that they are different from those in the previous column by a fraction. For example, we see from (3.13) that $\omega_{j+1}(x_i) = \omega_j(x_i) \left( \frac{x_i - c_j}{x_i - c_{j+1}} \right)$. Therefore, we can construct them recursively by multiplying a bracket to those in the previous column and then dividing them by a bracket. Therefore, each column other than the first column requires $2 \cdot 2^u k$ operations. Thus, we need $2^u k (k-1) \cdot 2$ operations to form the remaining $k-1$ columns.

(d) Finally, we can obtain the $(i,j)$th-entry of $L^{(u)}$ by dividing the numerators with the denominators, see (b) and (c). The cost is $2^u k^2$ operations. Therefore, the construction of $L^{(u)}$ requires $k(k-2) + 2^u (4k^2 - 4k) + 2$ multiplications.

Thus, we see that the total number of multiplications required to form $S^{(u)}$ and $L^{(u)}$ for all $u$ is

$$\sum_{u=1}^{l-2} \left\{ 12(2^{l-1-u} - 1)k + k(k-2) + 2^u (4k^2 - 4k) + 2 \right\} < 6 \cdot 2^l k + 4 \cdot 2^l k^2 < 7nk$$

and the total number of function evaluations required to form $S^{(u)}$ and $L^{(u)}$ for all $u$ is

$$(6 \cdot 2^l - 8) k^2 + k + \left\{ \sum_{u=1}^{l-2} 6(2^{l-1-u} - 1)k^2 \right\} < 9 \cdot 2^l k^2 = 9nk.$$
(iii) By (3.17) and (3.19), we have

$$By = A^{(0)}y + \sum_{u=1}^{l-2} \left[ I_{2l-u} \otimes L^{(u)} \right] \cdot S^{(u)} \cdot \left[ I_{2l-u} \otimes (L^{(u)})^T \right] y. \quad (3.20)$$

By using the tensor structure in (3.20), $\left[ I_{2l-u} \otimes (L^{(u)})^T \right] y$ can be obtained in $2^l k^2$ operations. Since there are $6(2^{l-1}-1)$ sub-blocks of size $k$-by-$k$ in $S^{(u)}$ (see (3.5) and (3.18)), it can be easily checked that the total number of multiplications required to form $By$ is

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

Numerical experiments in §4 will illustrate that our approximation matrices $B$ are accurate and stable and our multiplication scheme indeed attains the said complexity. Recall that Fredholm integral equations of the second kind are usually well-conditioned and hence can be solved by conjugate gradient type methods without preconditioning. The main cost in each iteration is the matrix-vector multiplication, see [12]. Thus using our scheme, the equations can be solved in $O(nk \log n)$ complexity.

In the next section, we will give the error analysis of our approximation scheme. We will prove that in order that $\|A - B\|_F \leq \epsilon$, the degree of the interpolation polynomial is given by (3.3).

### 3.3 Error Analysis

In this section, we present the error bound analysis of our approximation scheme. In the following, we use $D^m$ to denote the $m$th-order partial derivative of variable $x$ or $t$ or both. We also use the symbols $D^m_x$ and $D^m_t$ for partial derivative on the single variable $x$ and $t$ respectively. We assume that the kernel function $a(x,t)$ is asymptotically smooth, i.e.,

$$|D^m a(x,t)| \leq pm^m m! |x - t|^{d-m}, \quad (3.21)$$
where \( \alpha > 0 \), \( \rho > 0 \) and \( \delta \) are constants, see \([11]\) for instance. This assumption is quite general. For examples, for \( a(x, t) = \sqrt{|x - t|} \), we have \( |D^m a(x, t)| < 0.5m!|x - t|^{0.5-m} \) (i.e. \( \delta = 0.5 \)) in this case. For \( a(x, t) = \log |x - t| \), we have \( |D^m a(x, t)| = m!|x - t|^{-m} \). In this case, we have \( \delta = 0 \). For \( a(x, t) = 1/\sqrt{|x - t|} \), we have \( |D^m a(x, t)| < 0.5m!|x - t|^{-0.5-m} \). Therefore, we have \( \delta = -0.5 \) in this case. In fact, a sufficiently smooth kernel function will have \( \delta \geq 0 \) and a kernel function which is not smooth enough will have \( \delta < 0 \).

We prove that for any given tolerance \( \epsilon \), in order that \( ||A - B||_F \leq \epsilon \), the degree of interpolation polynomials used in our approximation scheme is given by \( k = O(\log \epsilon^{-1}) + \max(0, -\delta) \log n \). By this result, we see that for a sufficiently smooth kernel \( (\delta \geq 0) \) and for a given accuracy, we can use interpolation polynomials of a fixed degree \( k \) to get the required accuracy for all matrix sizes. If the kernel function is not smooth \( (\delta < 0) \), then the degree \( k \) of interpolation polynomials will increase like \( O(\log n) \) as the matrix size \( n \) increases in order to get the required accuracy.

Before we start, we recall that if \( f(x) \) is a function in \( C^m[\beta_1, \beta_2] \) and \( q(x) \) is interpolation polynomial at the Chebyshev points, then the error bound is

\[
|f(x) - q(x)| \leq \frac{1}{2^{m-1}m!} \sup_{\beta_1 \leq x \leq \beta_2} |f^{(m)}(x)| \left( \frac{\beta_2 - \beta_1}{2} \right)^m,
\]

see \([7, \text{pp.284-287}]\) for instance. Here the Chebyshev points in the interval \([\beta_1, \beta_2]\) are given by \( x_1, \ldots, x_k \), where

\[
x_i = \frac{\beta_1 + \beta_2}{2} + \frac{\beta_2 - \beta_1}{2} c_i
\]

and \( c_i \) are the roots of the degree \( k \) Chebyshev polynomial in the interval \([-1, 1]\) and are given by (3.10).

For 2-dimensional interpolation polynomials on the Chebyshev points, we have the following result.

**Lemma 1** Let the function \( a(x, t) \) be a function such that \( D_x D_t^m a(x, t) \) is continuous in the domain \([\beta_1, \beta_2] \times [\gamma_1, \gamma_2] \). Let \( q(x, t) \) be the interpolation polynomial on the Chebyshev points, i.e., the mesh points \((x_i, t_j) \) \((i, j = 1, \ldots, k)\)
be defined as
\[
x_i = \frac{\beta_1 + \beta_2}{2} + \frac{\beta_2 - \beta_1}{2} \cos \left( \frac{(2i-1)\pi}{2k} \right), \quad i = 1, 2, \ldots, k
\]
and
\[
t_j = \frac{\gamma_1 + \gamma_2}{2} + \frac{\gamma_2 - \gamma_1}{2} \cos \left( \frac{(2j-1)\pi}{2k} \right), \quad j = 1, 2, \ldots, k.
\]
Then
\[
||a - q|| \leq \frac{1}{2^{m-1}m!} \left( ||D^m_t a|| \left( \frac{\gamma_2 - \gamma_1}{2} \right)^m + ||D^m x a|| \left( \frac{\beta_2 - \beta_1}{2} \right)^m \right)
\]
\[
+ \frac{||D^m_t D^m x a|| (\beta_2 - \beta_1)^m \left( \frac{\gamma_2 - \gamma_1}{2} \right)^m}{(2^{m-1}m!)^2},
\]
where \(|| \cdot ||\) is the supremum norm in \([\beta_1, \beta_2] \times [\gamma_1, \gamma_2]\).

**Proof:** First of all, we define the 1-variable interpolation polynomial for a 2-dimensional function as
\[
\mathcal{I}_x a(x, t) = \sum_{j=1}^{k} a(x, t_j) p_j(t)
\]
and
\[
\mathcal{I}_t a(x, t) = \sum_{i=1}^{k} a(x_i, t) p_i(x),
\]
where \(p_i(x)\) is the Lagrange polynomial defined in \([\gamma_1, \gamma_2]\) and \(p_j(t)\) is the Lagrange polynomial defined in \([\beta_1, \beta_2]\). Obviously, the Lagrange interpolation polynomial \(q(x, t)\) of \(a(x, t)\) on the mesh \((x_i, t_j) (i, j = 1, \ldots, k)\) is given by
\[
q(x, t) = \mathcal{I}_{xt} a(x, t) = \sum_{i=1}^{k} \sum_{j=1}^{k} a(x_i, t_j) p_i(x) p_j(t).
\]
It is not difficult to see that
\[
\mathcal{I}_{xt} a = \mathcal{I}_t (\mathcal{I}_x a) = \mathcal{I}_x (\mathcal{I}_t a).
\]

By the triangular inequality
\[
||a - q|| \leq ||a - \mathcal{I}_x a|| + ||\mathcal{I}_x a - \mathcal{I}_{xt} a||,
\]
we only require to estimate the error bounds for \(||a - \mathcal{I}_x a||\) and \(||\mathcal{I}_x a - \mathcal{I}_{xt} a||\). By applying (3.22) and (3.24), we get
\[
||a(x, t) - \mathcal{I}_x a(x, t)|| \leq \frac{||D^m_t a||}{2^{m-1}m!} \left( \frac{\gamma_2 - \gamma_1}{2} \right)^m.
\]
Therefore, 
\[ \|a - I_x a\| \leq \left\| D_x^m a \right\| \left( \frac{\gamma_2 - \gamma_1}{2} \right)^m. \] (3.27)

Similarly, we have 
\[ \|I_x a - I_t (I_x a)\| = \|I_x a - I_t (I_x a)\| \leq \left\| D_x^m (I_x a) \right\| \left( \frac{\beta_2 - \beta_1}{2} \right)^m. \] (3.28)

From (3.24) we see that \( D_x^m I_x a = I_x D_x^m a \) and by (3.27) we get 
\[ \left\| D_x^m a - I_x D_x^m a \right\| \leq \left\| D_x^m D_x^m a \right\| \left( \frac{\gamma_2 - \gamma_1}{2} \right)^m. \] Therefore 
\[ \|D_x^m I_x a\| \leq \left\| D_x^m a \right\| + \|D_x^m D_x^m a\| \left( \frac{\gamma_2 - \gamma_1}{2} \right)^m. \] (3.29)

By (3.27), (3.28) and (3.29), we finally get 
\[ \|a - q\| \leq \frac{\|D_x^m a\|}{2^{m-1}m!} \left( \frac{\gamma_2 - \gamma_1}{2} \right)^m 
+ \frac{1}{2^{m-1}m!} \left( \frac{\beta_2 - \beta_1}{2} \right)^m \left\{ \|D_x^m a\| + \|D_x^m D_x^m a\| \left( \frac{\gamma_2 - \gamma_1}{2} \right)^m \right\} 
\leq \frac{1}{2^{m-1}m!} \left( \|D_x^m a\| \left( \frac{\gamma_2 - \gamma_1}{2} \right)^m + \|D_x^m a\| \left( \frac{\beta_2 - \beta_1}{2} \right)^m \right) 
+ \frac{1}{2^{m-1}m!} \left( \|D_x^m D_x^m a\| \left( \frac{\beta_2 - \beta_1}{2} \right)^m \left( \frac{\gamma_2 - \gamma_1}{2} \right)^m \right). \] (3.30)

After getting the above general result, we return our discussion to the accuracy of the approximation matrix \( B \). Let the kernel function \( a(x, t) \) satisfy assumption (3.21). By (3.6) and (3.7), we note that for any \((u, v)\)-subdomain defined by \([\beta_1, \beta_2] \times [\gamma_1, \gamma_2] \), then \( \beta_2 - \beta_1 = \gamma_2 - \gamma_1 = d_u \) and \( |x - t| \geq d_u \). Therefore, by (3.21) and (3.30), we get 
\[ \|a - q^{(u,v)}\| \leq \frac{\rho m^m (d_u)^{(5 - m)}}{2^{m-2}m!} \left( \frac{d_u}{2} \right)^m + \frac{\rho (2m)^{m} (d_u)^{(2m)}}{(2^{m-1}m!)^2} \left( \frac{d_u}{2} \right)^{2m}, \]
where \( q^{(u,v)} \) is the Lagrange interpolation polynomial of \( a(x, t) \) at the Chebyshev points on the \((u, v)\)-subdomain (cf. (3.11)). Noting that \( (\frac{2m}{m2^m})^2 \leq \frac{1}{2} \)
(which can easily be proved by mathematical induction), we have

\[ ||a - q^{(u,v)}|| \leq (d_u)^\delta \left( \frac{2\rho m^\alpha}{2^{2m-1}} + \frac{\rho(2m)^\alpha}{2^{2m+1}} \right) \leq (d_u)^\delta \frac{5\rho(2m)^\alpha}{2^{2m}}. \]  

(3.31)

By the estimation (3.31), we have the following main result.

**Theorem 7** Assume that the kernel function \( a(x,t) \) satisfies assumption (3.21). Then, for any \( \epsilon > 0 \), there exists an integer

\[ k = O(\log \epsilon^{-1}) + \max(0, -\delta) \log n, \]  

(3.32)

such that for any \( m \geq k \), we have

\[ ||A - B||_F \leq \epsilon. \]

**Proof:** We first prove that for \( k \) defined in (3.32), \( ||a - q^{(u,v)}|| \leq \epsilon \) for each \((u,v)\)-subdomain. We consider the case \( \delta \geq 0 \) first. From (3.31) and noting that \( d_u < 1 \), see (3.6), we get

\[ ||a - q^{(u,v)}|| \leq \frac{5\rho(2m)^\alpha}{2^{2m}} \]

in each \((u,v)\)-subdomain. In order that \( \frac{5\rho(2m)^\alpha}{2^{2m}} \leq \epsilon \), we need \( \log_2(5\rho) + \alpha(1 + \log_2 m) - 2m \leq \log_2 \epsilon \), i.e.,

\[ 2m - \alpha \log_2 m \geq \alpha + \log_2 \left( \frac{5\rho}{\epsilon} \right). \]

Let \( k_0 = \min\{m|m > \alpha \log_2 m \text{ and } m \geq \alpha/\log 2\} \). Since the function \( x - \alpha \log_2 x \) is increasing for \( x \geq \alpha/\log 2 \), it is easy to see that for \( m \geq k_0 \), we have \( m > \alpha \log_2 m \). Therefore, for

\[ m \geq k = \max \left( k_0, \alpha + \log_2 \left( \frac{5\rho}{\epsilon} \right) \right), \]  

(3.33)

we have \( ||a - q^{(u,v)}|| \leq \epsilon \).

For \( \delta < 0 \), noting that \( d_u = 1/2^{l-u} \geq 2^{-l+1} \), we see from (3.31) that

\[ ||a - q^{(u,v)}|| \leq 2^{-\delta(l-1)} \frac{5\rho(2m)^\alpha}{2^{2m}}. \]
Therefore \(|a - q^{(u,v)}| \leq \epsilon\) provided that

\[
m \geq \max(k_0, \alpha + \log_2 \left( \frac{5\beta}{\epsilon} \right)) + (-\delta)(l - 1)
\]

(c.f. (3.33)). From (3.34) and noting that \(n = 2^l k\) and \(l \leq \log_2 n\), (3.32) follows.

Now, we show that \(\|A - B\|_F \leq \epsilon\). Since \(|a - q^{(u,v)}| \leq \epsilon\) in each \((u,v)\)-subdomain, we know that the difference between the entries in the corresponding block matrices \(A^{(u,v)}\) and \(B^{(u,v)}\) is smaller than \(\epsilon\), i.e.

\[
|A^{(u,v)}_{r,s} - B^{(u,v)}_{r,s}| \leq \frac{1}{n - 1}|a - q^{(u,v)}| \leq \frac{\epsilon}{n - 1}, 1 \leq r, s \leq 2^u k.
\]

As \(\|A - B\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |[A]_{i,j} - [B]_{i,j}|^2}\), the result follows. \(\square\)

By the above theorem, for a sufficiently smooth kernel \((\delta \geq 0)\) and a given accuracy, we can choose one \(k\) that satisfies (3.32) for all matrix sizes. If the kernel is not smooth \((\delta < 0)\), then \(k\) will increase like \(O(\log n)\) as the matrix size \(n\) increases in order to get the required accuracy. We have already shown in Theorem 1 that the storage requirement for the approximation matrix \(B\) is less than \(10nk\). By Theorem 2, for \(\|A - B\|_F \leq \epsilon\), the memory requirement for storing \(B\) is of order \(O(n \log \epsilon^{-1}) + O(n \max(0, -\delta) \log n)\). We recall that the approximation matrix \(C\) proposed in [11] requires \(O(n \log n \log \epsilon^{-1})\) memory in order that \(\|A - C\|_F \leq \epsilon\), see [11]. Thus, the storage requirement of our scheme is of the same order as that in [11]. In §4, we will illustrate the theoretical results by numerical experiments, see Figure 3 in §4.

**3.4 Numerical Examples**

In this section, we consider numerical solutions of Fredholm integral equations of the second kind. Since the second kind integral equations are well-conditioned, we can solve them by using conjugate gradient type methods, see [12]. The main computational task in each iteration is the matrix-vector
Chapter 3  Fast Algorithm for Second Kind Integral Equations

multiplication of the form $Ay$. As the discretization matrix $A$ is dense, the cost is $O(n^2)$. We will use the fast multiplication scheme developed in §2 to overcome this difficulty. We will also illustrate the accuracy and stability of our approximation matrices by applying our scheme to examples considered in [1] and [2].

As in [1], we discretize (3.1) by the following formulae

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

and

$$[D]_{i,i} = d\left(\frac{i-1}{n-1}\right), \quad i = 1, \ldots, n$$

to obtain the matrix equation (3.2). The formulae correspond to a primitive, trapezoidal-like quadrature discretization of the integral equation. Since the matrix $A$ is dense, we can replace it by our approximation matrix $B$ in (3.19).

In the following, we test the accuracy, stability and efficiency of $B$ as an approximation to $A$.

We test our algorithm for the following seven kernel functions:

(i) $\log|x-t|$,  
(ii) $\cos(x^2) \log|x-t|$,  
(iii) $\cos(x^2)|x-t|^{-1/2}$,  
(iv) $\cos(x^2)|x-t|^{1/2}$,  
(v) $(1 + \frac{1}{2}\sin(100x)) \log|x-t|$,  
(vi) $\sin(100x) \log|x-t|$, and  
(vii) $\cos(1000|x-t|) \log|x-t|$.

Kernel functions (i) to (vi) are examples tested in [1] and [2]. For (i) to (iv), the coefficient function $d(x) \equiv 1$ and hence the matrix $D$ in (3.2) is the
identity matrix. For (v) and (vi), $d(x)$ are highly oscillatory but bounded. The discretization equation of kernel (vii) is given by

$$(I - Re(D_1 A D_2))f = g,$$

where $D_1$ and $D_2$ are diagonal matrices given by $[D_1]_{i,i} = e^{1000 \frac{i-1}{n-1} \sqrt{-1}}$ and $[D_2]_{i,i} = e^{-1000 \frac{i-1}{n-1} \sqrt{-1}}$, $i = 1, \ldots, n$, respectively. All the numerical results are computed by MATLAB on a Sun Enterprise 4000 workstation.

(i) Accuracy of the Approximation Matrix $B$:

We measure the accuracy of the approximation matrix $B$ by computing the relative error $\|A - B\|_F/\|A\|_F$, where $\| \cdot \|_F$ is the Frobenius norm. Table 2 shows the results for different $k$ and $l$. We recall that the size of the matrices is $n = 2^k$. Thus, the largest size of matrix under test is $2^{10} \cdot 16$ by $2^{10} \cdot 16$, i.e. 16384 by 16384. Note that kernel functions (v), (vi) and (vii) give the same dense approximation matrix $B$ as that of (i) as the $a(x,t)$ for all four kernels are all equal to $\log |x-t|$. Therefore, results for kernel functions (v) to (vii) are omitted in Table 2. We observe from Table 2 that for a fix $k$, the relative error of $B$ increases very slowly as the matrix size, or $l$, increases, see also Figure 3 below. Table 3 shows the numerical results for kernel functions (i) to (iv) given in [2]. By comparing Tables 2 and 3, we see that our scheme gets more accurate numerical results than that in [2]. It is because we use the Chebyshev points rather than use the uniformly-spaced points as our interpolation points.
<table>
<thead>
<tr>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 12$</th>
<th>$k = 14$</th>
<th>$k = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l$</td>
<td>$a(x, t) = \log</td>
<td>x - t</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5.79e-05</td>
<td>1.60e-08</td>
<td>7.21e-12</td>
<td>1.66e-13</td>
</tr>
<tr>
<td>6</td>
<td>7.42e-05</td>
<td>2.15e-08</td>
<td>9.92e-12</td>
<td>2.30e-13</td>
</tr>
<tr>
<td>8</td>
<td>8.14e-05</td>
<td>2.38e-08</td>
<td>1.11e-11</td>
<td>2.58e-13</td>
</tr>
<tr>
<td>10</td>
<td>8.42e-05</td>
<td>2.47e-08</td>
<td>1.15e-11</td>
<td>2.68e-13</td>
</tr>
</tbody>
</table>

| $l$     | $a(x, t) = \cos(xt^2) \log |x - t|$ |
| 4       | 5.72e-05 | 1.62e-08 | 7.33e-12 | 1.69e-13 | 4.06e-15 |
| 6       | 7.37e-05 | 2.17e-08 | 1.01e-11 | 2.33e-13 | 5.61e-15 |
| 8       | 8.09e-05 | 2.40e-08 | 1.12e-11 | 2.61e-13 | 6.28e-15 |
| 10      | 8.37e-05 | 2.49e-08 | 1.16e-11 | 2.71e-13 | 6.57e-15 |

| $l$ | $a(x, t) = \cos(xt^2)|x - t|^{-1/2}$ |
| 4   | 7.16e-05 | 2.83e-08 | 1.54e-11 | 3.83e-13 | 9.79e-15 |
| 6   | 1.07e-04 | 4.36e-08 | 2.42e-11 | 6.05e-13 | 1.55e-14 |
| 8   | 1.29e-04 | 5.35e-08 | 3.01e-11 | 7.51e-13 | 1.92e-14 |
| 10  | 1.44e-04 | 6.01e-08 | 3.39e-11 | 8.50e-13 | 2.19e-14 |

| $l$ | $a(x, t) = \cos(xt^2)|x - t|^{1/2}$ |
| 4   | 1.52e-05 | 2.82e-09 | 1.03e-12 | 2.21e-14 | 5.76e-16 |
| 6   | 1.89e-05 | 3.55e-09 | 1.32e-12 | 2.83e-14 | 7.03e-16 |
| 8   | 2.02e-05 | 3.79e-09 | 1.41e-12 | 3.03e-14 | 7.50e-16 |
| 10  | 2.05e-05 | 3.86e-09 | 1.44e-12 | 3.09e-14 | 7.60e-16 |

Table 2: $\|A - B\|_F/\|A\|_F$ for different kernels.
### Table 3: $||A - B||_F/||A||_F$ for different kernels given in [2].

<table>
<thead>
<tr>
<th>$l$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 11$</th>
<th>$k = 14$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>7.69e-05</td>
<td>3.06e-08</td>
<td>1.79e-10</td>
<td>1.04e-11</td>
</tr>
<tr>
<td>6</td>
<td>1.14e-04</td>
<td>4.68e-08</td>
<td>2.78e-10</td>
<td>1.86e-11</td>
</tr>
<tr>
<td>8</td>
<td>1.30e-04</td>
<td>5.40e-08</td>
<td>3.22e-10</td>
<td>2.27e-11</td>
</tr>
<tr>
<td>10</td>
<td>1.36e-04</td>
<td>5.67e-08</td>
<td>3.38e-10</td>
<td>2.41e-11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$l$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 11$</th>
<th>$k = 14$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>7.57e-05</td>
<td>3.10e-08</td>
<td>1.82e-10</td>
<td>1.18e-11</td>
</tr>
<tr>
<td>6</td>
<td>1.13e-04</td>
<td>4.73e-08</td>
<td>2.82e-10</td>
<td>1.93e-11</td>
</tr>
<tr>
<td>8</td>
<td>1.29e-04</td>
<td>5.44e-08</td>
<td>3.25e-10</td>
<td>2.23e-11</td>
</tr>
<tr>
<td>10</td>
<td>1.35e-04</td>
<td>5.71e-08</td>
<td>3.42e-10</td>
<td>2.33e-11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$l$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 11$</th>
<th>$k = 14$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>9.18e-05</td>
<td>5.24e-08</td>
<td>3.54e-10</td>
<td>1.12e-11</td>
</tr>
<tr>
<td>6</td>
<td>1.56e-04</td>
<td>9.09e-08</td>
<td>6.25e-10</td>
<td>1.55e-11</td>
</tr>
<tr>
<td>8</td>
<td>1.98e-04</td>
<td>1.17e-07</td>
<td>8.07e-10</td>
<td>1.87e-11</td>
</tr>
<tr>
<td>10</td>
<td>2.25e-04</td>
<td>1.34e-07</td>
<td>9.29e-10</td>
<td>2.01e-11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$l$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 11$</th>
<th>$k = 14$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2.09e-05</td>
<td>5.53e-09</td>
<td>2.75e-11</td>
<td>2.29e-11</td>
</tr>
<tr>
<td>6</td>
<td>2.92e-05</td>
<td>7.85e-09</td>
<td>3.94e-11</td>
<td>2.26e-11</td>
</tr>
<tr>
<td>8</td>
<td>3.20e-05</td>
<td>8.59e-09</td>
<td>4.31e-11</td>
<td>2.39e-11</td>
</tr>
<tr>
<td>10</td>
<td>3.28e-05</td>
<td>8.80e-09</td>
<td>4.41e-11</td>
<td>2.53e-11</td>
</tr>
</tbody>
</table>
(ii) Accuracy of the Approximation Solution and the Convergence Rate:

Next, we show the accuracy and efficiency of solving (3.1) using our approximation matrix \( B \). In the computations, the true solutions \( \mathbf{y} \) are arbitrary random vectors. For kernel functions (i) to (iv), the right hand sides are computed by \( \mathbf{b} = (I - A)\mathbf{y} \) and our approximation solutions are the solutions of \((I - B)\mathbf{z} = \mathbf{b}\). For kernel functions (v) and (vi), we set \( \mathbf{b} = (I - DA)\mathbf{y} \) and \( \mathbf{z} \) are the solutions of \((I - DB)\mathbf{z} = \mathbf{b}\). For kernel function (vii), we set \( \mathbf{b} = (I - Re(D_1AD_2))\mathbf{y} \) and \( \mathbf{z} \) is the solution of \((I - Re(D_1BD_2))\mathbf{z} = \mathbf{b}\), see (3.35). The relative errors \( \|\mathbf{y} - \mathbf{z}\|_2/\|\mathbf{y}\|_2 \) of the approximation solutions \( \mathbf{z} \) are shown in Tables 4a–4b.

All matrix equations are solved by the CGLS method. This method is based on solving the normal equation of the given equation by the conjugate gradient method, see [4] for instance. In the CGLS method, our initial guess is the zero vector 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 \( q \)th iteration. The numbers of iterations required for convergence for the seven kernels are given in Tables 5a–5b.

Since the kernel functions we tried are at most weakly singular, we see from Table 5 that the convergence rate is linear as expected. Recall from §2 that the cost of the matrix-vector multiplication \( B\mathbf{y} \) is of \( O(nk \log n) \) operations, the total cost of solving each of these systems is thus of \( O(nk \log n) \) operations. We emphasize again that in order to get the approximate solution \( \mathbf{z} \), we only have to form the factors of \( B \) which require only \( O(nk) \) operations and there is no need to form \( A \).

(iii) Cost of the Matrix-Vector Multiplications:

We compare the cost of matrix-vector multiplications \( A\mathbf{y} \) and \( B\mathbf{y} \), where \( B \) is given by (3.17) and (3.19) respectively and \( \mathbf{y} \) is any vector. Tables 6a–6b
### Table 4a: $||y - z||_2/||y||_2$ for different kernels.

<table>
<thead>
<tr>
<th>$l$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 12$</th>
<th>$k = 14$</th>
<th>$k = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3.16e-05</td>
<td>6.21e-09</td>
<td>4.84e-11</td>
<td>9.88e-11</td>
<td>1.24e-10</td>
</tr>
<tr>
<td>6</td>
<td>3.14e-05</td>
<td>7.10e-09</td>
<td>4.20e-11</td>
<td>6.98e-11</td>
<td>7.73e-11</td>
</tr>
<tr>
<td>8</td>
<td>2.97e-05</td>
<td>6.97e-09</td>
<td>3.88e-11</td>
<td>4.97e-11</td>
<td>4.34e-11</td>
</tr>
<tr>
<td>10</td>
<td>2.84e-05</td>
<td>6.88e-09</td>
<td>3.42e-11</td>
<td>3.08e-11</td>
<td>1.75e-10</td>
</tr>
</tbody>
</table>

| $l$ | $a(x, t) = \log |x - t|$ |
|-----|-----------------|
| 4   | 2.91e-05 | 5.92e-09 | 4.60e-11 | 7.29e-11 | 2.48e-11 |
| 6   | 2.87e-05 | 6.74e-09 | 3.60e-11 | 2.85e-11 | 4.07e-11 |
| 8   | 2.74e-05 | 6.62e-09 | 2.88e-11 | 3.04e-11 | 2.83e-11 |
| 10  | 2.62e-05 | 6.54e-09 | 2.48e-11 | 2.10e-11 | 2.23e-11 |

| $l$ | $a(x, t) = \cos(\pi x^2) \log |x - t|$ |
|-----|-----------------|
| 4   | 5.58e-05 | 1.76e-08 | 4.61e-11 | 3.91e-11 | 1.15e-10 |
| 6   | 7.22e-05 | 2.50e-08 | 1.04e-10 | 3.18e-11 | 4.66e-11 |
| 8   | 1.42e-04 | 3.31e-08 | 1.68e-10 | 7.47e-11 | 5.62e-11 |
| 10  | 8.65e-05 | 2.72e-08 | 1.65e-10 | 1.68e-10 | 1.89e-10 |

| $l$ | $a(x, t) = \cos(\pi x^2)|x - t|^{-1/2}$ |
|-----|-----------------|
| 4   | 3.65e-06 | 3.86e-10 | 9.47e-13 | 8.26e-13 | 5.52e-11 |
| 6   | 3.41e-06 | 3.96e-10 | 2.65e-11 | 2.57e-11 | 2.56e-11 |
| 8   | 3.77e-06 | 3.97e-10 | 1.84e-11 | 1.02e-11 | 1.14e-11 |
| 10  | 3.82e-06 | 3.95e-10 | 7.16e-12 | 5.72e-12 | 4.67e-12 |
Chapter 3  Fast Algorithm for Second Kind Integral Equations

\[
a(x, t) = \begin{cases} 
1 + \frac{1}{2} \sin(100x) \log |x - t| & \text{for } k = 4, 8, 12, 14, 16 \\
\sin(100x) \log |x - t| & \text{for } k = 10
\end{cases}
\]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 12$</th>
<th>$k = 14$</th>
<th>$k = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3.25e-05</td>
<td>6.32e-09</td>
<td>1.94e-10</td>
<td>2.36e-11</td>
<td>2.35e-11</td>
</tr>
<tr>
<td>6</td>
<td>3.16e-05</td>
<td>7.30e-09</td>
<td>8.83e-11</td>
<td>1.26e-10</td>
<td>1.53e-10</td>
</tr>
<tr>
<td>8</td>
<td>3.05e-05</td>
<td>7.43e-09</td>
<td>6.91e-11</td>
<td>4.65e-11</td>
<td>7.75e-11</td>
</tr>
<tr>
<td>10</td>
<td>2.95e-05</td>
<td>7.31e-09</td>
<td>6.19e-11</td>
<td>5.95e-11</td>
<td>4.61e-11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 12$</th>
<th>$k = 14$</th>
<th>$k = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2.36e-05</td>
<td>4.21e-09</td>
<td>1.34e-11</td>
<td>1.81e-11</td>
<td>1.83e-11</td>
</tr>
<tr>
<td>6</td>
<td>1.88e-05</td>
<td>4.95e-09</td>
<td>6.01e-11</td>
<td>9.88e-11</td>
<td>9.61e-11</td>
</tr>
<tr>
<td>8</td>
<td>2.11e-05</td>
<td>5.22e-09</td>
<td>7.55e-11</td>
<td>1.05e-10</td>
<td>7.47e-11</td>
</tr>
<tr>
<td>10</td>
<td>2.16e-05</td>
<td>5.28e-09</td>
<td>8.76e-11</td>
<td>5.63e-11</td>
<td>8.77e-11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 12$</th>
<th>$k = 14$</th>
<th>$k = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4.27e-06</td>
<td>9.93e-10</td>
<td>2.74e-11</td>
<td>4.51e-11</td>
<td>8.50e-11</td>
</tr>
<tr>
<td>6</td>
<td>2.88e-06</td>
<td>6.20e-10</td>
<td>2.25e-11</td>
<td>1.76e-11</td>
<td>4.23e-11</td>
</tr>
<tr>
<td>8</td>
<td>1.76e-06</td>
<td>5.27e-10</td>
<td>1.45e-11</td>
<td>8.46e-12</td>
<td>1.56e-11</td>
</tr>
<tr>
<td>10</td>
<td>1.29e-06</td>
<td>5.09e-10</td>
<td>5.71e-12</td>
<td>7.12e-12</td>
<td>6.12e-11</td>
</tr>
</tbody>
</table>

Table 4b: $||y - z||_2/||y||_2$ for different kernels.
\[ a(x, t) = \log |x - t| \]
\[ a(x, t) = \cos(xt^2) \log |x - t| \]
\[ a(x, t) = \cos(xt^2)|x - t|^{-1/2} \]
\[ a(x, t) = \cos(xt^2)|x - t|^{1/2} \]

Table 5a: Numbers of iterations required for convergence.
### Table 5b: Numbers of iterations required for convergence.

<table>
<thead>
<tr>
<th>$l$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 12$</th>
<th>$k = 14$</th>
<th>$k = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>13</td>
<td>14</td>
<td>13</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>

Table: $a(x, t) = (1 + \frac{1}{2} \sin(100x)) \log |x - t|$

<table>
<thead>
<tr>
<th>$l$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 12$</th>
<th>$k = 14$</th>
<th>$k = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

Table: $a(x, t) = \sin(100x) \log |x - t|$

<table>
<thead>
<tr>
<th>$l$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 12$</th>
<th>$k = 14$</th>
<th>$k = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>14</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>11</td>
</tr>
</tbody>
</table>

Table: $a(x, t) = \cos(1000|x - t|) \log |x - t|$
Chapter 3  Fast Algorithm for Second Kind Integral Equations

<table>
<thead>
<tr>
<th>l</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$By$</td>
<td>ratio</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>3.5000</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>2.2857</td>
</tr>
<tr>
<td>6</td>
<td>37</td>
<td>2.3125</td>
</tr>
<tr>
<td>7</td>
<td>85</td>
<td>2.2973</td>
</tr>
<tr>
<td>8</td>
<td>188</td>
<td>2.2118</td>
</tr>
<tr>
<td>9</td>
<td>413</td>
<td>2.1968</td>
</tr>
<tr>
<td>10</td>
<td>897</td>
<td>2.1719</td>
</tr>
<tr>
<td>11</td>
<td>1,935</td>
<td>2.1572</td>
</tr>
<tr>
<td>12</td>
<td>4,151</td>
<td>2.1452</td>
</tr>
</tbody>
</table>

Table 6a: Kflops counts in computing $By$ and $Ay$ for $k = 4$ and $k = 8$.

give the numbers of floating point operations in thousand (Kflops) required in computing the products $Ay$ and $By$. We note that the counts do not depend on the kernel functions used. In the tables, ratio denotes the ratio of operation counts when the matrix size $n$ is doubled (or $l$ is added by one). We can see from the ratio that the cost of the matrix-vector multiplication is approaching $O(nk\log n)$ for $By$ but it is $O(n^2)$ for $Ay$.

(iv) Relation between $k$ and the Accuracy of the Approximation:

Finally, we illustrate our results in Theorem 2, see (3.32). We test kernel functions (i) to (iv). Results for kernel functions (v) to (vii) are omitted in Figures 3a-3b as they have the same kernel functions as in (i) and (ii). Recall that kernel functions (i) and (ii) have $\delta = 0$, kernel function (iii) has $\delta = -0.5$ and kernel function (iv) has $\delta = 0.5$. We see from the figure that for a fixed $k$, when $l$ increases, the error for the kernel functions which have nonnegative $\delta$ is almost constant. For the kernel function which has a negative $\delta$, the error
Table 6b: Kflops counts in computing By and Ay for \( k = 12 \) and \( k = 16 \).

3.5 Concluding Remarks

In this thesis, we have developed a fast multiplication scheme for integral operators. The cost of our scheme is the same as that in [2]. However, our scheme is simpler. Besides, our method provides better approximations than that in [2] as we use the Chebyshev points which are the optimal interpolation points. They also enable us to carry out error analysis of our method which was not given in [2]. The results shown in Theorems 1 and 2 indicate that the memory requirement for storing the approximation matrix \( B \) is similar as that of the approximation matrix constructed in [11].
Chapter 3  Fast Algorithm for Second Kind Integral Equations

Figure 3a: The log of $\|A - B\|_F$ against $l$ for $k = 4$.

Figure 3b: The log of $\|A - B\|_F$ against $l$ for $k = 6$. 
References


