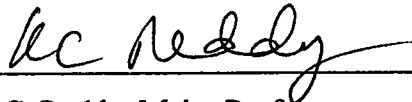


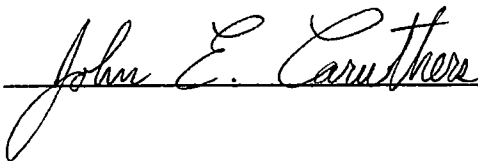
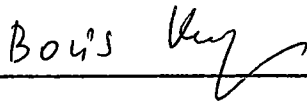
To the Graduate Council:

I am submitting herewith a thesis written by Julie Reneé Spangler entitled "Iterative Solutions of the Discretized Helmholtz Equation." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mathematics.



K. C. Reddy, Major Professor

We have read this thesis
and recommend its acceptance:



Accepted for the Council:



Associate Vice Chancellor
and Dean of The Graduate School

ITERATIVE SOLUTIONS OF THE DISCRETIZED HELMHOLTZ EQUATION

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Julie Renée Spangler

May 2000

ACKNOWLEDGEMENTS

Many people at UTSI have made my academic and research endeavors at UTSI fulfilling. I would especially like to thank Dr. K.C. Reddy and Dr. John Caruthers for all of their help with this research and for the hours spent in their offices. I want to thank my major professor, Dr. Reddy, for reading several drafts of this thesis and providing many insightful comments. I would finally like to thank my committee, Dr. K.C. Reddy, Dr. John Caruthers, and Dr. Boris Kuperzhmidt, for their helpful suggestions regarding this thesis and for the academic guidance each provided me.

ABSTRACT

This research determines the efficiency with which the discretized Helmholtz equation can be solved using a preconditioned Generalized Minimum Residual (GMRES) method. The Helmholtz equation is discretized by the Green's Function Discretization (GFD) method. Both two-dimensional (2-D) and three-dimensional (3-D) Helmholtz equations are analyzed. The efficiency of the method is compared to direct solvers. It will be shown in this thesis that the preconditioned iterative method used here requires fewer operations and less computing time than direct methods for even relatively small 3-D problems. Finally, it will be shown in each of the 2-D and 3-D problems that the iterative method can solve systems of equations that are much larger than the direct methods can handle. Specifically, we demonstrate the solution of a system with more than one million equations with complex variables.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION.	1
2. THEORY.	5
Krylov Subspace Methods.....	6
GMRES.	8
ILUT.....	11
Finite Difference Techniques.	13
Green's Function Discretization	16
3. RESULTS.....	20
Notation.....	20
Two Dimensional Test Problem	23
Three Dimensional Test Problem #1	34
Three Dimensional Test Problem #2..	46
4. CONCLUSIONS	56
LIST OF REFERENCES	58
APPENDICES..	60
APPENDIX 1. Coefficients of the Green's Function Discretization	61
APPENDIX 2. FORTRAN Codes.....	64
VITA.	81

LIST OF TABLES

TABLE	PAGE
3 1. 2-D condition numbers.....	33
3 2. 2-D direct and iterative CPU times (seconds)	33
3.3. 3-D problem #1 condition numbers.....	44
3.4. 3-D problem #1 direct and iterative CPU times.	44
3 5 3-D problem #2 condition numbers.	54
3 6. 3-D problem #2 direct and iterative CPU times.	54

LIST OF FIGURES

FIGURE	PAGE
2.1. The orthogonality condition.....	7
2.2 2-D rectangular stencil for point $\phi_{i,j}$	14
3.1. 2-D: Comparison of solutions produced by the iterative method, the direct method, and the analytical method $N_x = 31, k = 5.0$	24
3.2. 2-D: Difference in solutions produced by the iterative method and the direct method $N_x = 31, k = 5.0$	27
3.3. 2-D: Comparison of preconditioned GMRES and non-preconditioned GMRES $N_x = 11, k = 5.0$	28
3.4 2-D: Work required to reach convergence for different values of N_x and k	30
3.5. 3-D #1: Comparison of solutions produced by the iterative method, the direct method, and the analytical method $N_x = 11, k = 5.0$	35
3.6 3-D #1: Difference in solutions produced by the iterative method and the direct method $N_x = 11, k = 5.0$	38
3.7. 3-D #1. Comparison of preconditioned GMRES and non-preconditioned GMRES $N_x = 11, k = 5.0$	39
3.8. 3-D #1: Work required to reach convergence for different values of N_x and k	41
3.9. 3-D #2: Comparison of solutions produced by the iterative method, the direct method, and the analytical method $N_x = 11, k = 5.0, y = 0.5, z = 0.5$	47

3.10.	3-D #2: Difference in solutions produced by the iterative method and the direct method $N_x = 11, k = 5.0$	48
3.11.	3-D #2: Comparison of preconditioned GMRES and non-preconditioned GMRES $N_x = 11, k = 5.0$	49
3.12.	3-D #2: Work required to reach convergence for different values of N_x and k	52

Chapter 1

INTRODUCTION

Models of applied or theoretical variables often lead to a system of linear equations

$$Ax = b \quad (1.1)$$

where A is an $n \times m$ coefficient matrix containing the relationships between the variables, b is an n -dimensional column vector based on the data, and x is an m -dimensional column vector containing the unknown variables. A system of linear equations can be written in this matrix form when multiple unknowns are related through several relationships. Usually, the first technique learned to solve this system of equations when A is a square matrix is the method of successive elimination of variables. It amounts to multiplying on the left of both sides of the equation by the inverse of the matrix A : A^{-1} . There are two basic classes of solution methods: direct and iterative. The former is the more classical class of methods and includes Gaussian elimination. The latter is more recent in its development and comprises the set of methods that will be discussed in this thesis.

The need to solve larger systems of equations that were previously unsolvable by direct methods due to the immense amount of storage and arithmetic operations required by the computer has encouraged the development of iterative techniques. According to [1], a “very large” system of equations consisted of approximately 20 equations in 1950 and 20,000 in 1995. Iterative techniques are better equipped to solve very large systems

of equations than direct methods. The definition of “very large” is therefore changing with the advent of iterative techniques. Iterative methods find an approximate solution to a problem, not the exact solution; however, the solution can be made as accurate as necessary.

The purpose of this research is to solve the discretized form of the Helmholtz equation

$$\nabla^2 \phi + k^2 \phi = 0 \quad (1.2)$$

where ϕ is the velocity potential and k is the wavenumber, using the Incomplete Lower Upper (LU) factorization with Threshold (ILUT) preconditioned Generalized Minimum Residual (GMRES) iterative solver. The Helmholtz equation simulates wave propagation through a medium, as in acoustics. The amount of computational work required by the ILUT preconditioned GMRES solver is observed and is used to assess the practicality of this method for solving large systems that cannot be solved by direct methods or simple iterative methods like Gauss-Seidel. The computational work can be measured by the number of arithmetic operations performed in the solution process. The final purpose of this thesis is to understand the practical aspects of solving large systems of equations using the preconditioned GMRES method.

Both the two- and three-dimensional Helmholtz equations are discretized using the Green's Function Discretization (GFD) method developed by J. E. Caruthers and associates [2]. It is a sixth-order accurate method on a uniform mesh. One two-dimensional (2-D) test case and two three-dimensional (3-D) test cases are considered. In each case the Helmholtz equation is defined in a square (a cube in 3-D) region. The

boundary conditions applied to the 2-D case are comprised of two Dirichlet conditions, in which the velocity potential ϕ is defined, and two Neumann conditions, in which the normal derivative of ϕ is defined. The first 3-D case is similar to the 2-D case in that it has two Dirichlet boundary conditions and four Neumann boundary conditions applied on the sides of the cube. The second 3-D test case involves a known traveling wave emanating from a chosen point source such that this wave traverses over the three-dimensional cube of interest. This last case is of particular interest because the resulting equations are complex-valued, whereas the previous two cases involve only the real variables.

This thesis will demonstrate that the ILUT preconditioned GMRES method is an efficient method for solving very large systems of equations of the type produced by the GFD for the Helmholtz equation. This method will be shown to be capable of solving systems of equations with greater than one million equations, which is impractical using direct methods [1]. It is also shown that the ILUT preconditioner increases the efficiency of the GMRES solver.

Throughout this research, the GMRES and ILUT routines are used from the SPARSKIT [3] version 2 package of FORTRAN routines for sparse matrices. More specifically, since there are multiple forms of the GMRES algorithm available in SPARSKIT, the PGMRES routine, written by Y. Saad [3] is used for the two cases using real matrices since the program accounts for pre-conditioning. Y. Saad provided the complex version of GMRES [4], which is currently not contained in SPARSKIT. This version did not account for preconditioning, so the author of this research altered it

accordingly and altered the ILUT and various subroutines to handle complex matrices.

These codes are provided in Appendix 2 of this thesis.

Chapter 2

THEORY

Direct methods for solving systems of linear equations, such as Gaussian elimination, perform well for small-scale problems. However, solving large-scale problems with Gaussian elimination is not feasible due to large growth of the operation count, even for banded systems that we encounter for the Helmholtz equation. If the number of points in one direction of a uniform grid is denoted N_x , then the operation count grows as $O(2N_x^P)$, where $P = 4$ for two dimensions and $P = 7$ for three dimensions with real variables. For systems with complex variables the multiplicative factor in the operation count is higher. It becomes impractical to solve systems on the order of 20,000 variables by direct methods on present day single processors, whereas we would like to be able to solve systems with one million variables or more. Iterative methods are needed to solve these larger problems. One class of iterative techniques is the Krylov subspace methods, of which the Generalized Minimum Residual (GMRES) method is a member.

This chapter will cover the basic theory of the iterative and discretization techniques used in this research. First, Krylov subspace methods will be discussed, followed by discussions of GMRES and the ILUT preconditioner. The theory of the discretization techniques begins with a basic synopsis of finite difference techniques, followed by a description of the Green's Function Discretization.

Krylov Subspace Methods

The Krylov subspace methods are based on projection methods that solve a problem in a lower dimensional Krylov subspace. Projection methods seek an approximate solution x_m to the problem $Ax = b$ such that the norm of the residual $r_m = b - Ax_m$ is minimized [5]. The approximate solution x_m is chosen from the affine subspace $x_0 + K_m$, where $K_m = \text{span}\{r_0, Ar_0, \dots, A^{m-1}r_0\}$ is the Krylov subspace and x_0 is an initial guess to the solution x . The vector r_0 is the initial residual $b - Ax_0$. Requiring the norm of the residual to be a minimum creates the subspace of constraints, L_m , such that r_m must be perpendicular to L_m , i.e. r_m is defined to be orthogonal to the m linearly independent vectors that form the basis of L_m . The configuration of the Krylov subspace $K_m = \{r_0, Ar_0, \dots, A^{m-1}r_0\}$ implies that the approximations are of the form

$$A^{-1}b \approx x_m = x_0 + p_{m-1}(A)r_0 \quad (2.1)$$

where $p_{m-1}(A)$ is a specific polynomial of degree $m-1$.

All the Krylov subspace techniques provide similar types of polynomial approximations. However, the choice of the constraints, L_m , differs and affects the iterative techniques. Two of the best-known choices for L_m include $L_m = K_m$ and $L_m = AK_m$. Let us consider the special case $L_m = AK_m$. As previously mentioned, the information provided by the initial guess x_0 is exploited by choosing the approximate solution x_m from the affine subspace $x_0 + K_m$. Now, if we write $x_m = x_0 + z_m$, where

$z_m \in K_m$, the residual vector $r_m = b - A(x_0 + z_m) = r_0 - Az_m$. Thus, the approximation process used by Krylov subspace methods can be described as finding

$$x_m = x_0 + z_m, \quad z_m \in K_m \quad (2.2)$$

such that $(r_0 - Az_m) \perp L_m$, i.e. $(r_0 - Az_m, w) = 0 \quad \forall w \in L_m$.

This orthogonality condition is shown in Figure 2.1.

The Krylov subspace methods search for a solution x_m to an n -th dimensional matrix problem in an m -th dimensional Krylov subspace K_m . The dimension m of the Krylov subspace increases by one at each step of the approximation process [5]. However, the dimension of the Krylov subspace typically is significantly smaller than the original dimension of the problem, i.e. $m \ll n$. This clearly is beneficial when attempting to solve a problem with a large dimension that otherwise would be unsolvable using direct methods.

In summary, the Krylov subspace methods are a general projection method based

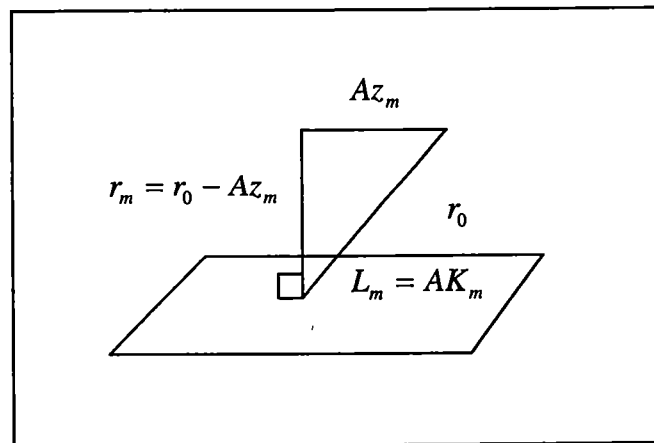


Figure 2.1. The orthogonality condition

on selecting a pair of subspaces K_m and L_m with appropriate bases that span each space. The choice of L_m and preconditioning method dictate the differences between Krylov subspace methods.

GMRES

The Generalized Minimum Residual (GMRES) method, developed by Saad and Schultz [6], is a Krylov subspace method based on taking $L_m = AK_m$, where K_m again denotes the m -th Krylov subspace. This choice of the subspace of constraints L_m forms the class of minimum residual Krylov subspace methods. The approximate solution $x_m \in x_0 + K_m$ is chosen such that it minimizes the residual norm $\|r_m\|_2 = \|b - Ax_m\|_2 = \|r_0 - Az_m\|_2$ over the Krylov subspace K_m .

Conceptually, GMRES starts with an initial guess x_0 and initial residual vector $r_0 = b - Ax_0$ and defines the m -th approximate solution as $x_m = x_0 + z_m$, where z_m is the element in K_m which solves

$$\min \|b - A(x_0 + z_m)\|_2 = \min \|r_0 - Az_m\|_2 \quad (2.3)$$

over all $z_m \in K_m$

The GMRES algorithm can be summarized as follows [5].

1. Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, and $v_1 = \frac{r_0}{\beta}$. Define $e_1 = [1, 0, 0, \dots, 0]^T$.
2. Define the $(m+1) \times m$ Hessenberg matrix $\bar{H}_m = \{h_{ij}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$. Set $\bar{H}_m = 0$.
3. For $j = 1, 2, \dots, m$ do:
4. Compute $w_j = Av_j$

5. For $i = 1, \dots, j$ do:
6. $h_y = (w_j, v_i)$
7. $w_j = w_j - h_y v_i$
8. Enddo
9. $h_{j+1,j} = \|w_j\|_2$. If $h_{j+1,j} = 0$ set $m = j$ and go to 12
10. $v_{j+1} = \frac{w_j}{h_{j+1,j}}$
11. Enddo
12. Compute y_m the minimizer of $\|\beta e_1 - \bar{H}_m y\|_2$ and $x_m = x_0 + V_m y_m$.

The GMRES algorithm uses the Arnoldi-Modified Gram-Schmidt method, an orthogonal projection method onto K_m , in lines 4 through 10 of the previous description.

Arnoldi's method produces an upper Hessenberg matrix $\bar{H}_m$ with elements h_y and computes an L_2 -orthonormal basis $\{v_1, v_2, \dots, v_m\}$ of the Krylov subspace K_m . Matrix V_m is the $n \times m$ matrix whose columns are the basis vectors $v_1, v_2, \dots, v_m$

From lines 4, 7, and 10 in the GMRES algorithm we see that

$$\begin{aligned}
 w_j &= Av_j - \sum_{i=1}^j h_y v_i & \text{and} & & v_{j+1} &= \frac{w_j}{h_{j+1,j}} \\
 \Rightarrow h_{j+1,j} v_{j+1} &= Av_j - \sum_{i=1}^j h_y v_i \\
 \Rightarrow Av_j &= V_{m+1} \bar{H}_m \\
 \Rightarrow AV_m &= V_{m+1} \bar{H}_m.
 \end{aligned} \tag{2.4}$$

Letting $\beta = \|r_0\|_2$ produces

$$\begin{aligned}
 b - Ax &= b - A(x_0 + V_m y) \\
 &= r_0 - AV_m y \\
 &= r_0 - V_{m+1} \bar{H}_m y
 \end{aligned}$$

$$\begin{aligned}
&= \beta v_1 - V_{m+1} \bar{H}_m y \\
&= V_{m+1} (\beta e_1 - \bar{H}_m y).
\end{aligned} \tag{2.5}$$

The column vectors of V_{m+1} are orthonormal, so

$$J(y) = \|b - A(x + V_m y)\|_2 = \|\beta e_1 - \bar{H}_m y\|_2 \tag{2.6}$$

Thus, the GMRES approximation is the unique vector of $x_0 + K_m$ which minimizes $J(y)$. This produces the final line of the GMRES algorithm discussed previously:

$$x_m = x_0 + V_m y_m \text{ where } y_m \text{ is the vector } y \text{ which produces } \min \| \beta e_1 - \bar{H}_m y \|_2 \tag{2.7}$$

The steps comprising the GMRES algorithm are repeated m times until convergence is reached, where m is the dimension of the Krylov subspace K_m . Clearly, the size of the Krylov subspace is an important factor in determining the amount of memory needed to solve a given system of linear equations. Large m not only increases the operation count, it also requires more storage. A simple solution to this problem [6] is the restarted version of GMRES, denoted GMRES(t), which limits the size of the Krylov subspace to t . If convergence is reached in less than t steps the procedure stops. Else, after step t is completed, the procedure is restarted with $x_0 = x_t$, the current approximate solution computed at step t . The procedure is repeated with a new Krylov subspace and subspace of constraints. This ensures that the Krylov subspace is always of manageable size. The restarted version of GMRES is used in this research because the goal of this research is to solve larger systems of equations.

ILUT

Preconditioning is often an invaluable tool when solving large systems of equations. Applying a preconditioner such as the Incomplete LU factorization with Threshold (ILUT) to the coefficient matrix before using an iterative solver such as GMRES will generally make the coefficient matrix more well-conditioned and will take advantage of any structure the matrix may have. Applying a preconditioner to a coefficient matrix can greatly reduce the number of iterations required by the iterative solver for convergence. However, if an inappropriate preconditioning technique has been chosen, the iterative solver may never reach convergence or the solver could stagnate, i.e., the residual stays the same and does not reduce with each iteration step as it should when the method is performing properly.

Rather than solving the original problem $Ax = b$, the preconditioned system

$$M^{-1}Ax = M^{-1}b$$

is solved, where M^{-1} is computed in such a way that $M^{-1}A$ is better conditioned than A . Generally, $M^{-1}A$ and M are not computed explicitly [5]. The iterative solver usually operates with A and M^{-1} whenever necessary.

Unlike Gaussian elimination, which reduces the coefficient matrix A to a full LU factorization, an incomplete LU factorization reduces matrix A to an incomplete LU factorization such that the residual matrix $R = LU - A$ satisfies certain constraints [5]. The elements are dropped from the factorization (set to zero) according to their size relative to the other elements in the row and the prescribed level of fill (lfil). The level of fill determines the number of elements to be kept in L and U , excluding the diagonal elements, which are always kept. For example, if lfil is set to 10, the L array will contain

10 non-zero elements in each row and the U array will contain 11 non-zero elements in each row, including the element on the diagonal of A. Prescribing a level of fill limits the amount of memory needed to store the LU factorization of matrix A. As mentioned, the elements are also dropped according to their size. A tolerance parameter (droptol) is set such that the largest elements in each row are kept. If droptol is set to zero we have the ILU(0) preconditioner, which drops elements based solely on the level of fill.

The general algorithm for ILUT can be described as follows [5]. In the algorithm, w denotes a full-length working row, w_k is the k -th element of this row, and a_{i*} is the i -th row of A.

1. For $i = 1, \dots, n$ do.
2. $w = a_{i*}$
3. For $k = 1, \dots, i-1$ and when $w_k \neq 0$ do:
4. $w_k = \frac{w_k}{a_{kk}}$
5. Apply a dropping rule to w_k
6. If $w_k \neq 0$ then
7. $w = w - w_k * u_{k*}$
8. Endif
9. Enddo
10. Apply a dropping rule to row w
11. $l_{i,j} = w_j$ for $j = 1, \dots, i-1$ (creating the matrix L)
12. $u_{i,j} = w_j$ for $j = i, \dots, n$ (creating the matrix U)
13. $w = 0$
14. Enddo

Generally, the dropping strategy employed in line 5 drops an element w_k if it is smaller than the relative tolerance (droptol multiplied by the 2-norm of the elements in the i -th row). The dropping strategy employed in line 10 again drops any element in the

row less than the relative tolerance. Then, only a maximum of "lfil" number of largest elements are kept in the lower part of the matrix in each row, while a maximum of "lfil" plus one elements are kept in the upper part of the matrix in each row.

Finite Difference Techniques

The method of finite differences is a type of basic numerical scheme that is discussed here to provide a foundation for difference schemes. It is not the discretization scheme utilized in this research, however, because it is not as accurate as the Green's Function Discretization, which will be presented next. The finite difference discretization was utilized by the author of this thesis to test the FORTRAN codes used. It is felt, however, that an introduction to the theory of finite differences will provide a general basis for difference schemes.

The consideration of both applied and theoretical problems often produces differential equations. Sometimes, the solution to the differential equation can be solved in terms of common functions. However, this is not usually the case. Differential equations are usually solved using numerical schemes since applied problems may not have an analytical solution.

Consider a differential equation that is known to be valid over a domain D . The continuous solution over D is approximated by representing the solution at various discrete points within and on the boundary of D . This region of points is called the discretized grid. The values of the solution are assumed to be known, or some other conditions are imposed, on parts of the boundary of D and are referred to as boundary

conditions. The goal is to determine the value of the solution to the differential equation at each interior and some boundary points of D . The solution depends on the discrete approximation to the continuous solution of the differential equation.

Rather than solving the Helmholtz equation in the continuous form shown in equation (1.2), the partial derivatives can be replaced with algebraic difference approximations. This produces algebraic equations for ϕ at each grid point in the domain. An arbitrarily shaped computational grid can be used to discretize the Helmholtz equation, but here, a rectangular shape is employed with finite difference discretizations. A typical two dimensional grid is shown in Figure 2.2

A five-point stencil for point $\phi_{i,j}$ concerns writing $\phi_{i,j}$ in terms of its neighboring points on the main cross, $\phi_{i-1,j}, \phi_{i+1,j}, \phi_{i,j-1}, \phi_{i,j+1}$. A nine-point stencil uses all eight points shown in Figure 2.2 surrounding $\phi_{i,j}$. The stencils are similarly created for three dimensions. Picture three of the two dimensional rectangular grids shown in Figure 2.2. The first corresponds to $z = -1$, the second to $z = 0$, and the third to $z = 1$. Thus, the 3-D

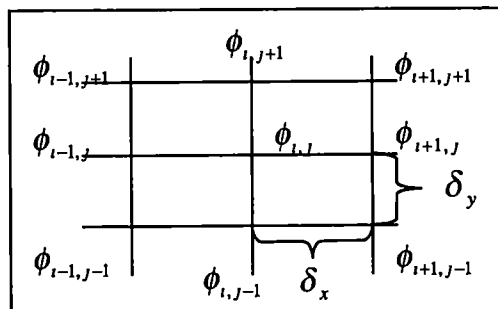


Figure 2.2. 2-D rectangular stencil for point $\phi_{i,j}$

stencil analogous to the five point stencil in 2-D has 7 points and the 3-D stencil analogous to the 9 point stencil in 2-D has 27 points.

The five-point central difference representation for $\partial^2\phi/\partial x^2$ in 2-D is written as

$$\frac{\phi_{i+1,j} + \phi_{i-1,j} - 2\phi_{i,j}}{\delta_x^2} \quad (2.8)$$

The central difference formula for $\partial^2\phi/\partial y^2$ is similarly represented as

$$\frac{\phi_{i,j+1} + \phi_{i,j-1} - 2\phi_{i,j}}{\delta_y^2} \quad (2.9)$$

For simplicity, assume that the horizontal and vertical grid spacings are equal, i.e. $\delta_x = \delta_y = \delta$. Combining this assumption with the central difference formulas given in equations (2.8) and (2.9), the Helmholtz equation can be written in discretized form as follows

$$\frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j}}{\delta^2} + k^2\phi_{i,j} = 0$$

or

(2.10)

$$\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} + (k^2\delta^2 - 4)\phi_{i,j} = 0.$$

Repeating this procedure for every interior point of the grid produces a system of linear equations for the Helmholtz equation that can be represented by a matrix equation of the form (1.1). The right-hand side b contains the appropriate boundary conditions and x contains the unknowns ϕ_i .

Clearly, the solution x more accurately represents the continuous solution as the number of points in the computational domain increases. But, as the number of grid

points increases, the number of unknowns also increases. The goal of this research is to solve these larger, more accurate, representations of the Helmholtz equation.

The drawback to the five-point finite discretization is that it is only second-order accurate, i.e. it approximates the Helmholtz equation to $O(\delta^2)$. This is not as accurate as the Green's Function Discretization, which uses a nine-point stencil

Green's Function Discretization

The method of finite differences is one way to discretely represent the Helmholtz equation, but only with second-order accuracy. An alternative method, the Green's Function Discretization (GFD) developed by J E Caruthers and associates [2], is sixth-order accurate, i.e. the error is of $O(\delta^6)$, with appropriate boundary conditions. The GFD method is not only more accurate than finite difference approximations, but it can accurately produce the wave solutions associated with the Helmholtz equation with a coarser mesh than is possible with finite difference methods. Finite differences require a minimum of 10 points per wave, whereas GFD requires only 2-3 points per wave [7].

Referring to [7], the GFD method can be derived as follows. The acoustic velocity potential ϕ , at a point r in a region with data points r_m , can be represented as a superposition of the fields generated by N hypothetical sources of strength γ_j [2]. This can be represented as

$$\phi(r) = \sum_{j=1}^N \gamma_j \psi_j(r) \quad (2.11)$$

where the $\psi_j(r)$ are the solutions of a linear equation such as the Helmholtz equation. These solutions are valid within the small region containing the M discrete points $r_1, r_2, \dots, r_M$ which surround point r_0 . The acoustic potential at each of these points $r_i, i = 1, \dots, M$ is therefore

$$\phi(r_i) = \sum_{j=1}^N \gamma_j \psi_j(r_i). \quad (2.12)$$

Representing this in matrix form

$$\Phi^T = \gamma^T \Psi \quad (2.13)$$

produces row vectors Φ^T and γ^T of dimensions M and N , respectively, and an $N \times M$ matrix Ψ .

The case in which $N > M$ is the case of interest because it is an underdetermined system with an infinite number of solutions γ^T that produce ϕ 's satisfying the Helmholtz equation in the region of interest. Therefore, an appropriate choice of the solution must be made. Caruthers et al suggest that a minimum L_2 -norm requirement be imposed on the solutions γ^T , i.e. the solution with the minimum $\sum_{n=1}^N \gamma_n^2$ is chosen [2]. This can be satisfied through implementation of the Singular Value Decomposition (SVD) followed by general inversion

Once the appropriate solution has been chosen, let

$$\gamma^T = \alpha^T \Psi^H \quad (2.14)$$

where the superscript H denotes the complex conjugate transpose. Substituting this into equation (2.13) produces

$$\begin{aligned}
\Phi^T &= \alpha^T \Psi^H \Psi \\
\Rightarrow \alpha^T &= \Phi^T [\Psi^H \Psi]^{-1} \\
\Rightarrow \gamma^T &= \Phi^T [\Psi^H \Psi]^{-1} \Psi^H
\end{aligned} \tag{2.15}$$

Referring back to equations (2.12) and (2.13) we see that

$$\begin{aligned}
\phi(r_0) &= \gamma^T \Psi_0 = \Phi^T [\Psi^H \Psi]^{-1} \Psi^H \Psi_0 \\
&= \Phi^T a, \quad \text{where } a = [\Psi^H \Psi]^{-1} \Psi^H \Psi_0.
\end{aligned} \tag{2.16}$$

Equation (2.16) is the difference result representing the point r_0 in terms of its surrounding points $r_1, \dots, r_M$, which are incorporated into a . Repeating this process for every point in the domain on which the Ψ are defined discretizes the entire domain. The resulting system for the total P points can be represented by

$$\phi(r_0^p) = (\Phi^p)^T a, \quad p = 1, \dots, P. \tag{2.17}$$

It can be assumed that a does not vary with p if the lattice geometry is uniform and the linear equation L does not vary from point to point [7].

The typical 9-point stencil shown in figure (2.2) is applicable here, where r_0 represents the center point of the stencil. The quantity $r_m - r_i$ is the distance between the two points of the stencil r_m and r_i . The process of determining a produces an 8×8 matrix for the $\Psi^H \Psi$. However, the solution process was simplified by the observation that in the limit of $N \rightarrow \infty$, a has only two distinct elements a_1 and a_2 [7]. These elements are the coefficients of the discretization. The forms of the coefficients are taken from [7] and are shown in Appendix 1 of this thesis. The discretization can now be written as

$$\begin{aligned}\phi_{i,j} = & a_1 (\phi_{i-1,j-1} + \phi_{i+1,j-1} + \phi_{i+1,j+1} + \phi_{i-1,j+1}) \\ & + a_2 (\phi_{i,j-1} + \phi_{i+1,j} + \phi_{i,j+1} + \phi_{i-1,j})\end{aligned}\quad (2.18)$$

Writing this for each element $\phi_{i,j}$ in the computational grid produces a system of linear equations that can be solved for each unknown element.

In three dimensions, a similar result is proven for the coefficient matrix a [7]. The 27-point stencil is applicable in this instance. There turns out to be only three distinct coefficients a_1 , a_2 and a_3 . The formulas for the coefficients were provided by J. E. Caruthers [8] and can be found in Appendix 1 of this thesis. The Green's function discretization of the 3-D Helmholtz equation is

$$\begin{aligned}\phi_{i,j} = & a_1 (\phi_{i-1,j,k} + \phi_{i+1,j,k} + \phi_{i,j-1,k} + \phi_{i,j+1,k} + \phi_{i,j,k-1} + \phi_{i,j,k+1}) \\ & + a_2 (\phi_{i-1,j-1,k} + \phi_{i-1,j+1,k} + \phi_{i+1,j-1,k} + \phi_{i+1,j+1,k} + \phi_{i-1,j-1,k-1} \\ & \quad + \phi_{i-1,j+1,k-1} + \phi_{i+1,j-1,k-1} + \phi_{i+1,j+1,k-1} + \phi_{i-1,j-1,k+1} + \phi_{i-1,j+1,k+1} \\ & \quad + \phi_{i+1,j-1,k+1} + \phi_{i+1,j+1,k+1}) \\ & + a_3 (\phi_{i,j-1,k-1} + \phi_{i,j+1,k+1} + \phi_{i,j+1,k-1} + \phi_{i,j-1,k+1} + \phi_{i-1,j,k-1} \\ & \quad + \phi_{i+1,j,k+1} + \phi_{i+1,j,k-1} + \phi_{i-1,j,k+1}).\end{aligned}\quad (2.19)$$

Expressing each point in the 3-D Helmholtz equation in this manner produces a system of linear equations that can then be solved.

Chapter 3

RESULTS

To test the viability of the ILUT preconditioned GMRES(t) method, three test cases are studied in this thesis. The first to be considered is a two dimensional case with conditions imposed on the boundary of the square in which the Helmholtz equation is to be solved. The second test problem to be considered is a three dimensional case with imposed boundary conditions similar to the 2-D case. These first two problems produce a coefficient matrix $A \in \mathbb{R}^{N \times N}$. The final problem to be considered is also a three dimensional problem, but the imposed boundary conditions in this case create $A \in \mathbb{C}^{N \times N}$. This last case is of the most interest because it is the more realistic of the three cases and it requires the greatest amount of storage and computational work. The main purpose of studying these three cases is not only to prove that the iterative scheme produces a satisfactory approximate solution, but that it can efficiently solve large systems of equations.

Notation

Before discussing the results obtained in this research, some preliminary definitions will be presented. First, it is widely accepted that the number of operations performed by an iterative scheme is the number of floating point operations (flops). There are differing definitions of what a flop is [9]. According to [9], an “old flop” involves a floating point add and a floating point multiply. However, the more precise definition of a flop counts each floating point addition (subtraction) and each

multiplication (division) operation as one flop. This is the characterization utilized in this research. When dealing with complex arithmetic, each complex addition will be counted as two flops; each complex multiplication will be counted as six flops, and each complex division will be counted as eleven flops.

Not only is it desirable to know the operation count, but it is also important to know how many iterations the GMRES(t) method requires for convergence. One iteration is equivalent to one step of the GMRES algorithm described previously. In each step of the GMRES algorithm, there is one matrix-vector multiplication. Each time a matrix-vector multiplication (matvec) is performed, the number of iterations increases by 1. The matvecs are the most costly calculations in the GMRES algorithm.

Another important measure of the efficiency of a solution method is the CPU time required to solve a problem. This is important from a practical point of view as well as an indicator of how "quickly" a solution converges.

The quantity ε is the criterion used to stop the iteration process of GMRES(t). The process is stopped when

$$\|r_m\|_2 \leq \varepsilon \|r_0\|_2 \quad (3.1)$$

Solving (3.1) for ε , we see that it can be denoted the relative residual norm. Typically, adequate values for ε range between 10^{-5} to 10^{-7} , depending on how good the choice for the initial guess x_0 is. The worse the choice of x_0 , the smaller ε must be to ensure an adequate approximation to the solution of the problem. In each of the cases that will be analyzed in this research, the initial guess is set equal to the right-hand-side of the equation, i.e. $x_0 = b$.

The Krylov subspace dimension, t , must also be defined before the solution process can begin. As t is increased, the solution process tends to converge more quickly because the “search space” (Krylov subspace) for the approximate solution has been enlarged. However, as previously discussed, increasing t also increases the amount of storage necessary for the Krylov subspace basis vectors. It also increases the work load of the Gram-Schmidt orthogonalization process in GMRES. In all three cases discussed in this thesis, t is held at a value of 20. This restriction on t saves storage space that can be used to handle larger problems.

Another quantity that needs to be mentioned is kL , where k denotes the wavenumber and L is the length of one side of the square (or cubic) region on which the Helmholtz equation is being solved. For each of the three test problems, the discretized Helmholtz equation is solved for $kL = 5.0$, $kL = 10.0$, and $kL = 50.0$. Without loss of generality, each of the three problems has been designed so that $L = 1$, and thus it is the value of the wavenumber that is being held constant.

The number of points in a specific direction of the computational grid will be denoted N_x , N_y , and N_z . In each case presented here, the regions on which the Helmholtz equation has been discretized is uniform so that $N_x = N_y = N_z$. For simplicity, this parameter will hence be denoted N_x . The number of equations (or rows in the coefficient matrix A) will be denoted N . In 2-D, $N = N_x^2$ and in 3-D, $N = N_x^3$.

The drop tolerance (droptol) and level of fill (lfil) utilized by the ILUT preconditioner also need to be provided. Denoting the number of diagonals in the coefficient matrix as D , it is found in this research that the best level of fill is $(D-1)/2$.

Since this is the minimum number of diagonals that must be contained in the factored matrices L and U, the droptol is set small.

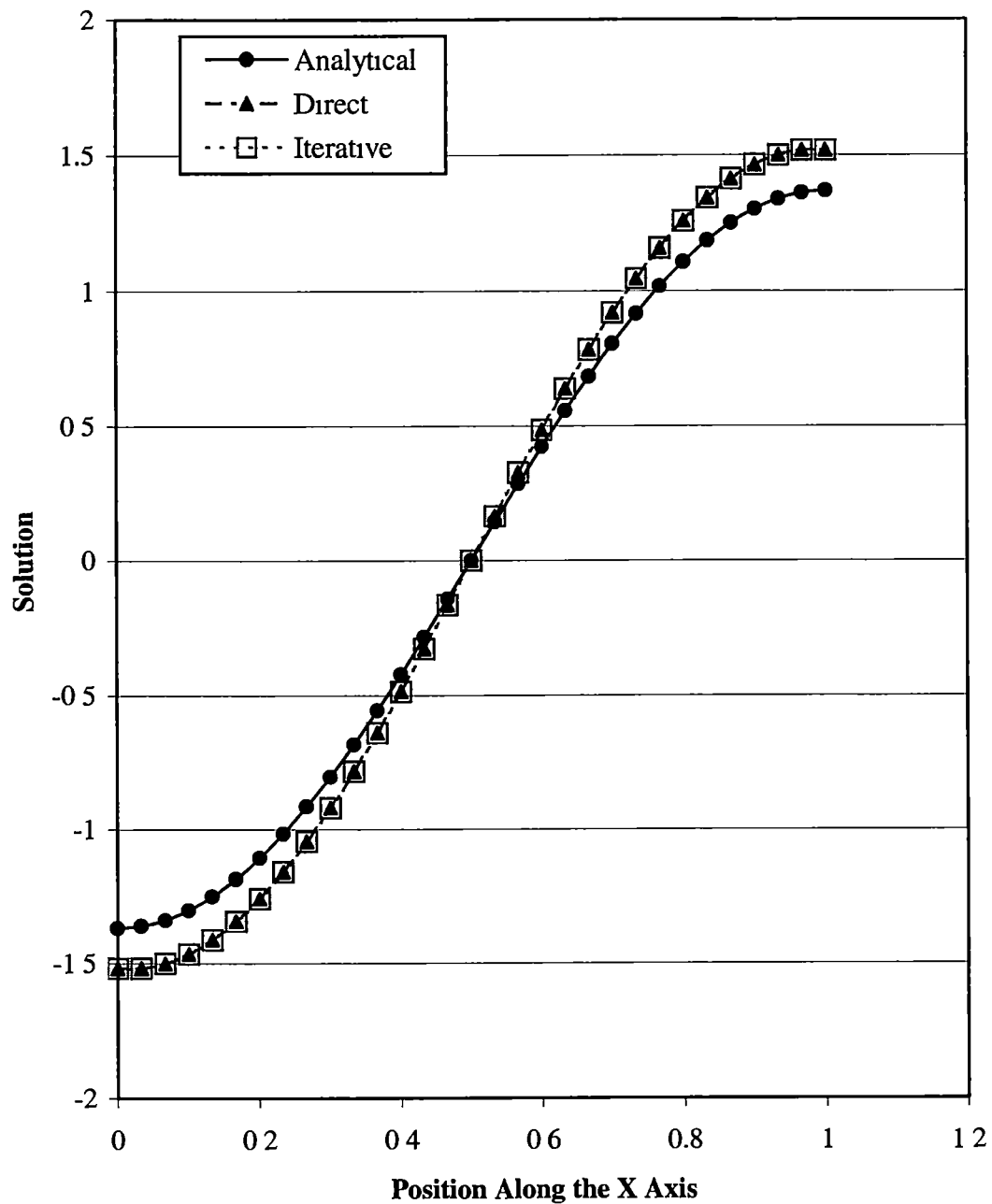
Now that the basic parameters of concern in each problem have been discussed, the results will be presented.

Two Dimensional Test Problem

To determine the practicality of the preconditioned GMRES(t) method, the method is first applied to a two dimensional Helmholtz equation. The problem is designed on a square with $0 \leq x \leq 1$ and $0 \leq y \leq 1$. The boundary conditions are defined as follows

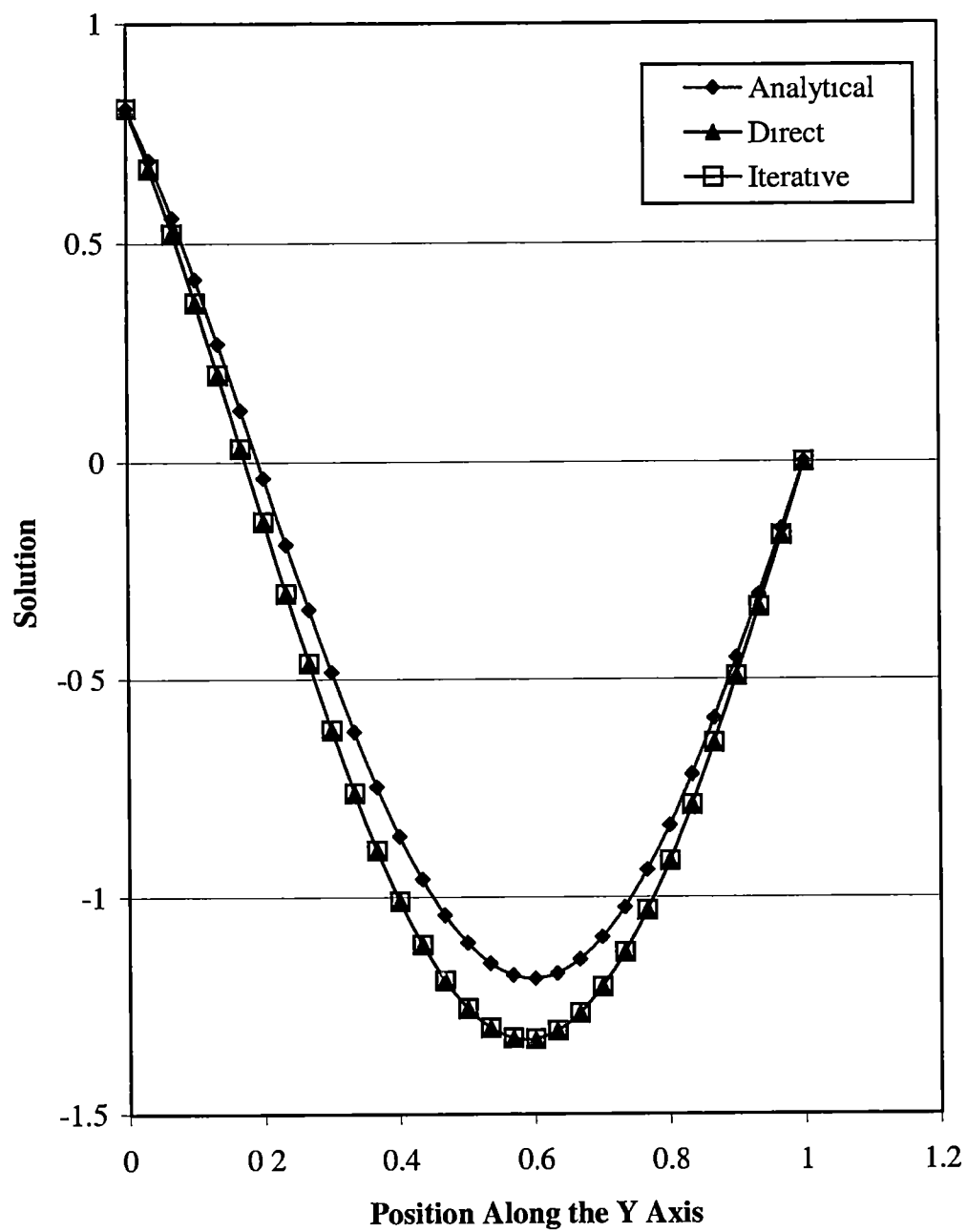
$$\begin{aligned} \frac{\partial \phi}{\partial x}(0, y) &= 0 & \frac{\partial \phi}{\partial x}(1, y) &= 0 \\ \phi(x, 0) &= \cos \pi x & \phi(x, 1) &= 0 \end{aligned} \tag{3.2}$$

Initially, the iterative solution produced by the preconditioned GMRES(t) method is compared to the solution obtained from a direct solver and to the analytical solution. The wavenumber k and the number of grid points in each direction, N_x , are held fixed at 5.0 and 31, respectively. The direct solver utilizes the band structure of the coefficient matrix A and is a part of the LAPACK [10] set of linear algebra routines. Cross sections of the solutions are obtained from each of the three solution methods with $y = 0.5$ shown in Figure 3.1.a and $x = 0.2$ in Figure 3.1.b. The first figure shows that the iterative and direct solutions differ most from the analytical solution at the boundaries ($x = 0$ and $x = 1$). The Neumann conditions defined on these boundaries are of lower



a. Cross section of solution with $y = 0.5$

Figure 3.1. 2-D. Comparison of solutions produced by the iterative method, the direct method, and the analytical method
 $N_x = 31$, $k = 5.0$



b. Cross section of solution with $x = 0.2$

Figure 3.1. (Continued)

accuracy than the discretizing method. The conditions have been defined so that a point $(0, y_1)$ is set equal to the point (δ, y_1) where $0 \leq y_1 \leq 1$ (the condition is similarly defined for $x = 1$). However, the solutions do exhibit the same solution pattern as the analytical solution and do agree better at points further from the boundaries. The second figure displays the solutions at the Dirichlet boundaries.

Additional verification of the preconditioned GMRES(t) method is provided in Figure 3.2. This figure displays a comparison between the direct solution and the iterative solution as the relative residual norm, ε , is decreased. The parameters k and N_x are again 5.0 and 31. In this figure, the error is the 2-norm of the difference between the direct and iterative solutions. As ε is decreased the iterative solution more closely approximates the direct solution. This further corroborates the findings in Figure 3.1 that the ILUT preconditioned GMRES(t) method is a valid solution technique for this problem.

The reason for using the ILUT preconditioning technique in conjunction with GMRES(t) is demonstrated in Figures 3.3 a and 3.3 b. N_x and k are still set equal to 31 and 5.0, respectively. From these figures, it is seen that the preconditioner significantly reduces the number of iterations and flops required for convergence

Now that the preconditioned iterative method has been shown to produce viable results, the method will now be tested for the three values of k mentioned previously: 5.0, 10.0, and 50.0. The parameters for ILUT are set with $\text{droptol} = 10^{-8}$ and $\text{lfil} = 4$. The results are displayed in Figures 3.4.a and 3.4.b. These figures demonstrate that for $k = 5.0$ and $k = 10.0$ the number of iterations and flops required for convergence increases with increasing N_x . The number of iterations and flops actually decreases with increasing

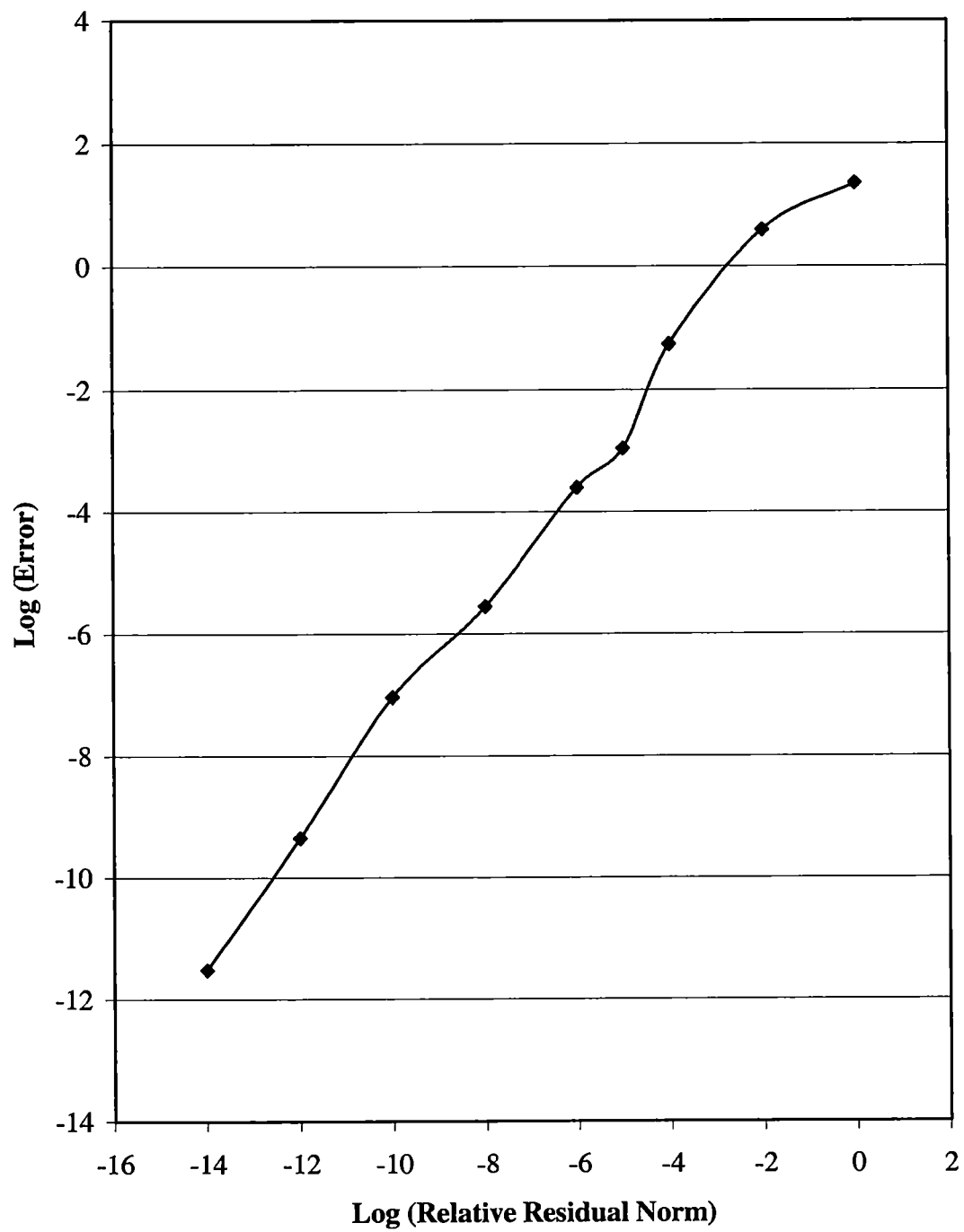
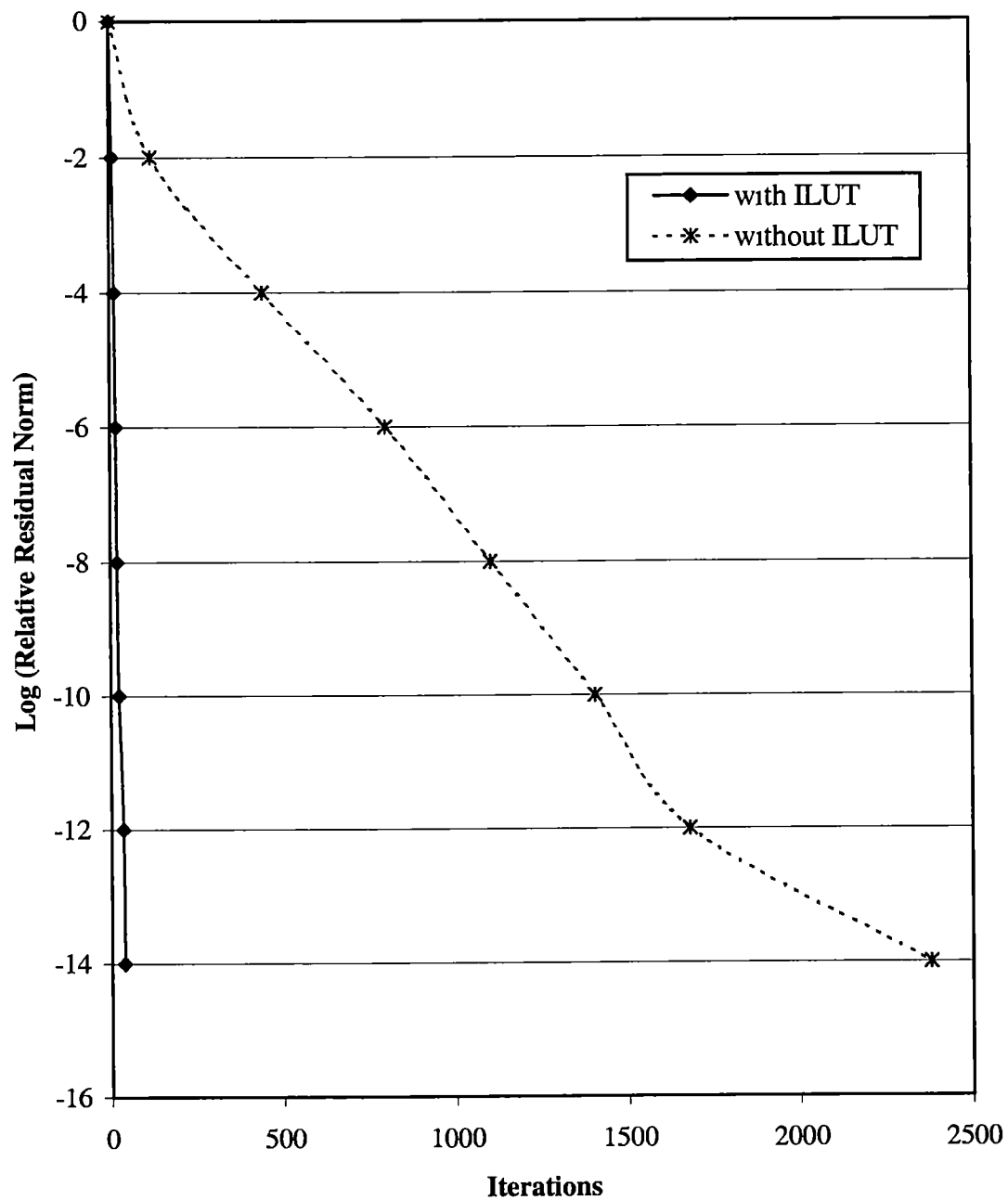
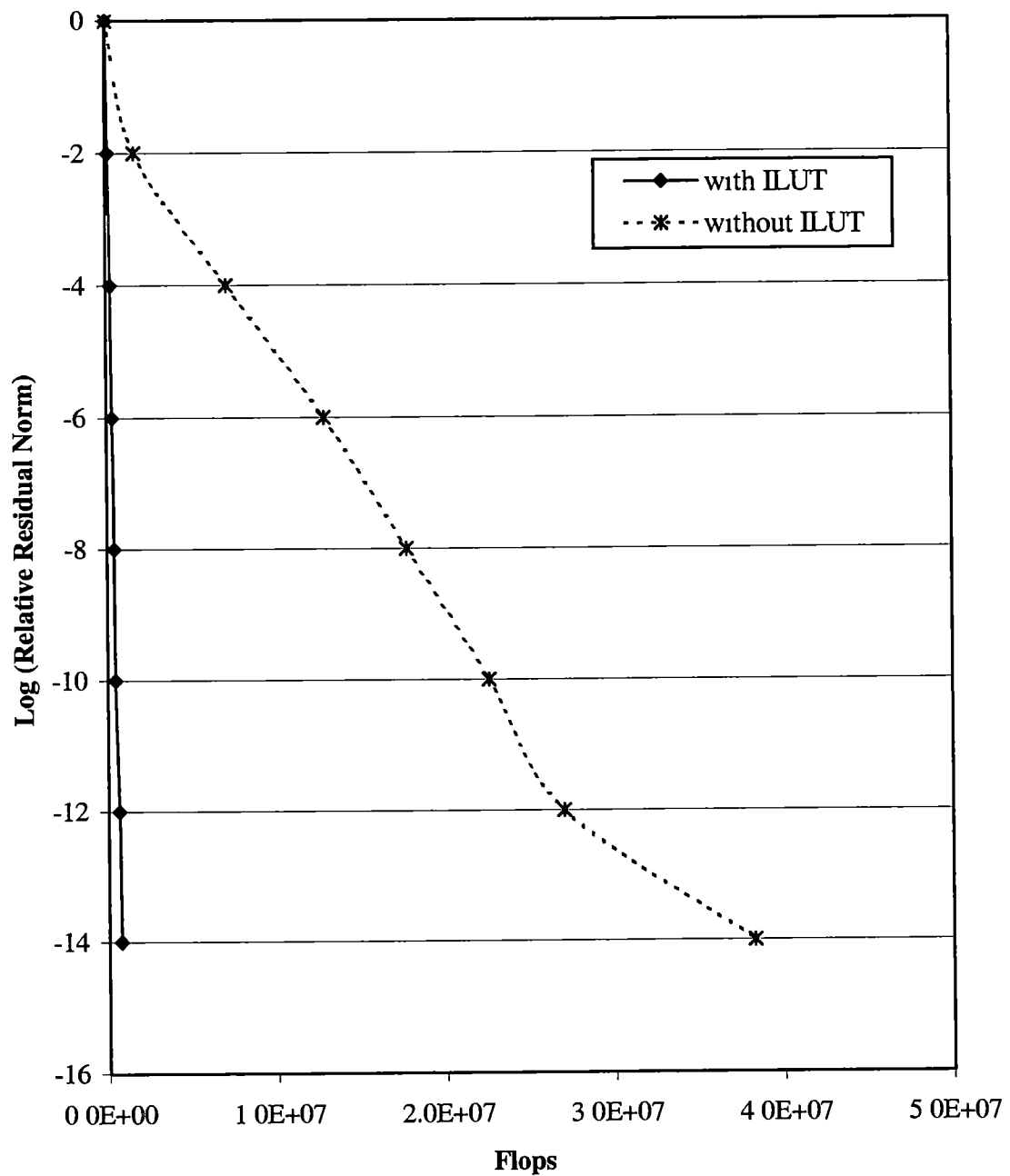


Figure 3.2. 2-D: Difference in solutions produced by the iterative method and the direct method
 $N_x = 31, k = 5.0$



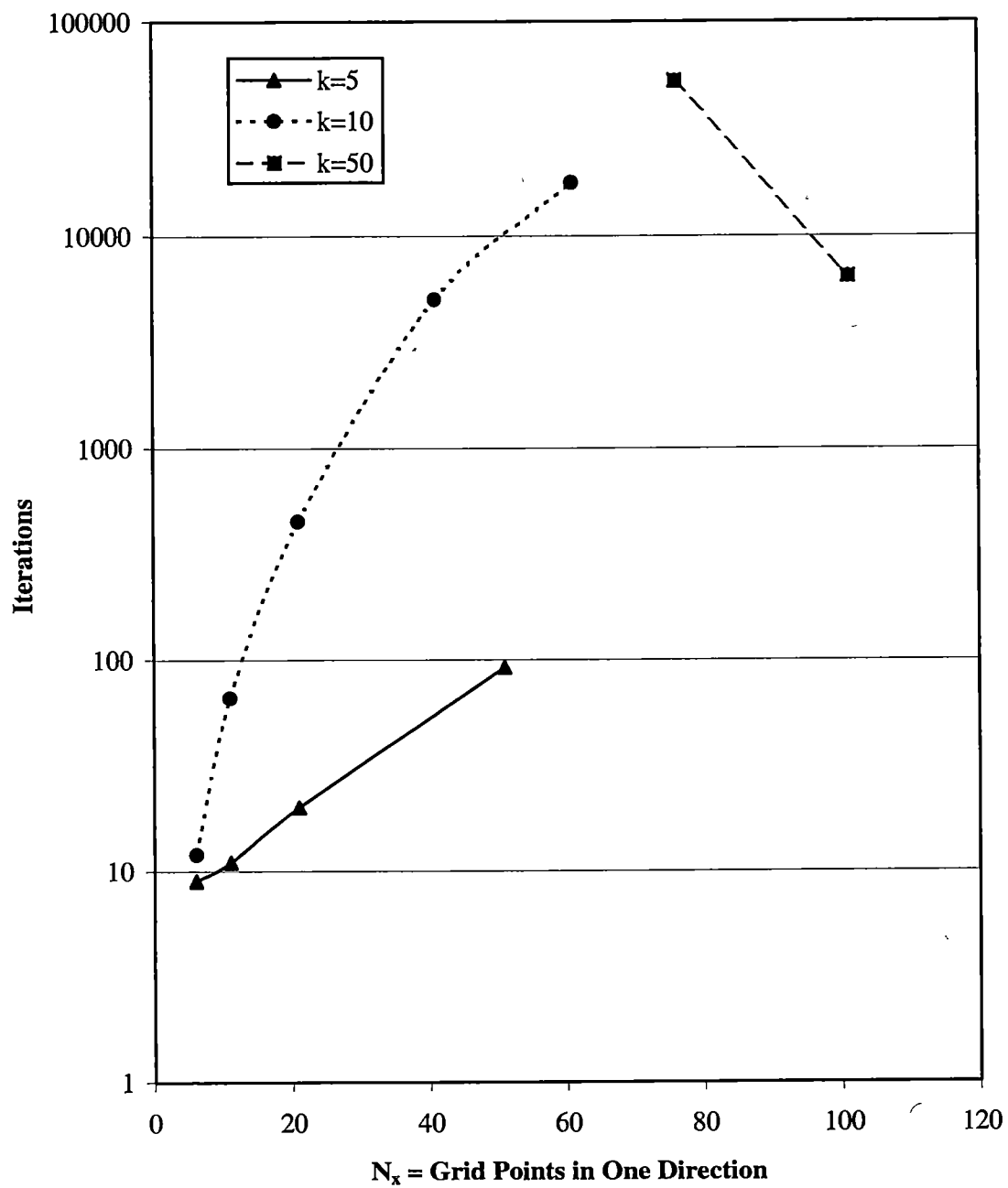
a. Log (relative residual norm) vs. number of iterations

Figure 3.3. 2-D: Comparison of preconditioned GMRES and non-preconditioned GMRES
 $N_x = 11, k = 5.0$



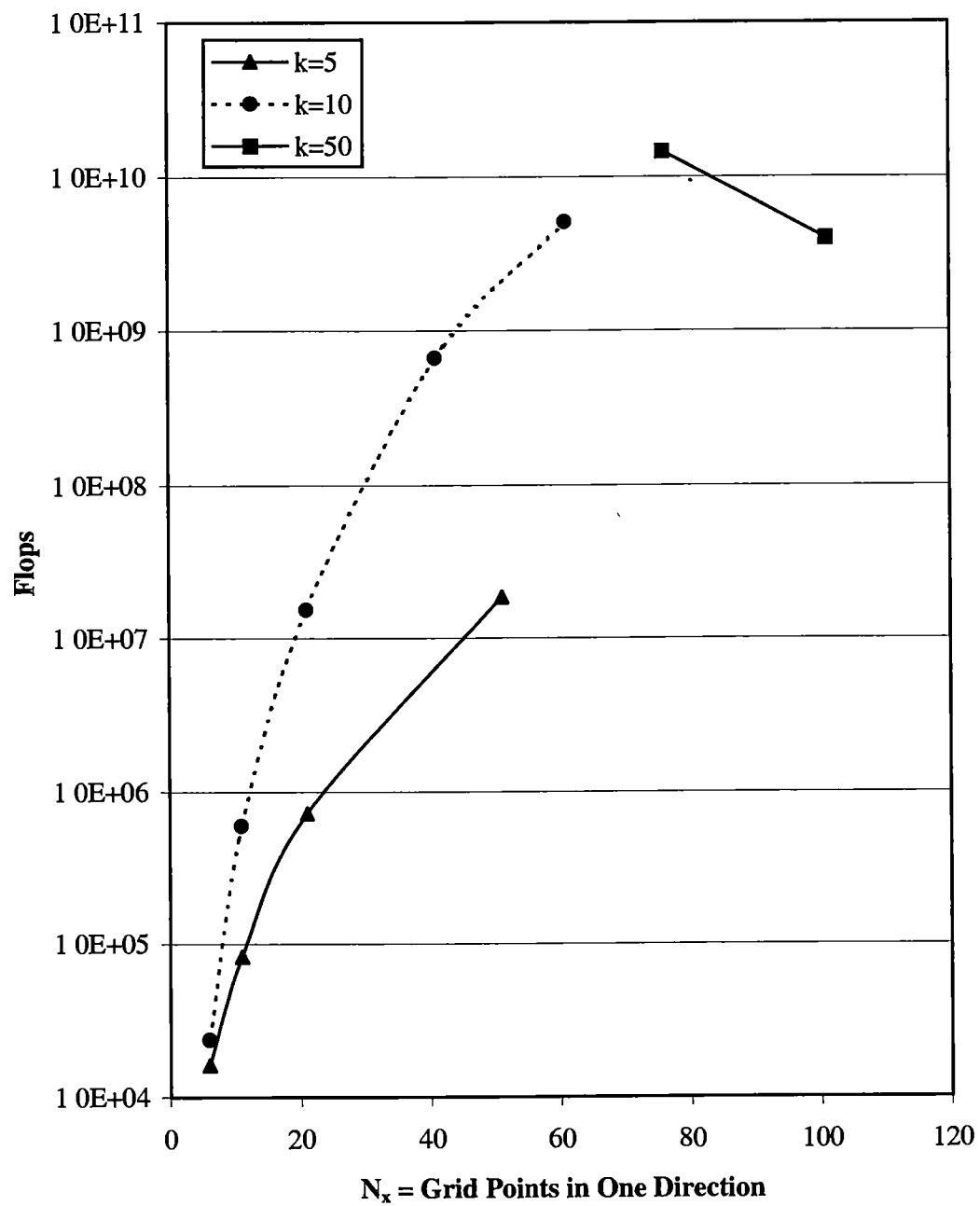
b. Log (relative residual norm) vs. number of flops

Figure 3.3. (Continued)



a. Number of iterations vs N_x

Figure 3.4. 2-D: Work required to reach convergence for different values of N_x and k



b. Number of flops vs. N_x

Figure 3.4. (Continued)

N_x for $k = 50.0$. Also, for $k = 50.0$, the solution process would not converge for $N_x = 26$ ($k\delta = 2$) or $N_x = 51$ ($k\delta = 1$).

As discussed previously, the number of operations required by a banded direct method is approximately $O(2N_x^4)$. However, this approximation is an underestimate of the actual number of operations necessary to solve the problem. In this research, the operations have been counted exactly for the preconditioned GMRES(t) method. Thus, a comparison of the two counts will not provide an accurate measure of the abilities of the iterative technique. The flops were not incorporated into the LAPACK direct routines because the direct solver is not the focus of this thesis. The operation count for smaller problems is not of as much interest in this thesis as it is for larger problems. It should therefore be noted that the band solver in LAPACK was unable to solve the problem with $N_x = 101$ due to the size of the arrays, although the operation count may not be prohibitive. This verifies the ability of the iterative solver to solve larger systems of equations than direct methods in 2-D.

The condition numbers, $\|A\| \|A^{-1}\|$, of the original coefficient matrix A are displayed in Table 3.1. All but one of the condition numbers are quite small. For $k = 10.0$ and $N_x = 6$ ($k\delta = 2$), the condition number is quite large, but the preconditioned iterative method was able to converge. The small size of the other condition numbers indicates that A is well-conditioned. Nonetheless, applying the preconditioner greatly reduces the computational work, as shown in Figures 3.3.a and 3.3.b.

Comparisons of the CPU times (in seconds) used by the direct and iterative methods to solve selected problems are displayed in Table 3.2. Overall, the direct solver times are smaller than the iterative solver times. However, it is more important with

Table 3.1. 2-D condition numbers

k	N_x	Condition Number
5.0	6	10 05
5.0	11	7.377
5 0	16	7 026
10.0	6	1.8×10^6
10.0	11	10.433
10.0	16	7.949

Table 3.2. 2-D direct and iterative CPU times (seconds)

k	N_x	Direct CPU Time	Iterative CPU Time
5.0	6	1.11×10^{-3}	4 27
5 0	51	7.14×10^{-3}	2.92
10.0	6	1.21×10^{-3}	5.44×10^{-3}
10 0	41	2.04	112.6
50.0	76	85 65	2400
50 0	101	Unable to Solve	3000

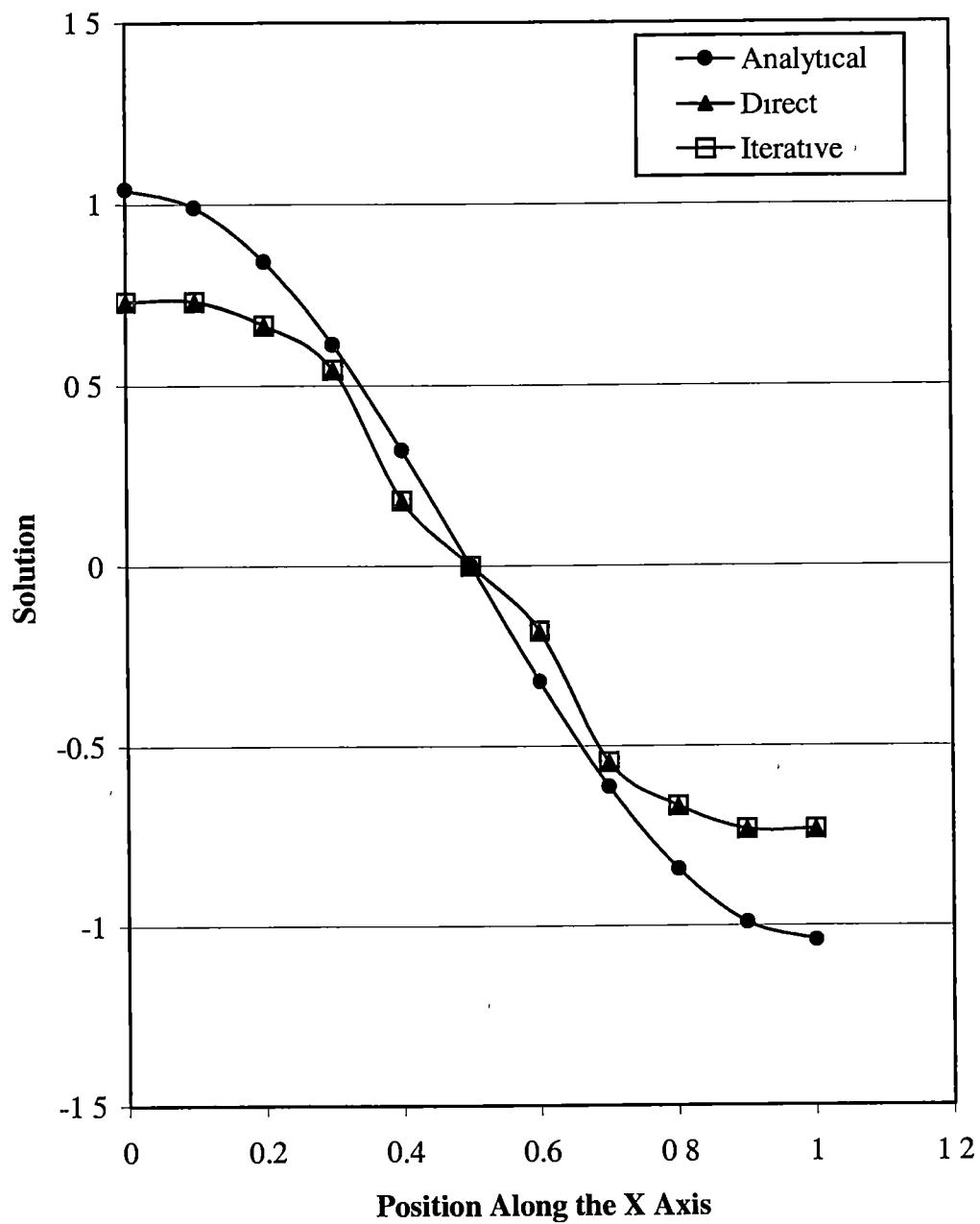
these relatively small problems that the iterative method is able to solve the $N_x = 101$ problem, which is not large, while the direct solver cannot.

Three Dimensional Test Problem #1

The practicality of the preconditioned GMRES method is again tested, this time in three dimensions. This problem is defined on a cube with $0 \leq x \leq 1$, $0 \leq y \leq 1$, and $0 \leq z \leq 1$. The boundary conditions are defined as follows

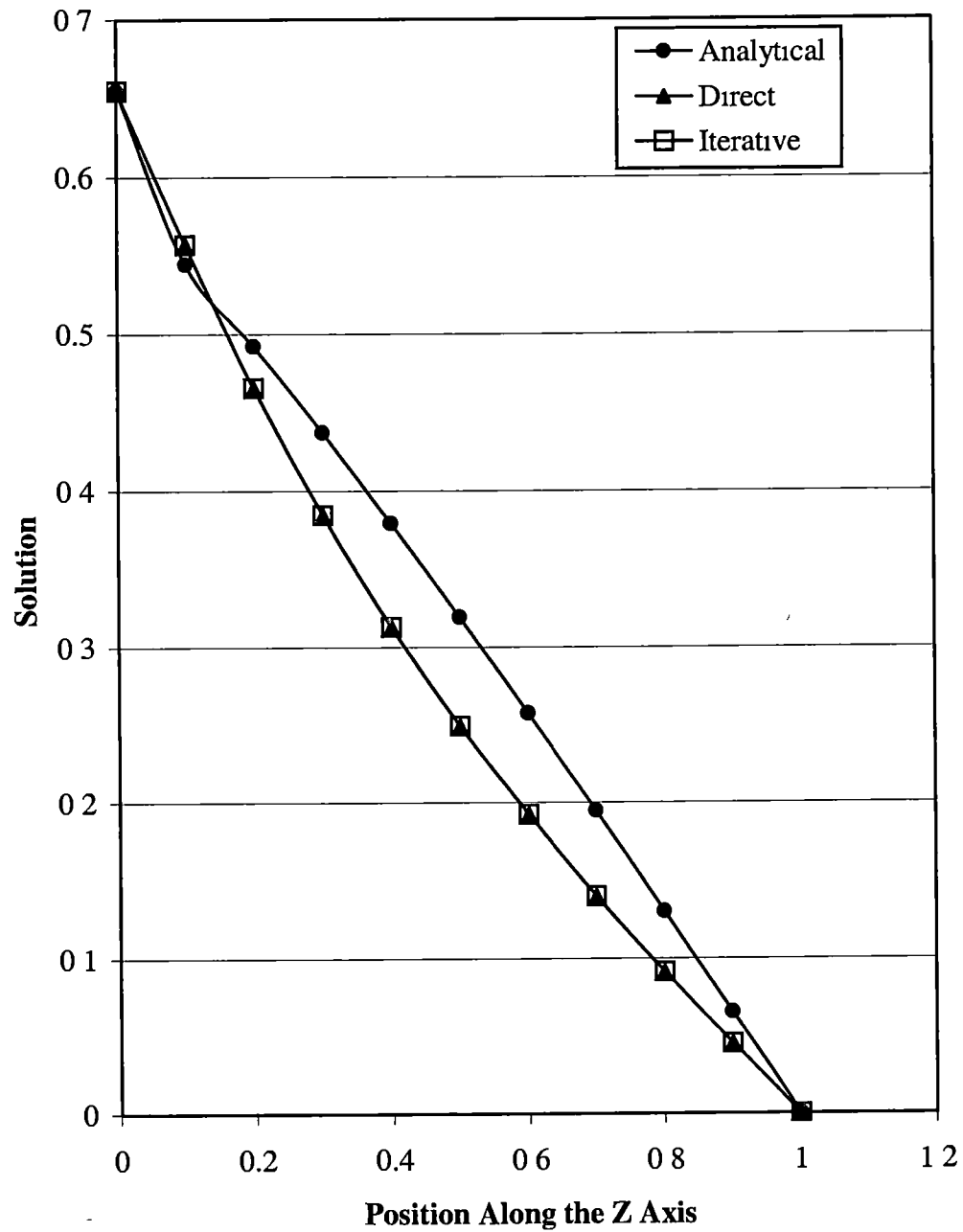
$$\begin{aligned} \frac{\partial \phi}{\partial x}(0, y, z) &= 0 & \frac{\partial \phi}{\partial x}(1, y, z) &= 0 \\ \frac{\partial \phi}{\partial y}(x, 0, z) &= 0 & \frac{\partial \phi}{\partial y}(x, 1, z) &= 0 \\ \phi(x, y, 0) &= \cos \pi x \cos \pi y & \phi(x, y, 1) &= 0 \end{aligned} \quad (3.3)$$

As was done in the 2-D test problem, the validity of the preconditioned GMRES(t) is first verified by comparing the iterative solution with the analytical and direct solutions. The results are shown in Figures 3.5.a and 3.5.b. The parameters k and N_x are held constant at values of 50 and 11, respectively. The first figure shows a cross section of the solution with $y = 0.2$ and $z = 0.2$. As with the 2-D problem, the iterative and direct solutions exhibit the greatest difference from the analytical solution at the boundaries. This is again due to the Neumann conditions. The iterative and direct solutions do approximate the analytical solution at points further from the boundaries. The second figure shows a cross section of the solutions with $x = 0.2$ and $y = 0.2$. The iterative and direct solutions approximate the analytical solution better in this case due to the Dirichlet conditions.



a. Cross section of solution with $y = 0.2$, $z = 0.2$

Figure 3.5. 3-D #1 Comparison of solutions produced by the iterative method, the direct method, and the analytical method
 $N_x = 11$, $k = 5.0$



b. Cross section of solution with $x = 0.2$, $y = 0.2$

Figure 3.5. (Continued)

The iterative solutions are further verified through comparison with the direct solution as the relative residual norm ε is decreased. Figure 3.6 displays these results, where the error is the 2-norm of the difference in the two solutions, $N_x = 11$, and $k = 5.0$. As expected, the iterative solution approaches the direct result as ε decreases.

A comparison between the preconditioned GMRES(t) method and the non-preconditioned GMRES(t) method is shown in Figures 3.7.a and 3.7.b. N_x is held constant at 16 and k is held at 5.0. As seen in the 2-D case, the preconditioning drastically improves the solution process in terms of the number of iterations and the number of floating point operations.

Now that the preconditioned iterative method has been shown to produce viable results, the method will be tested for the three values of k mentioned previously: 5.0, 10.0 and 50.0. The parameters for ILUT are set with $\text{droptol} = 10^{-4}$ and $\text{lfil} = 13$. The results are displayed in Figures 3.8.a and 3.8.b. These figures demonstrate that the number of iterations and flops required for convergence increases with increasing N_x and with k . For $k = 10.0$, the solution process would not converge for $N_x = 6$ ($k\delta = 2$) or $N_x = 11$ ($k\delta = 1$). The results for $k = 50.0$ are not displayed in the figure. As with the 2-D case, the solution process would not converge for $k = 50.0$ with $N_x = 26$ ($k\delta = 2$) or $N_x = 51$ ($k\delta = 1$). The only converged solution computed was for $N_x = 76$. This case required 1.19×10^{12} operations, whereas a direct method would require more than 2.93×10^{13} operations. However, a direct method would be unable to solve this problem, so the operation count notwithstanding, the iterative method still outperforms the direct method.

The operation counts for the iterative and direct solution methods cannot be

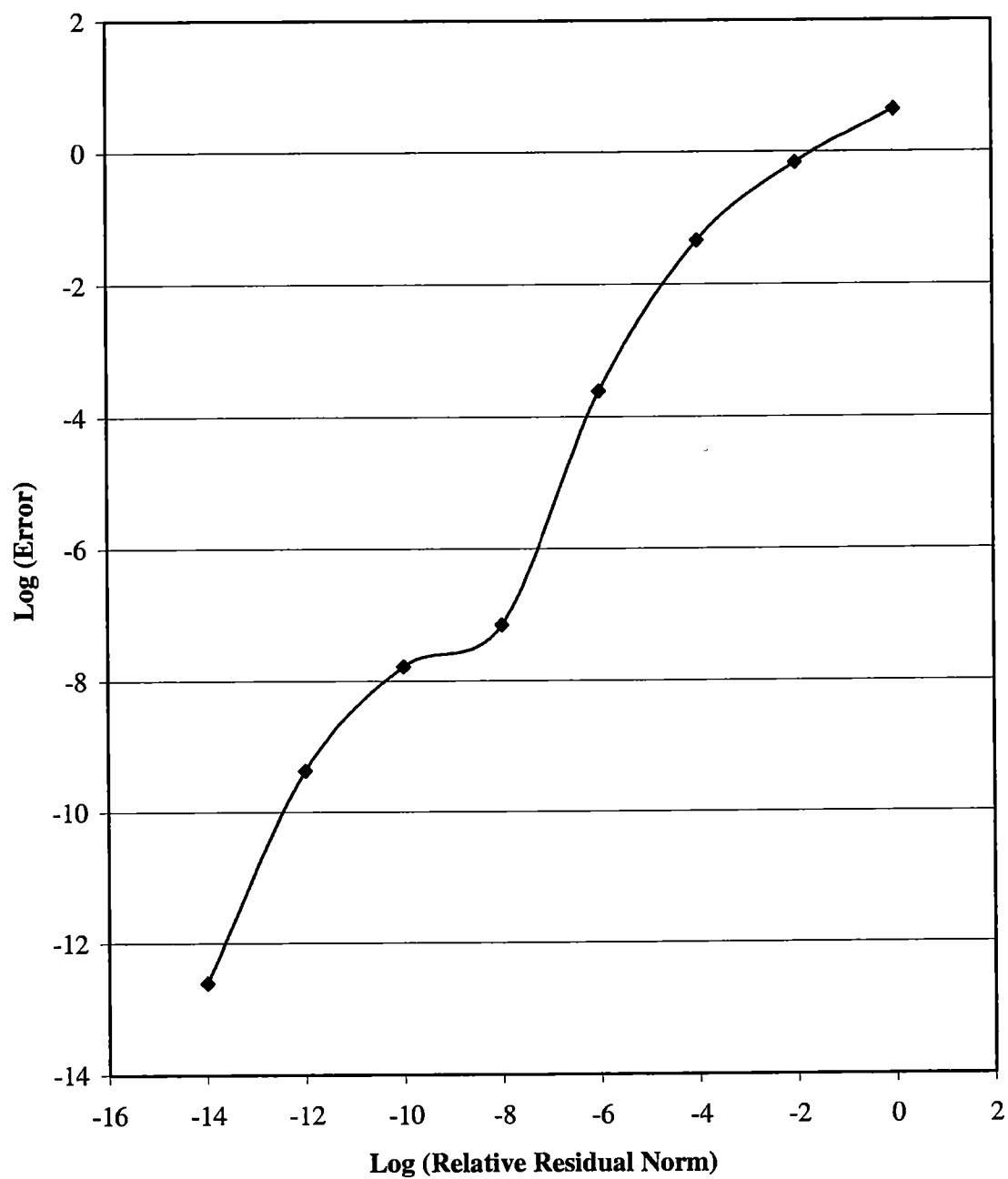
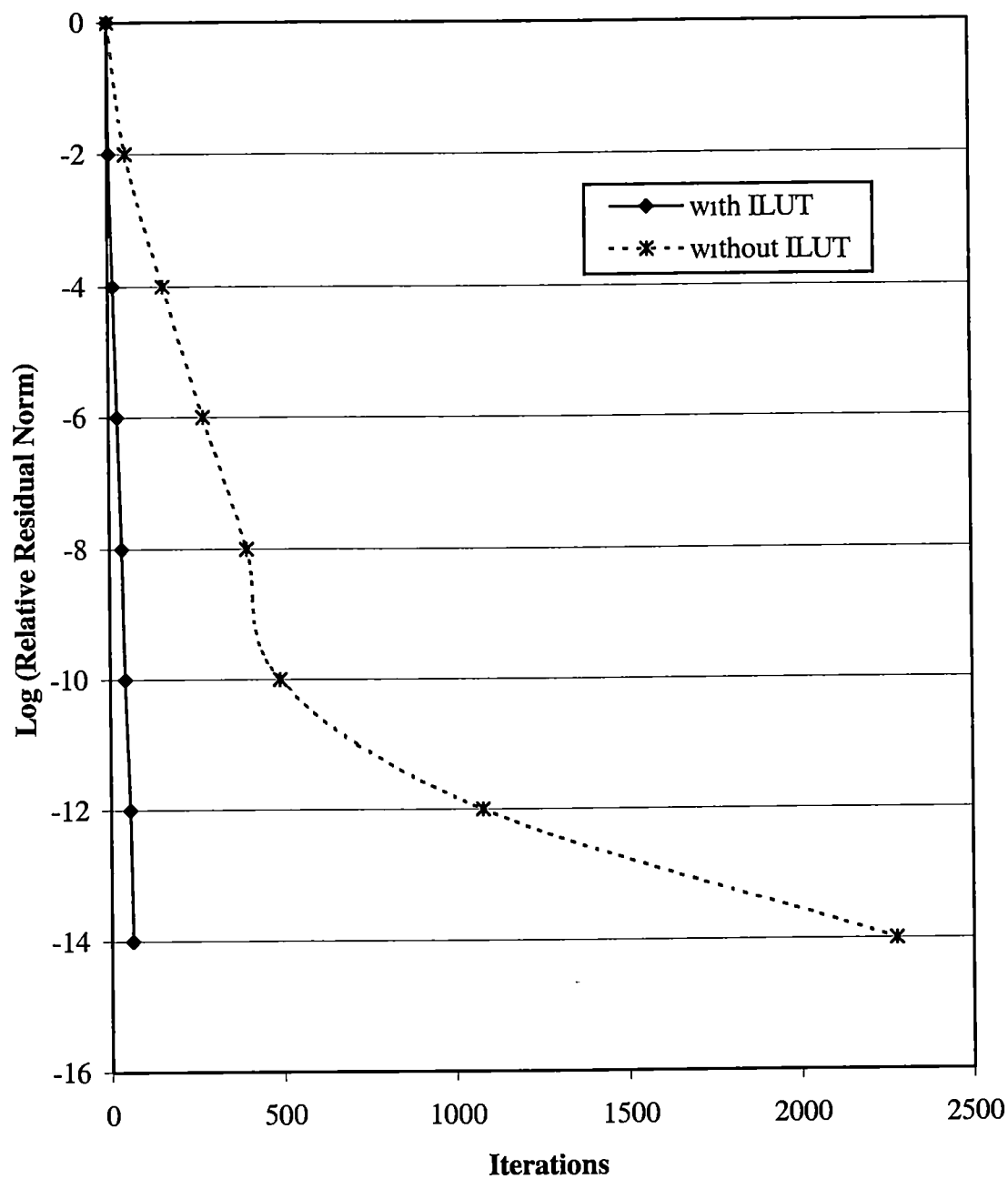
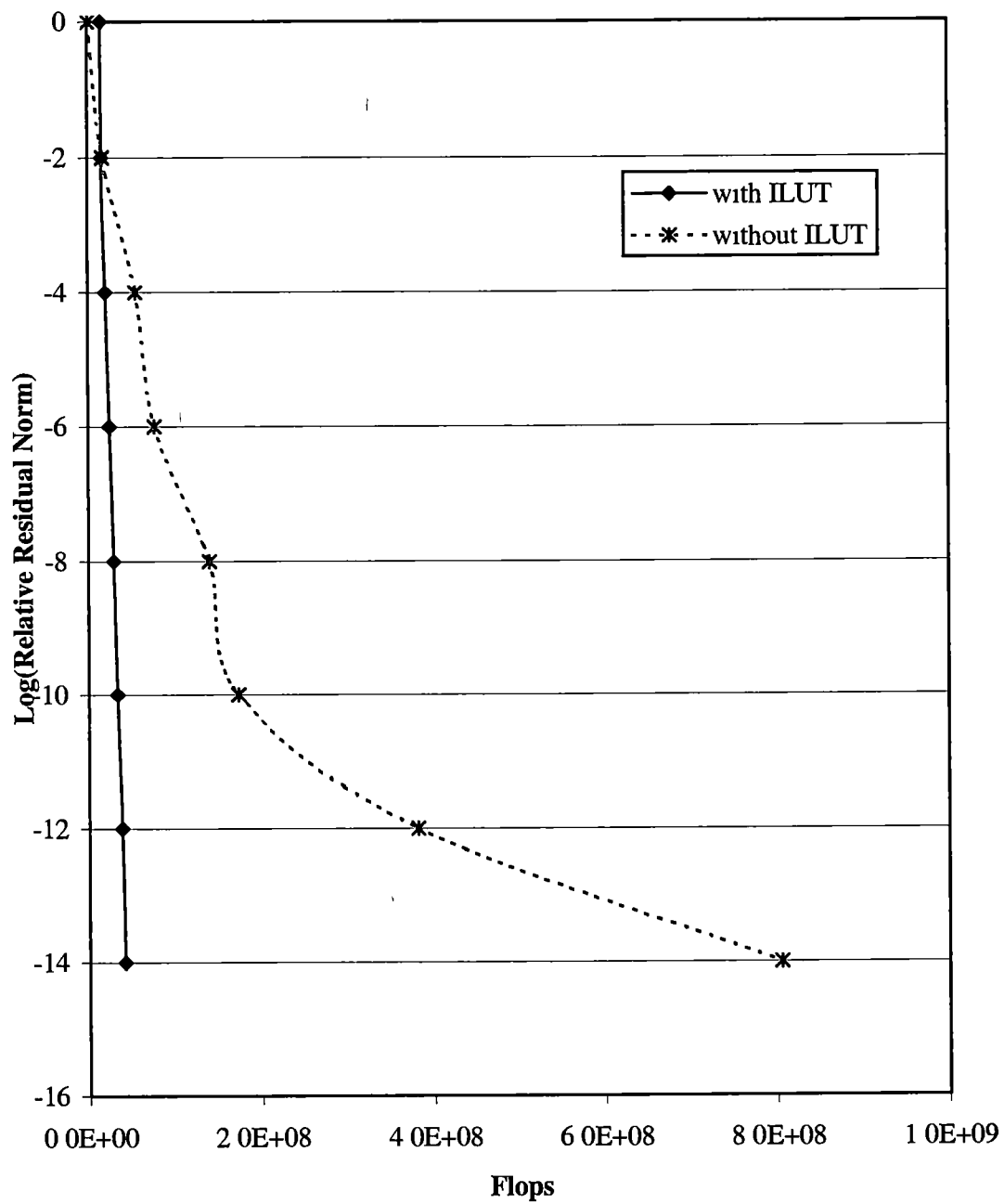


Figure 3.6. 3-D #1: Difference in solutions produced by the iterative method and the direct method
 $N_x = 11, k = 5.0$



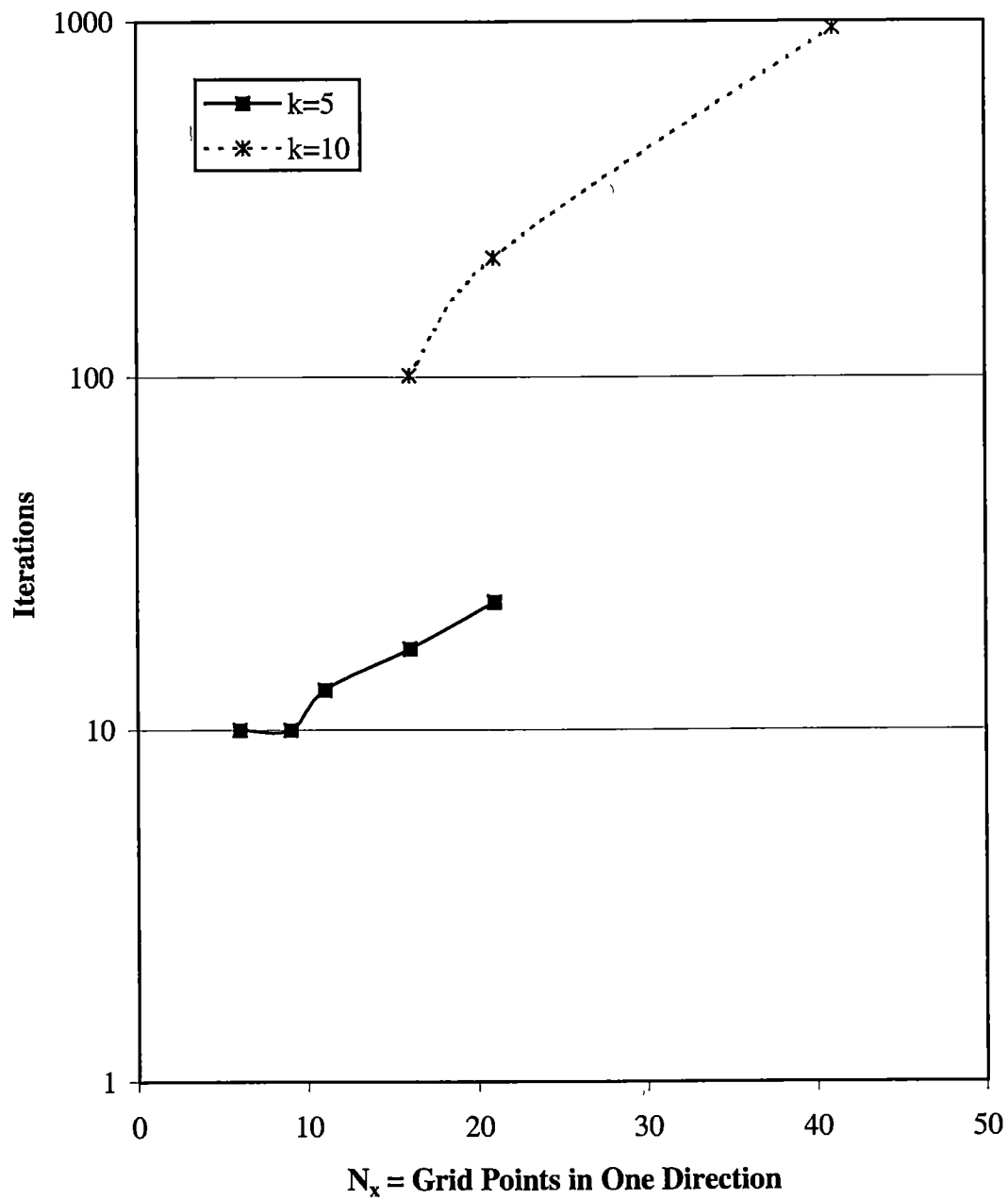
a. Log (relative residual norm) vs number of iterations

Figure 3.7. 3-D #1. Comparison of preconditioned GMRES and non-preconditioned GMRES
 $N_x = 11, k = 50$



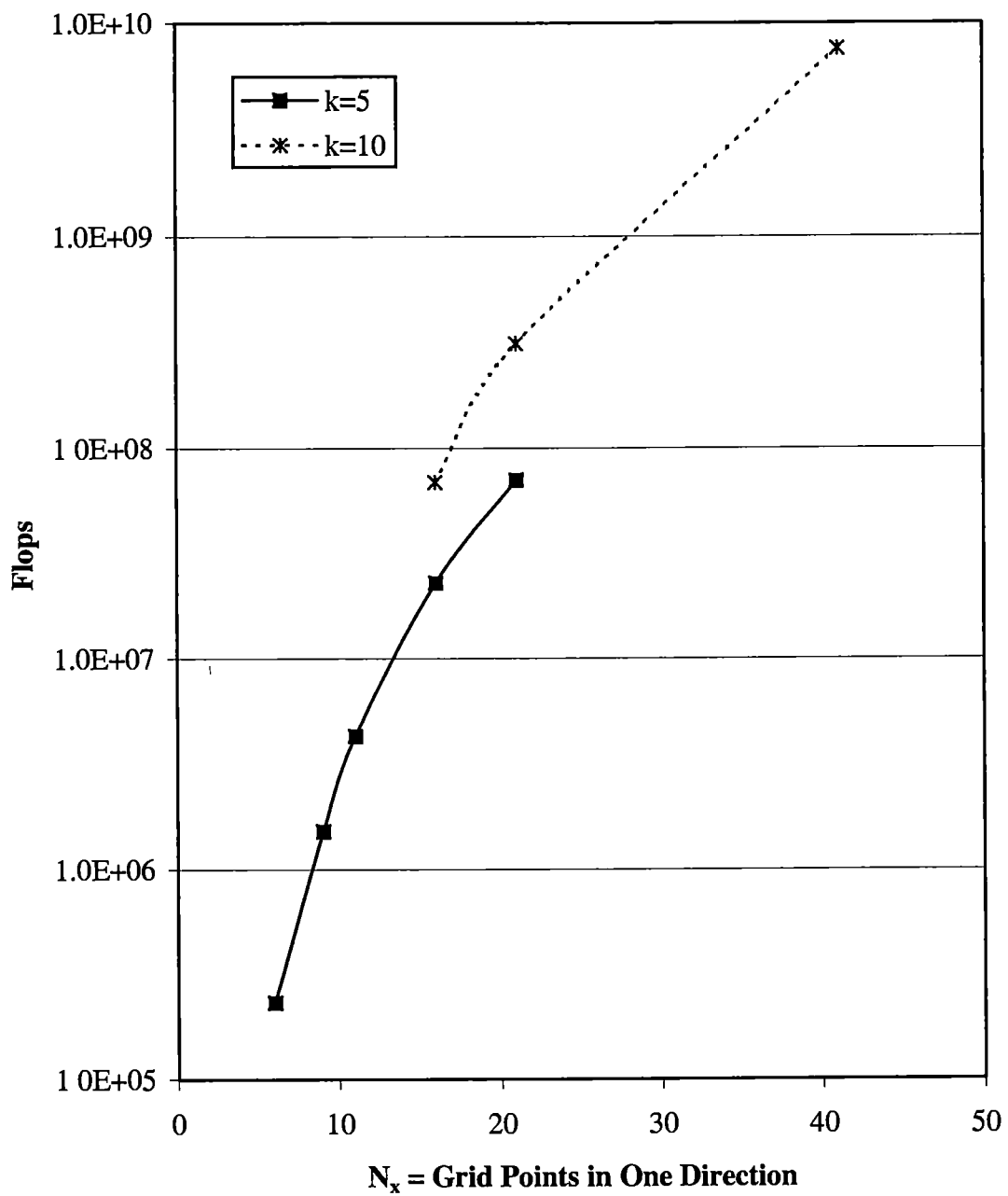
b. Log (relative residual norm) vs. number of flops

Figure 3.7. (Continued)



a. Number of iterations required for convergence

Figure 3.8. 3-D #1: Work required to reach convergence for different values of N_x and k



b. Number of flops required for convergence

Figure 3.8. (Continued)

accurately compared for reasons discussed in the 2-D test problem. However, since the direct operation count is an underestimate of the actual number of operations, the direct and iterative operation counts can be compared if the iterative count is larger than the underestimated direct count. For example, consider $k = 5.0$ and $N_x = 21$. The approximate direct operation count is greater than $2N_x^7 = 3.60 \times 10^9$, while the iterative solver requires only 7.06×10^7 operations. The iterative method continues to outperform the direct method for larger N_x . The iterative solver clearly solves the problem in fewer operations than the actual direct solver would be able to.

The condition numbers of the original matrix A are displayed in Table 3.3. As with the 2-D test problem, the condition number corresponding to $k\delta = 2$ ($k = 10$, $N_x = 6$) has the highest condition number. As previously noted, the iterative method did not reach convergence for the $k = 10$ with $N_x = 6$ or $N_x = 11$ case, although the original condition numbers are reasonable.

The CPU times for selected problems used by both the direct and the iterative solvers are shown in Table 3.4. The iterative method requires less computing time for $N_x = 16$. The direct method takes approximately the same amount of time to solve problems with the same N_x whereas the iterative method takes longer as k increases. The iterative method is unable to solve the $k = 10$ with $N_x = 6$ or $N_x = 11$ problems, but the direct method is capable of solving these problems. However, the direct method is incapable of solving the large $k = 50.0$ cases. Therefore, the iterative method is more viable for solving large systems of equations. The iterative solution times for the $k = 50.0$ case are large in comparison with the smaller problems, but are feasible.

Table 3.3. 3-D problem #1 condition numbers

k	N_x	Condition Number
5.0	6	12.44
5.0	11	9.96
5.0	16	9.61
10.0	6	317.94
10.0	11	12.64
10.0	16	10.37

Table 3.4. 3-D problem #1 direct and iterative CPU times

k	N_x	Direct CPU Time	Iterative CPU Time
5.0	6	1.35×10^{-2} sec.	4.99×10^{-2} sec.
5.0	16	99.01 sec.	6.62 sec.
10.0	6	1.71×10^{-2} sec	Unable to Solve
10.0	11	2.12 sec.	Unable to Solve
10.0	16	99.97 sec	20.54 sec.
50.0	76	Unable to Solve	42 hours

Three Dimensional Test Problem #2

The final test of the ILUT preconditioned GMRES(t) method is defined on a cube with $0 \leq x \leq 1$, $0 \leq y \leq 1$, and $0 \leq z \leq 1$. A sound wave of the form

$$\phi = \frac{e^{-ikr}}{4\pi r} \quad (3.4)$$

is chosen to emanate from a point source at the location (2,-1,-2). The quantity r is the distance between the grid point of consideration and the point (2,-1,-2). The three sides of the cube “facing” the point source are assigned Dirichlet conditions, i.e., the value of ϕ is defined on these sides by substituting the appropriate value of r into (3.4). The remaining three sides have radiation boundary conditions enforced, i.e. the strength of the wave at one of these boundary points is set equal to the strength of the wave at an adjacent point on the grid. This radiation boundary condition is very general and can be utilized in problems where the source ϕ is unknown since this condition does not require the knowledge of the wave's amplitude. Thus, changing the strength of the original wave does not affect the form of this boundary condition. For example, consider a grid point $(0, y_1, z_1)$ on the $x = 0$ face. The strength of the wave ϕ at this point is equal to the strength α of the wave at grid point (δ, y_1, z_1) . From the wave form of (3.4) we can write

$$\phi(\delta, y_1, z_1) = \alpha \frac{e^{-ikr_\delta}}{4\pi r_\delta}$$

and

$$\phi(0, y_1, z_1) = \alpha \frac{e^{-ikr_0}}{4\pi r_0}$$

where r_δ and r_0 are the distances from the points (δ, y_1, z_1) and $(0, y_1, z_1)$ to the point source. Eliminating α from the above equation, we obtain

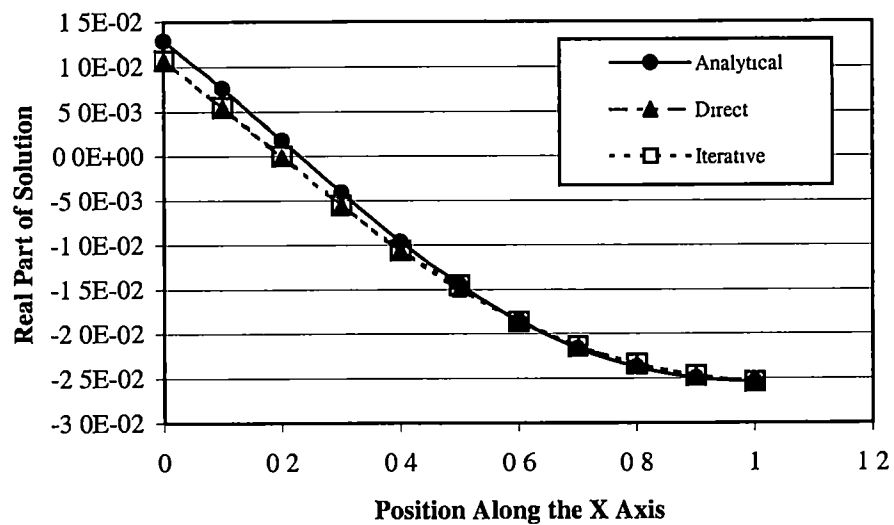
$$\phi(0, y_1, z_1) = \phi(\delta, y_1, z_1) \frac{4\pi r_\delta}{e^{-ikr_\delta}} \frac{e^{-ikr_0}}{4\pi r_0}$$

All the boundary points on the sides of the cube defined by these radiating conditions are represented in a similar manner.

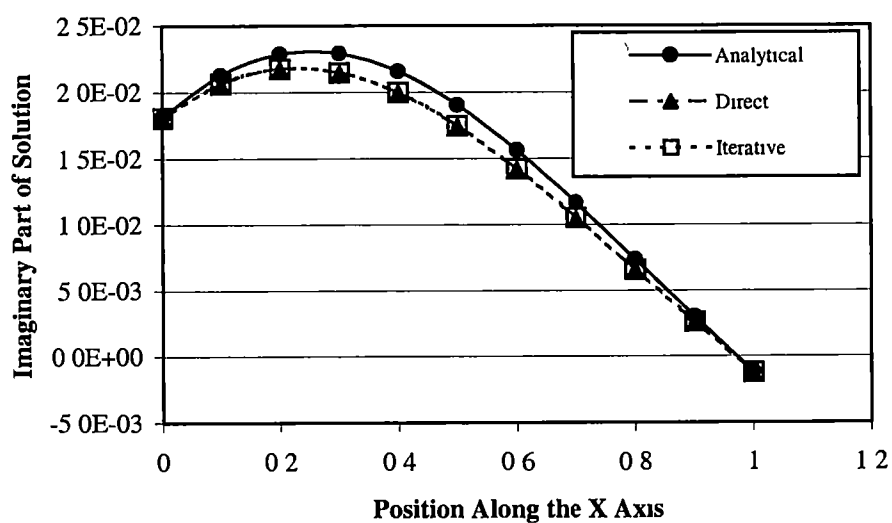
As was done with the previous two test problems, the iterative solution is first compared to the analytical and direct solutions. The results are shown in Figure 3.9. For simplicity, the real parts of the solutions are compared in Figure 3.9.a and the imaginary parts of the solutions are compared in Figure 3.9.b. The iterative and direct solutions agree with the analytical solution better than was seen in the previous two problems. The boundary conditions discussed for this problem are more accurate than the boundary conditions used in the other problems and therefore the iterative and direct solutions more accurately represent the analytical solution.

The iterative solution is now compared to the direct solution in Figure 3.10. This figure displays the 2-norm of the error (the difference in the two solutions) versus the relative residual norm ϵ . As noted in the previous test problems, the error decreases with decreasing ϵ . Notice that an ϵ of 10^{-5} corresponds to an error norm of 10^{-5} whereas for the previous two problems this value of ϵ corresponds to an error norm of 10^{-3} , which again substantiates the claim that the solution produced in this problem more accurately represents the exact solution.

The results obtained from the ILUT preconditioned GMRES(t) method are compared to the results obtained from the non-preconditioned GMRES(t) method in Figures 3.11.a and 3.11.b. As with the previous cases, the preconditioned method is



a. Real parts of the complex solutions



b. Imaginary parts of the complex solutions

Figure 3.9. 3-D #2: Comparison of solutions produced by the iterative method, the direct method, and the analytical method
 $N_x = 11$, $k = 5.0$, $y = 0.5$, $z = 0.5$

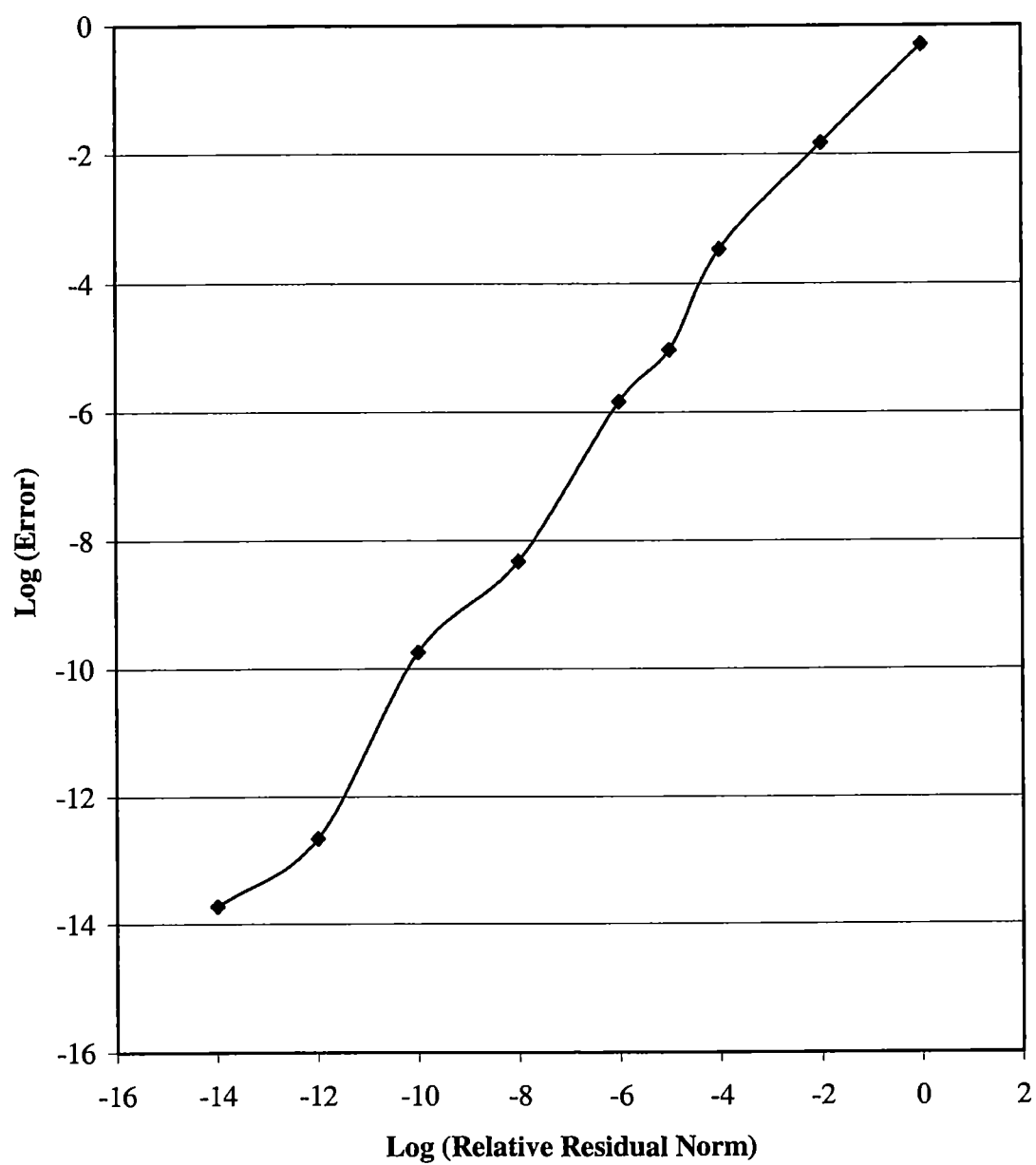
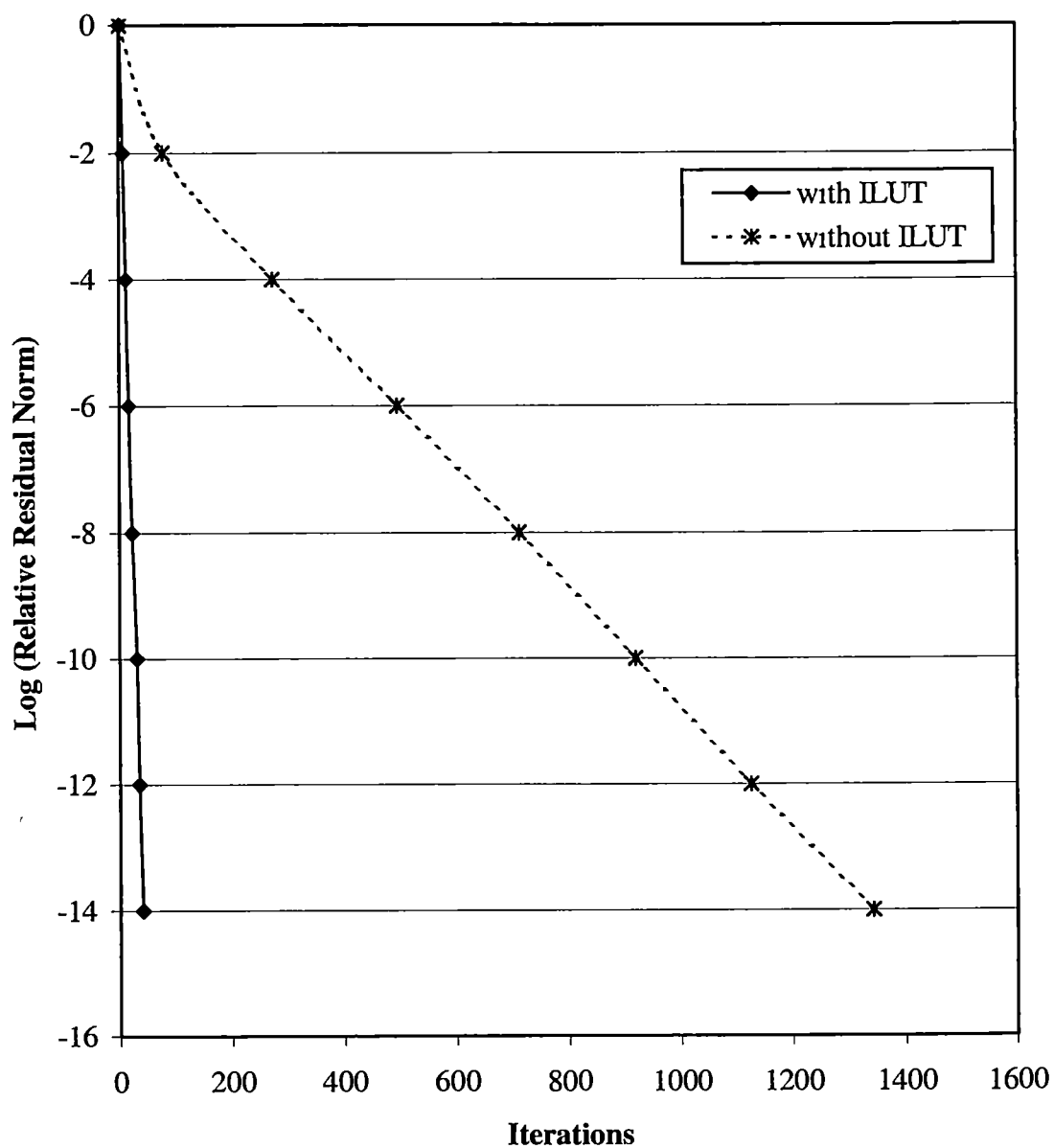
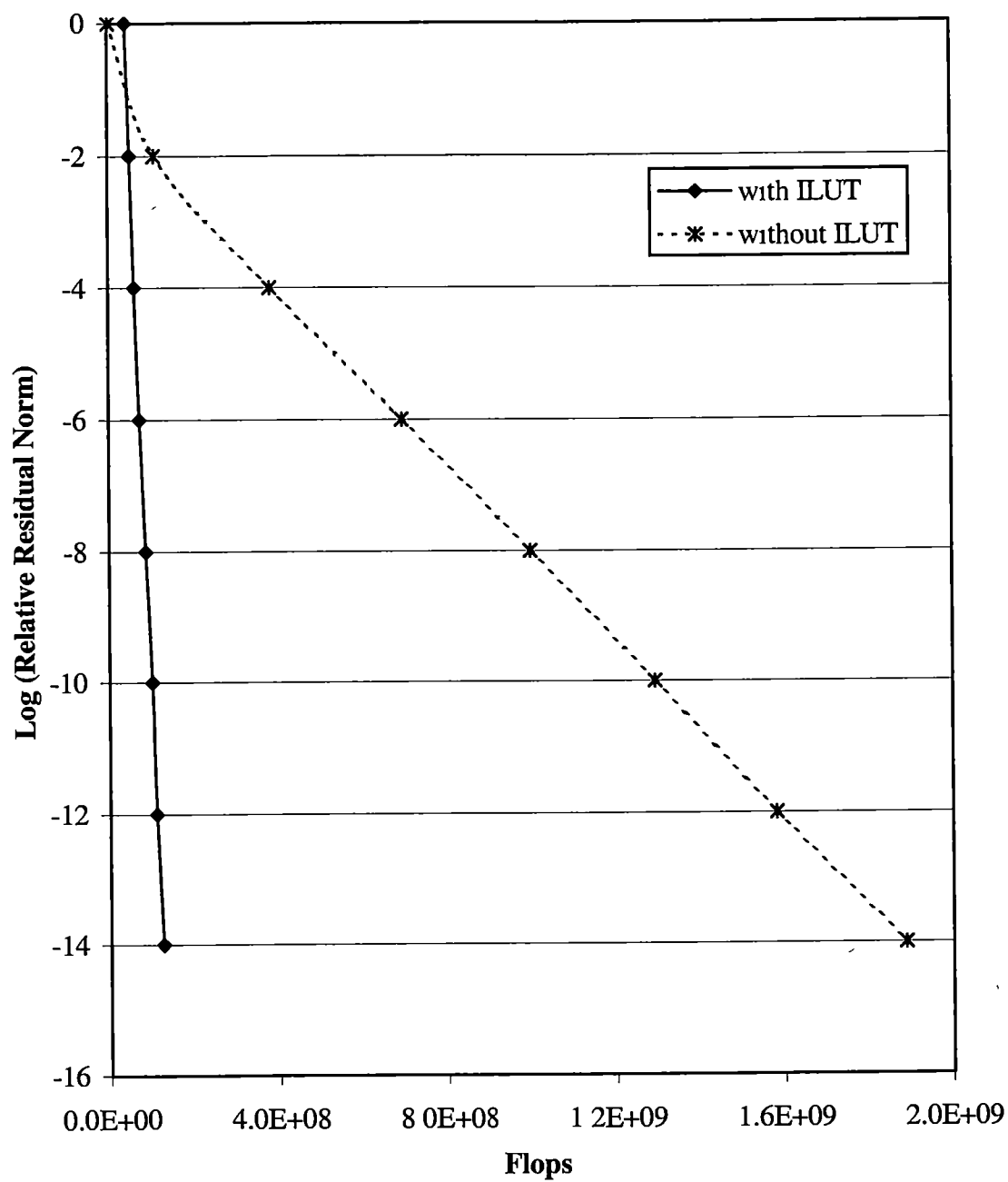


Figure 3.10. 3-D #2: Difference in solutions produced by the iterative method and the direct method
 $N_x = 11, k = 5.0$



a. Log (relative residual norm) vs number of iterations

Figure 3.11. 3-D #2: Comparison of preconditioned GMRES and non-preconditioned GMRES
 $N_x = 11, k = 50$



b. Log (relative residual norm) vs. number of flops

Figure 3.11. (Continued)

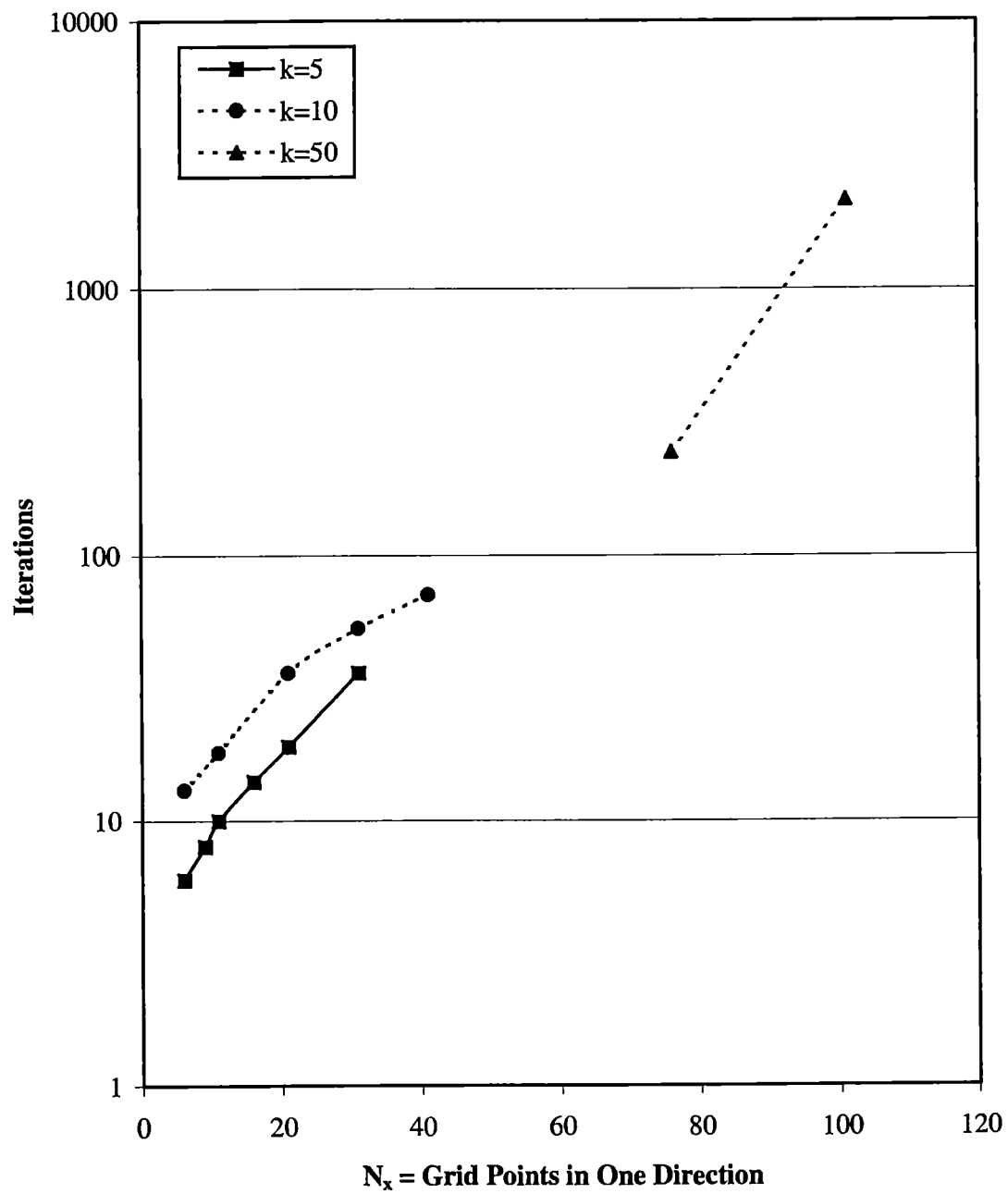
significantly more efficient than the non-preconditioned method. The preconditioning is therefore valid, as shown in the previous problems.

Since the ILUT preconditioned GMRES(t) method has been shown to produce viable results, the method is tested for the three values of k : 5.0, 10.0, and 50.0. The parameters for ILUT are set with $\text{droptol} = 10^{-4}$ and $\text{lfil} = 13$. The results are displayed in Figures 3.12.a and 3.12.b. These figures demonstrate that the number of iterations and flops required for convergence increases with increasing N_x and with increasing k . Again, as with the previous test problems, for $k = 50.0$, the solution process would not converge for $N_x = 26$ ($k\delta = 2$) or $N_x = 51$ ($k\delta = 1$)

As discussed in the first 3-D problem, the direct and iterative operation counts can be compared to obtain an idea of how well the iterative solution process performs. For example, consider $k = 5.0$ and $N_x = 11$. The direct operation count is greater than $8N_x^7 = 1.56 \times 10^8$, whereas the iterative operation count is only 1.19×10^7 . It can be seen that for larger N_x the iterative method continues to outperform the direct method

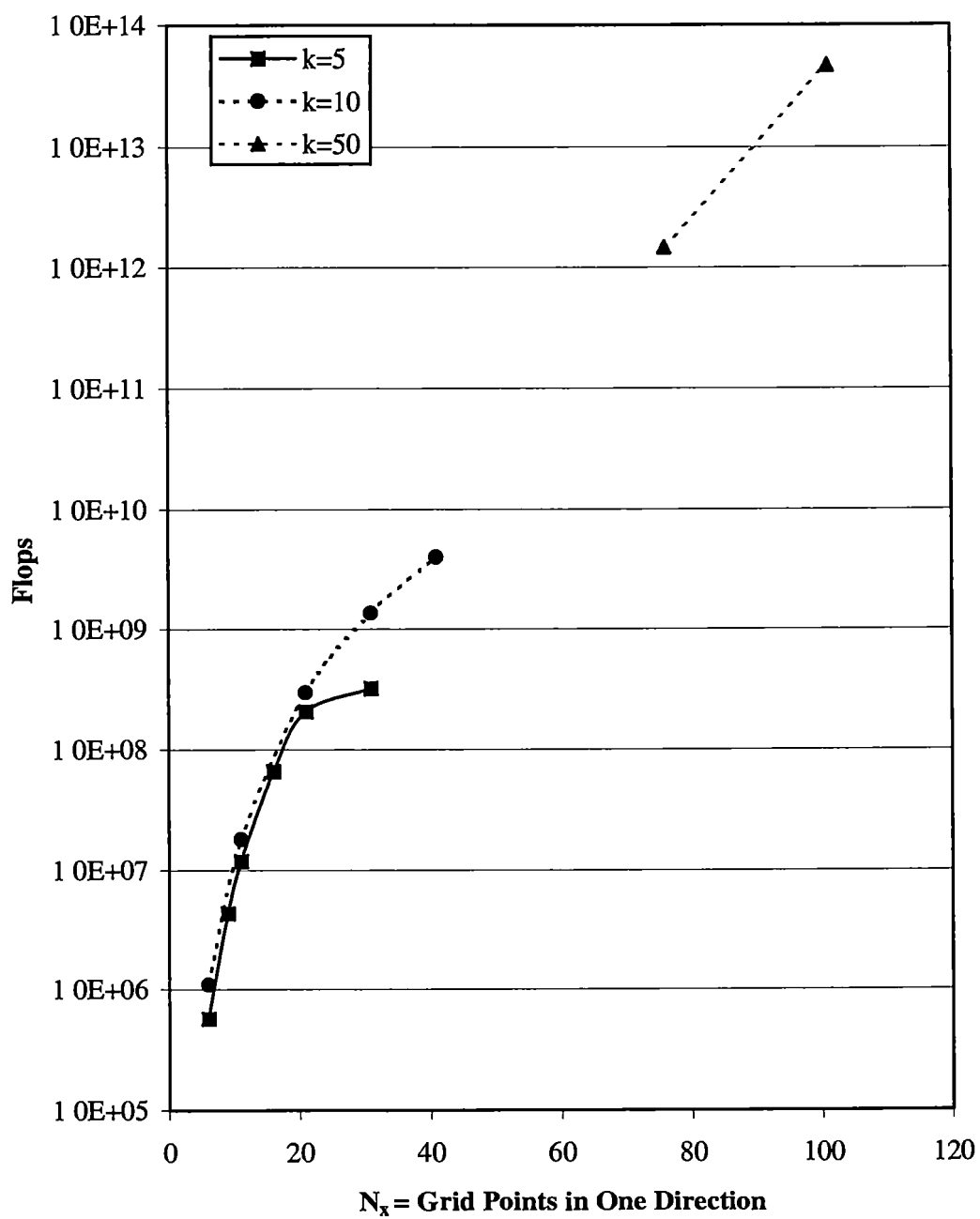
The condition numbers of the original matrix A are shown in Table 3.5. The condition numbers are all quite small, except for the condition number computed for $k = 10.0$ and $N_x = 11$ (although it is still reasonably small). This observation was also noted in the previous two problems. Again, the condition numbers do not provide adequate information for predicting whether the iterative technique will converge.

The CPU times required by the direct and iterative methods to solve specific problems are displayed in Table 3.6. The iterative method clearly solves the problems more quickly than the direct method. The direct method's solution time does not depend



a. Number of iterations required for convergence

Figure 3.12. 3-D #2: Work required to reach convergence for different values of N_x and k



b. Number of flops required for convergence

Figure 3.12. (Continued)

Table 3.5. 3-D problem #2 condition numbers

k	N_x	Condition Number
5 0	6	10 42
5 0	11	9 34
5 0	16	9.20
10.0	6	224.20
10 0	11	10.67
10 0	16	9.716

Table 3.6. 3-D problem #2 direct and iterative CPU times

k	N_x	Direct CPU Time	Iterative CPU Time
5 0	6	1.07×10^{-2} sec.	8.60×10^{-2} sec.
5 0	21	152.96 sec	36 75 sec
10.0	6	1.07×10^{-2} sec.	1.2×10^{-1} sec
10.0	21	152 73 sec.	55.13 sec.
50 0	76	Unable to Solve	4320 sec.
50.0	101	Unable to Solve	18 hours

on k , only on N_x , whereas the iterative method's solution time depends on both parameters. The iterative CPU times for $k = 50.0$ are very reasonable, and are actually smaller than the times computed for the previous 3-D $k = 50.0$ problem. The fact that the iterative method outperforms the direct method in this complex-valued 3-D problem is noteworthy due to the real-world applications of this problem.

Chapter 4

CONCLUSIONS

It has been shown that the ILUT preconditioned GMRES(t) method efficiently solves large systems of equations. Specifically, the iterative method was able to solve a complex-valued 3-D problem with over one million variables. Direct methods are unable to solve such large problems. In the 2-D and 3-D problems it was shown that the iterative method is capable of solving larger problems than a direct solver can handle. The iterative method was further proven to outperform direct methods in both 3-D cases with a smaller operation count and computing time than the direct method for larger problems.

It was also observed that for $k = 50.0$ the solution process did not converge for $N_x = 26$ ($k\delta = 2$) or for $N_x = 51$ ($k\delta = 1$) in all three test problems. As the value of k is increased, the higher the frequency. Thus, more waves need to be discretized and perhaps these values of $k\delta$ are not small enough to adequately describe the problems.

Although the ILUT preconditioned GMRES(t) method has been shown to be a viable method for solving large systems of equations, there are perhaps other preconditioners that could improve the solution time even more than ILUT. The preconditioner considered here is very general and a preconditioner designed specifically for the problems discussed here could perform better. These solution times could further be improved by incorporating parallel computing capabilities into the solution process. This may also enable one to solve larger problems than those presented here. Both of

these possible improvements are beyond the scope of this thesis, however, and provide suggestions for possible future research.

LIST OF REFERENCES

LIST OF REFERENCES

- [1] Trefethen, L.N. and Bau, D., *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.
- [2] Caruthers, J.E., French, J.C. and Raviprakash, G.K., "Green Function Discretization for Numerical Solution of the Helmholtz Equation," *Journal of Sound and Vibration*, Volume 187(4), pp 553-568, 1995.
- [3] Saad, Y., *SPARSKIT: a basic tool kit for sparse matrix computations*, Version 2, University of Minnesota, Minneapolis, MN, June 6, 1994.
- [4] Saad, Y., private communication, 1999.
- [5] Saad, Y., *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, MA, 1996.
- [6] Saad, Y. and Schultz, M.H., "GMRES: A Generalized Minimum Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM J. Sci. Stat Comput.*, Volume 7(3), pp 856-869, 1986.
- [7] Caruthers, J.E., Steinhoff, J.S., Engels, R.C., "An Optimal Finite Difference Representation for a Class of Linear PDE's with Application to the Helmholtz Equation," *Journal of Computational Acoustics*, Vol. 7(4), pp 245-252, 1999.
- [8] Caruthers, J.E., private communication, 2000.
- [9] Golub, G.H., Van Loan, C.F., *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1996.
- [10] Anderson, E., et al, *Lapack User's Guide*, SIAM, Philadelphia, PA, 1995.

APPENDICES

APPENDIX 1

Coefficients of the Green's Function Discretization

The expressions for the 2-D coefficients a_1 and a_2 are found in [7]. They are referenced here for convenience. The coefficients can be written as follows

$$a_1 = \frac{-2J_0(k\delta\sqrt{2})^2 - J_0(2k\delta)J_0(k\delta\sqrt{2}) - J_0(k\delta\sqrt{2}) + 2J_0(k\delta)^2 + 2J_0(k\delta\sqrt{5})J_0(k\delta)}{D}$$

$$a_2 = \frac{-2J_0(k\delta\sqrt{2})J_0(k\delta\sqrt{5}) - J_0(k\delta\sqrt{2})J_0(k\delta) + J_0(2k\delta\sqrt{2}) + J_0(k\delta) + 2J_0(2k\delta)J_0(k\delta)}{D}$$

where

$$D = 2J_0(2k\delta\sqrt{2})J_0(k\delta\sqrt{2}) + J_0(2k\delta\sqrt{2}) + J_0(2k\delta)J_0(2k\delta\sqrt{2}) + 1 + 3J_0(2k\delta) - 8J_0(k\delta\sqrt{5})J_0(k\delta) - 4J_0(k\delta)^2 + 2J_0(k\delta\sqrt{2}) - 4J_0(k\delta\sqrt{5})^2 + 4J_0(2k\delta)J_0(k\delta\sqrt{2}) + 2J_0(2k\delta)^2$$

The expressions for the 3-D coefficients were obtained from J. E. Caruthers [8] and are referenced here for convenience. The coefficients can be derived as follows.

$$D = \begin{matrix} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,1} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,1} & d_{3,1} & d_{3,2} & d_{3,3} \end{matrix}$$

where

$$d_{1,1} = 1 + 4 \frac{\sin \sqrt{2}k\delta}{\sqrt{2}k\delta} + \frac{\sin 2k\delta}{2k\delta}$$

$$d_{1,2} = 4 \left(\frac{\sin k\delta}{k\delta} + \frac{\sin \sqrt{3}k\delta}{\sqrt{3}k\delta} + \frac{\sin \sqrt{5}k\delta}{\sqrt{5}k\delta} \right)$$

$$d_{1,3} = 4 \left(\frac{\sin \sqrt{2}k\delta}{\sqrt{2}k\delta} + \frac{\sin \sqrt{6}k\delta}{\sqrt{6}k\delta} \right)$$

$$d_{2,1} = 2 \left(\frac{\sin k\delta}{k\delta} + \frac{\sin \sqrt{3}k\delta}{\sqrt{3}k\delta} + \frac{\sin \sqrt{5}k\delta}{\sqrt{5}k\delta} \right)$$

$$d_{2,2} = 1 + 4 \frac{\sin \sqrt{2}k\delta}{\sqrt{2}k\delta} + 2 \frac{\sin 2k\delta}{2k\delta} + 4 \frac{\sin \sqrt{6}k\delta}{\sqrt{6}k\delta} + \frac{\sin 2\sqrt{2}k\delta}{2\sqrt{2}k\delta}$$

$$d_{2,3} = 2 \frac{\sin k\delta}{k\delta} + 2 \frac{\sin 3k\delta}{3k\delta} + 4 \frac{\sin \sqrt{5}k\delta}{\sqrt{5}k\delta}$$

$$d_{3,1} = 3 \left(\frac{\sin \sqrt{2}k\delta}{\sqrt{2}k\delta} + \frac{\sin \sqrt{6}k\delta}{\sqrt{6}k\delta} \right)$$

$$d_{3,2} = 3 \frac{\sin k\delta}{k\delta} + 3 \frac{\sin 3k\delta}{3k\delta} + 6 \frac{\sin \sqrt{5}k\delta}{\sqrt{5}k\delta}$$

$$d_{3,3} = 1 + 3 \frac{\sin 2k\delta}{2k\delta} + 3 \frac{\sin 2\sqrt{2}k\delta}{2\sqrt{2}k\delta} + \frac{\sin 2\sqrt{3}k\delta}{2\sqrt{3}k\delta}$$

Using these expressions for the elements of D , the coefficients a_1 , a_2 , and a_3 can be computed in the following manner.

$$\begin{aligned}
a_1 &= \frac{\sin k\delta}{k\delta} \\
a_2 &= [D]^{-1} \frac{\sin \sqrt{2}k\delta}{\sqrt{2}k\delta} \\
a_3 &= \frac{\sin \sqrt{3}k\delta}{\sqrt{3}k\delta}
\end{aligned}$$

These exact expressions for the coefficients can be used for $k\delta \geq 1$ [8]. The following Taylor expansion for the coefficients is valid for $k\delta < 1$.

$$\begin{aligned}
a_1 &= \frac{7}{64} + \frac{433}{18432}(k\delta)^2 + \frac{34063889}{9385279488}(k\delta)^4 + \frac{131409172619}{256781246791680}(k\delta)^6 \\
&\quad + \frac{64230230484929023}{915242316564623532840}(k\delta)^8
\end{aligned}$$

$$\begin{aligned}
a_2 &= \frac{3}{128} + \frac{229}{36864}(k\delta)^2 + \frac{101041417}{93852794880}(k\delta)^4 + \frac{82333213799}{513562493583360}(k\delta)^6 \\
&\quad + \frac{205966291111437191}{9152423165646235238400}(k\delta)^8
\end{aligned}$$

$$\begin{aligned}
a_3 &= \frac{1}{128} + \frac{29}{12288}(k\delta)^2 + \frac{13892369}{31284264960}(k\delta)^4 + \frac{11851659439}{171187497861120}(k\delta)^6 \\
&\quad + \frac{1596783655386853}{160568827467477811200}(k\delta)^8
\end{aligned}$$

APPENDIX 2

FORTRAN Codes

The FORTRAN codes utilized in this research for the complex three dimensional test problem are provided here. The original complex GMRES(t) code was provided by Y. Saad [4] and the author of this thesis incorporated preconditioning into the code. The codes provided in SPARSKIT v. 2 [3] for the ILUT and various other subroutines were altered to handle complex valued problems

```
c This is the complex restarted GMRES code that needs to be preceded by
c an ILU preconditioner  It utilizes common blocks
c so that large problems may be solved.
      subroutine pgmrez(eps,maxits,iout,flopgm)
      include 'param h'
      common/BLK1/a(nzmax), ja(nzmax), ia(n+1)
      common/BLK2/alu(1wk), jlu(1wk), ju(n)
      common/BLKV/vv(n,1m+1)
      common/BLKR/rhs(n)
      common/BLKB/sol(n)
      common/BLKH/hh(51,50),c(50),s(50),rs(51)
      integer maxits,iout, ja, ia, jlu, ju
      real*8 eps
      complex*16 a, vv,sol,rhs, alu

c-----
c complex gmres algorithm . simple version .  (May 23,1985)
c altered for preconditioning by Julie R. Spangler (August, 1999)
c-----
c parameter list.
c n      == size of problem
c 1m     == size of krylov subspace:  should not exceed 50 in this
c        version (can be reset in code. looking at comment below)
c rhs    == right hand side
c sol    == initial guess on input, approximate solution on output
c vv     == work space of size n x (1m+1); stores basis vectors of
c        Krylov subspace
c eps    == tolerance for stopping criterion. process is stopped
c        as soon as ( ||.|| is the euclidean norm):
c        || current residual||/||initial residual|| <= eps
c maxits== maximum number of iterations allowed
c iout   == output unit number number for printing intermediate results
```

```

c          if (iout le 0) no statistics are printed.
c hh == Hessenberg matrix created by the Arnoldi algorithm

```

```

c-----
c subroutines used:

```

```

c camux1, camux2, zaxpy1, zaxpy2

```

```

c functions used:

```

```

c zdot1, znrm2, znrm2a

```

```

c-----
      implicit none
      integer ih, flopgm, jj
      parameter (ih = 50)
      complex*16 hh,c,s,rs,zdot1,dconjg,tz
      real*8 ro, t, gam, epsmac, eps1, znrm2a, znrm2b
      integer*4 n, n1, j, its, iout, k, k1, i, i1,i1
      integer flopgm

```

```

c-----
c      arnoldi size should not exceed immax in this version..

```

```

c      to reset modify parameter instruction accordingly.

```

```

c-----
      data epsmac/1.d-16/
      n1 = n + 1

```

```

      its = 0

```

```

      flopgm = 0

```

```

c      outer loop starts here. -compute initial residual
10      continue

```

```

      call camux1 (1)
      do 21 j=1,n
        vv(j,1) = rhs(j) - vv(j,1)

```

```

21      continue
      FLOPGM = FLOPGM + 2*N

```

```

c      compute norm of r0
20      ro = znrm2a(1)
      if (ro eq. 0.0d0) return
      tz = dcmlpx(1.0d0/ro,0.0d0)
      FLOPGM = FLOPGM + 1
      do 210 j=1, n

```

```

        vv(j,1) = vv(j,1)*tz

```

```

210      continue
      FLOPGM = FLOPGM + 6*N

```

```

      if (its eq. 0) eps1=eps*ro

```

```

      if (iout .gt. 0) write(iout, 199) its, ro

```

```

c      initialize 1-st term of rhs of hessenberg system..

```

```

      rs(1) = dcmlpx(ro,0.0d0)

```

```

      i = 0

```

```

4      i=i+1

```

```

      its = its + 1

```

```

      i1 = i + 1

```

```

      call clusol1 (i)

```

```

      call camux2 (i1)

```

```

c      modified gram - schmidt...

```

```

      do 55 j=1, i

```

```

        tz = zdot1(j,i1)

```

```

        FLOPGM = FLOPGM + 8*n

```

```

        hh(j,i) = tz
        call zaxpy1(-tz,j,11)
        FLOPGM = FLOPGM + 8*n
55    continue
        t = znrm2a(11)
        FLOPGM = FLOPGM + 5*n
        hh(11,i) = dcmlpx(t)
        tz = dcmlpx(1.0d0/t,0.0d0)
        do 57 k=1,n
            vv(k,11) = vv(k,11)*tz
57    continue
        FLOPGM = FLOPGM + 6*N + 1
c     done with modified gram schimd and arnoldi step..
c     now update factorization of hh
c
        if (i .eq. 1) goto 121
c----- apply previous transformations to i-th column of h
        do 66 k=2,11
            k1 = k-1
            tz = hh(k1,1)
            hh(k1,1) = dconjg(c(k1))*tz + s(k1)*hh(k,1)
            hh(k,1) = -s(k1)*tz + c(k1)*hh(k,1)
            FLOPGM = FLOPGM + 28
66    continue
121   gam=dsqrt(cdabs(hh(i,1))**2 + cdabs(hh(11,1))**2)
        if (gam .eq. 0.0d0) gam = epsmac
c----- determine next plane rotation
        tz = dcmlpx(1.0d0/gam , 0.0d0)
        c(i) = hh(1,1)*tz
        s(1) = hh(11,1)*tz
        rs(11) = -s(1)*rs(1)
        rs(1) = dconjg(c(1))*rs(1)
c----- determine residual norm and test for convergence-
        hh(1,1) = dconjg(c(1)) *hh(1,1) + s(1)*hh(11,1)
        ro = cdabs(rs(11))
        FLOPGM = FLOPGM + 43
c
        if (1out gt. 0) write(1out, 199) its, ro
        if (1 .lt. 1m .and. (ro .gt. eps1)) goto 4
c
c     compute solution. first solve upper triangular system.
c
        rs(i) = rs(i)/hh(1,1)
        FLOPGM = FLOPGM + 11
        do 30 11=2,11
            k=11-11+1
            tz=rs(k)
            do 40 j=k+1,11
                tz = tz-hh(k,j)*rs(j)
                FLOPGM = FLOPGM + 8
40        continue
            rs(k) = tz/hh(k,k)
            FLOPGM = FLOPGM + 11
30    continue
c     done with back substitution..

```

```

c      now form linear combination to get solution
      tz=rs(1)
      do 15 k=1,n
        rhs(k) = vv(k,1)*tz
15     continue
      FLOPGM = FLOPGM+6*n

      do 16 j=2, 1
        tz = rs(j)
        call zaxpy2(tz,j)
        FLOPGM = FLOPGM+8*n
16     continue
c
      call clusol2
      do 17 k=1,n
        sol(k) = sol(k) + rhs(k)
17     continue
      FLOPGM = FLOPGM + 2*n
c      compute residual vector and continue...
c
      do 24 j=1,i
        jj = 11-j+1
        rs(jj-1) = -s(jj-1)*rs(jj)
        rs(jj) = dconjg(c(jj-1))*rs(jj)
        FLOPGM = FLOPGM+12
24     continue
c
      do 25 j=1,11
        tz = rs(j)
        if (j .eq. 1) then
          tz = tz - (1.0d0,0.0d0)
        endif
        call zaxpy1 (tz, j, 1)
        FLOPGM = FLOPGM + 8*n
25     continue
c
199  format('   its =', i4, ' res. norm =', d20.6)
c-----restart if necessary...
      if (ro .gt. eps1 .and. its .lt. maxits) goto 20
      return
      end
c-----end-of-pgmrez -----

      subroutine zaxpy1(t,r,p)
      include 'param.h'
      common/BLKV/vv(n,im+1)
      complex*16 vv,t
      integer r,p
      do 1 k=1,n
        vv(k,p)=vv(k,p)+vv(k,r)*t
1     continue
      return
      end

      subroutine zaxpy2(t,p)

```

```

include 'param.h'
common/BLKV/vv(n,im+1)
common/BLKR/x(n)
complex*16 vv, t, x
integer p, k
do 1 k=1, n
    x(k) = x(k) + vv(k,p)*t
1 continue
return
end

function zdot1(p,r)
include 'param.h'
common/BLKV/vv(n,im+1)
complex*16 zdot1, t, vv
integer p, r
c computes the inner product t=(x,y)
t = (0.0d0,0.0d0)
do 10 k=1, n
    t = t + dconjg(vv(k,p))*vv(k,r)
10 continue
zdot1=t
return
end

function znrm2(m)
include 'param.h'
common/BLKC/x(n)
complex*16 x
real*8 znrm2, t
    m = n
t = 0.0d0
do 10 j=1, n
    t=t+cdabs(x(j))**2
10 continue
znrm2 = dsqrt(t)
return
end

function znrm2a(p)
include 'param.h'
common/BLKV/vv(n,im+1)
complex*16 vv
real*8 znrm2a, t
integer p, j
c computes the inner product t=(x,y)
t = 0.0d0
do 10 j=1, n
    t = t + cdabs(vv(j,p))**2
10 continue
znrm2a = dsqrt(t)
return
end

```

```

C-----
C
C               S P A R S K I T
C
C-----
C
C               ITERATIVE SOLVERS MODULE
C
C-----
C
C This Version Dated: August 13, 1996. Warning: meaning of some
C
C ===== arguments have changed w.r.t. earlier versions. Some
C
C               Calling sequences may also have changed
C
C-----
C
C Contents: I am making the ILUT complex. All altered
C
C programs have a 'C' starting their name to denote complex program.
C
C   Julie R. Spangler (August, 1999)
C
C
C CILUT      Incomplete LU factorization with dual truncation strategy
C
C
C
C-----sample-accelerator-and-LU-solvers-----
C
C
C
C CLUSOL    : forward followed by backward triangular solve (Precond.)
C C-----
C --C
C subroutine used.
C
C
C
C CQSPLIT   : quick split routine used by ilut to sort out the k
C C          elements in absolute value
C
C-----
C
C usage: call preconditioner then call pgmrez
C
C          This routine utilizes common blocks
C
C
C-----
C          subroutine cilut(lfil,droptol,flopil,ierr,jw, w)
C-----
C          include 'param.h'

```

```

common/BLK1/a(nzmax),ja(nzmax),ia(n+1)
common/BLK2/alu(iwk),jlu(iwk),ju(n)
implicit none
integer n, nzmax
real*8 droptol
complex*16 a,alu,w(n)
integer ia, ja,jlu,ju,jw(2*n),lfil
integer iw, flopil, ierr

c-----
c          *** CILUT preconditioner ***
c    incomplete LU factorization with dual truncation mechanism
c-----
c    Author: Yousef Saad *May, 5, 1990, Latest revision, August 1996
c-----
c  PARAMETERS
c-----
c
c  on entry:
c=====
c  n          = integer. The row dimension of the matrix A. The matrix
c
c  a,ja,ia = matrix stored in Compressed Sparse Row format.
c
c  lfil      = integer. The fill-in parameter. Each row of L and each row
c              of U will have a maximum of lfil elements (excluding the
c              diagonal element). lfil must be .ge. 0.
c              ** WARNING: THE MEANING OF LFIL HAS CHANGED WITH RESPECT TO
c              EARLIER VERSIONS.
c
c  droptol = real*8  Sets the threshold for dropping small terms in the
c              factorization. See below for details on dropping strategy.
c
c
c  iw        = integer. The lengths of arrays alu and jlu. If the arrays
c              are not big enough to store the ILU factorizations, ilut
c              will stop with an error message
c
c  On return.
c=====
c
c  alu,jlu = matrix stored in Modified Sparse Row (MSR) format
c  containing
c              the L and U factors together. The diagonal (stored in
c              alu(1:n) ) is inverted Each i-th row of the alu,jlu matrix
c              contains the i-th row of L (excluding the diagonal entry=1)
c              followed by the i-th row of U.
c
c  ju        = integer array of length n containing the pointers to
c              the beginning of each row of U in the matrix alu,jlu.
c
c  ierr      = integer. Error message with the following meaning.
c              ierr = 0    --> successful return.
c              ierr .gt. 0 --> zero pivot encountered at step number
c              ierr.
c              ierr = -1   --> Error. input matrix may be wrong

```

```

c                                     (The elimination process has generated a
c                                     row in L or U whose length is .gt  n )
c      ierr  = -2  --> The matrix L overflows the array al.
c      ierr  = -3  --> The matrix U overflows the array alu.
c      ierr  = -4  --> Illegal value for lfil.
c      ierr  = -5  --> zero row encountered
c
c work arrays.
c=====
c  jw      = integer work array of length 2*n.
c  w       = real work array of length n
c
c-----
c  w, ju (1:n) store the working array [1:11-1 = L-part, 11:n = u]
c  jw(n+1:2n) stores nonzero indicators
c
c Notes:
c -----
c The diagonal elements of the input matrix must be nonzero (at least
c 'structurally').
c
c-----
c---- Dual drop strategy works as follows.
c
c      1) Theresholding in L and U as set by droptol. Any element whose
c          magnitude is less than some tolerance (relative to the abs
c          value of diagonal element in u) is dropped.
c
c      2) Keeping only the largest lfil elements in the i-th row of L
c          and the largest lfil elements in the i-th row of U (excluding
c          diagonal elements).
c
c Flexibility: one can use droptol=0 to get a strategy based on
c keeping the largest elements in each row of L and U. Taking
c droptol .ne. 0 but lfil=n will give the usual threshold strategy
c (however, fill-in is then unpredictable).
c-----
c      locals
c      integer ju0,k,j1,j2,j,ii,i,lenl,lenu,jj,jrow,jpos,len,m
c      real*8 tnorm
c      complex*16 t, s, fact
c      if (lfil .lt 0) goto 998
c-----
c      initialize ju0 (points to next element to be added to alu,jlu)
c      and pointer array.
c-----
c      ju0 = n+2
c      jlu(1) = ju0
c      m = 1
c
c      initialize nonzero indicator array.
c
c      flop11 = 0
c      do 1 j=1,n
c          jw(n+j) = 0

```

```

1  continue

c-----
c  beginning of main loop.
c-----

do 500 ii = 1, n

    if (ii .eq. m*100) then
        write(1,*) 'flopilut=', flopil
        flopil = 0
        m= m+1
    endif
    j1 = ia(ii)
    j2 = ia(ii+1) - 1
    tnorm = 0.0d0
    do 501 k=j1,j2
        tnorm = tnorm+cdabs(a(k))
        flopil = flopil + 5
501  continue
    if (tnorm .eq. 0.0) goto 999
    tnorm = tnorm/real(j2-j1+1)

c
c  unpack L-part and U-part of row of A in arrays w
c
    lenu = 1
    lenl = 0
    jw(ii) = ii
    w(ii) = (0.0d0, 0.0d0)
    jw(n+ii) = ii

c
do 170 j = j1, j2
    k = ja(j)
    t = a(j)
    if (k .lt. ii) then
        lenl = lenl+1
        jw(lenl) = k
        w(lenl) = t
        jw(n+k) = lenl
    else if (k .eq. ii) then
        w(ii) = t
    else
        lenu = lenu+1
        jpos = ii+lenu-1
        jw(jpos) = k
        w(jpos) = t
        jw(n+k) = jpos
    endif
170  continue
    jj = 0
    len = 0

c
c  eliminate previous rows
c
150  jj = jj+1
    if (jj .gt. lenl) goto 160

```

```

c-----
c      in order to do the elimination in the correct order we must
select
c      the smallest column index among jw(k), k=jj+1, ..., lenl.
c-----
      jrow = jw(jj)
      k = jj
c
c      determine smallest column index
c
      do 151 j=jj+1,lenl
        if (jw(j) .lt jrow) then
          jrow = jw(j)
          k = j
        endif
151      continue
c
      if (k .ne jj) then
c      exchange in jw
        j = jw(jj)
        jw(jj) = jw(k)
        jw(k) = j
c      exchange in jr
        jw(n+jrow) = jj
        jw(n+j) = k
c      exchange in w
        s = w(jj)
        w(jj) = w(k)
        w(k) = s
      endif

c
c      zero out element in row by setting jw(n+jrow) to zero.
c
      jw(n+jrow) = 0
c
c      get the multiplier for row to be eliminated (jrow).
c
      fact = w(jj)*alu(jrow)
      if (cdabs(fact) .le droptol) goto 150
      flop11 = flop11 + 10
c
c      combine current row and row jrow
c
      do 203 k = ju(jrow), jlu(jrow+1)-1
        s = fact*alu(k)
        j = jlu(k)
        jpos = jw(n+j)
        if (j .ge ii) then
c
c      dealing with upper part.
c
          if (jpos .eq. 0) then
c
c      this is a fill-in element

```

```

c
      lenu = lenu+1
      if (lenu gt. n) goto 995
      i = ii+lenu-1
      jw(i) = j
      jw(n+j) = i
      w(i) = - s
    else

c
c   this is not a fill-in element
c
      w(jpos) = w(jpos) - s
      flopil = flopil + 4
      endif
    else

c
c   dealing with lower part.
c
      if (jpos .eq. 0) then

c
c   this is a fill-in element
c
      lenl = lenl+1
      if (lenl gt. n) goto 995
      jw(lenl) = j
      jw(n+j) = lenl
      w(lenl) = - s
    else

c
c   this is not a fill-in element
c
      w(jpos) = w(jpos) - s
      flopil = flopil + 4
      endif
    endif
    flopil = flopil + 6
203    continue

c
c   store this pivot element -- (from left to right -- no danger of
c   overlap with the working elements in L (pivots).
c
      len = len+1
      w(len) = fact
      jw(len) = jrow
      goto 150
160    continue

c
c   reset double-pointer to zero (U-part)
c
      do 308 k=1, lenu
        jw(n+jw(ii+k-1)) = 0
308    continue

c
c   update L-matrix

```

```

c      len1 = len
c      len = min0(len1,lfil)
c
c      sort by quick-split
c
c      call cqsplrit (w,jw,len1,len)
c
c      store L-part
c
c      do 204 k=1, len
c          if (ju0 .gt. iwk) goto 996
c          alu(ju0) = w(k)
c          jlu(ju0) = jw(k)
c          ju0 = ju0+1
204      continue
c
c      save pointer to beginning of row 11 of U
c
c      ju(11) = ju0
c
c      update U-matrix -- first apply dropping strategy
c
c      len = 0
c      do k=1, lenu-1
c          if (cdabs(w(ii+k)) .gt. droptol*tnorm) then
c              len = len+1
c              w(ii+len) = w(ii+k)
c              jw(ii+len) = jw(ii+k)
c          endif
c      enddo
c      lenu = len+1
c      len = min0(lenu,lfil)
c
c      call cqsplrit (w(ii+1), jw(ii+1), lenu-1,len)
c
c      copy
c
c      t = dcmlpx(cdabs(w(ii)), 0 0d0)
c      flop11 = flop11 + 4
c      if (len + ju0 .gt. iwk) goto 997
c      do 302 k=ii+1,ii+len-1
c          jlu(ju0) = jw(k)
c          alu(ju0) = w(k)
c          t = t + dcmlpx(cdabs(w(k)), 0.0d0)
c          ju0 = ju0+1
c          flop11 = flop11 + 6
302      continue
c
c      store inverse of diagonal element of u
c
c      if (w(ii) .eq. (0.0d0,0.0d0)) then
c
w(ii)=dcmlpx((0.0001+droptol)*tnorm,(0.0001+droptol)*tnorm)
c          flop11 = flop11 + 4

```

```

        end if
c
        alu(11) = (1.0d0,0.0d0)/ w(11)
c        flopi1 = flopi1 + 15
c        update pointer to beginning of next row of U.
c
        jlu(ii+1) = ju0
c-----
c        end main loop
c-----
500  continue
    ierr = 0
    return

c
c        incomprehensible error. Matrix must be wrong.
c
995  ierr = -1
    return

c
c        insufficient storage in L.
c
996  ierr = -2
    return

c
c        insufficient storage in U.
c
997  ierr = -3
    return

c
c        illegal lfil entered.
c
998  ierr = -4
    return

c
c        zero row encountered
c
999  ierr = -5
    return
c-----end-of-cilut-----
--
c-----
-
        end
c-----
c-----
-

subroutine clusol1(p)
include 'param.h'
common/BLK2/alu(1wk), jlu(1wk), ju(n)
common/BLKV/vv(n,1m+1)
common/BLKR/x(n)
complex*16 x,alu, vv
integer p,jlu, ju, floplus
c-----
c

```

```

c This routine solves the system (LU)  $x = y$ ,
c given an LU decomposition of a matrix stored in (alu, jlu, ju)
c modified sparse row format
c
c-----
c on entry.
c n   = dimension of system
c y   = the right-hand-side vector
c alu, jlu, ju
c     = the LU matrix as provided from the ILU routines.
c
c on return
c x   = solution of LU  $x = y$ .
c-----
c
c Note: routine is in place. call lusol (n, x, x, alu, jlu, ju)
c       will solve the system with rhs x and overwrite the result on x
c
c-----
c local variables
c
c       integer j,k, floplus
c
c forward solve
c
c       FLOPLUS = 0
c       do 40 j = 1, n
c         x(j) = vv(j,p)
c         do 41 k=jlu(j),ju(j)-1
c           x(j) = x(j) - alu(k)* x(jlu(k))
c           FLOPLUS = FLOPLUS + 8
c41        continue
c40      continue
c
c backward solve.
c
c       do 90 j = n, 1, -1
c         do 91 k=ju(j),jlu(j+1)-1
c           x(j) = x(j) - alu(k)*x(jlu(k))
c           FLOPLUS = FLOPLUS + 8
c91      continue
c         x(j) = alu(j)*x(j)
c         FLOPLUS = FLOPLUS + 6
c90      continue
c
c       write(1,*) 'floplus=', FLOPLUS
c       return
c-----end of clusol -----
c-----
c
c       end
c
c       subroutine clusol2
c       include 'param.h'
c       common/BLK2/alu(iwk),jlu(iwk),ju(n)

```

```

common/BLKR/x(n)
complex*16 alu, x
integer jlu, ju
integer j, k, floplus

FLOPLUS = 0
do 40 j=1,n

    do 41 k=jlu(j),ju(j)-1
        x(j) = x(j) - alu(k)*x(jlu(k))
        FLOPLUS = FLOPLUS + 8
41    continue
40    continue

c    backward solve

do 90 j=n,1,-1
    do 91 k=ju(j),jlu(j+1)-1
        x(j) = x(j) - alu(k)*x(jlu(k))
        FLOPLUS = FLOPLUS + 8
91    continue
        x(j) = alu(j)*x(j)
        FLOPLUS = FLOPLUS + 6
90    continue

write(1,*) 'floplus=', floplus
return
end

c-----
c-----
c    subroutine cqspllt(a,ind,n,ncut)
c    complex*16 a(n)
c    integer ind(n), n, ncut
c-----
c    does a quick-sort split of a complex array.
c    on input a(1:n). is a complex array
c    on output a(1:n) is permuted such that its elements satisfy:
c
c    cdabs(a(1)) .ge. abs(a(ncut)) for 1 .lt. ncut and
c    cdabs(a(1)) .le. abs(a(ncut)) for 1 .gt. ncut
c
c    ind(1:n) is an integer array which permuted in the same way as
c    a(*).
c-----
c    complex*16 tmp
c    real*8 abskey
c    integer itmp, first, last
c
c    first = 1
c    last = n
c    if (ncut .lt. first .or. ncut .gt. last) return
c
c    outer loop -- while mid .ne. ncut do
c

```

```

1      mid = first
      abskey = cdabs(a(mid))
      do 2 j=first+1, last
          if (cdabs(a(j)) .gt. abskey) then
              mid = mid+1
c      interchange
          tmp = a(mid)
          itmp = ind(mid)
          a(mid) = a(j)
          ind(mid) = ind(j)
          a(j) = tmp
          ind(j) = itmp
          endif
2      continue
c
c      interchange
c
      tmp = a(mid)
      a(mid) = a(first)
      a(first) = tmp
c
      itmp = ind(mid)
      ind(mid) = ind(first)
      ind(first) = itmp
c
c      test for while loop
c
      if (mid .eq. ncut) return
      if (mid .gt. ncut) then
          last = mid-1
      else
          first = mid+1
      endif
      goto 1
c-----end-of-cqspl1t-----
c-----
      end
      subroutine camux1(p)
      include 'param.h'
      common/BLK1/a(nzmax),ja(nzmax),ia(n+1)
      common/BLKV/vv(n,im+1)
      common/BLKB/x(n)
      implicit none
      complex*16 x,a, vv, t
      integer ja, ia, j,p

      integer j,k,flopam

      flopam = 0
      do 100 j=1,n
          t = (0.0d0,0.0d0)
          do 99 k=ia(j), ia(j+1)-1
              t = t + a(k)*x(ja(k))
              flopam = flopam + 8
99          continue

```

```

      vv(j,p) = t
100  continue

      return
      end

      subroutine camux2(p)
      include 'param.h'
      common/BLK1/a(nzmax),ja(nzmax),ia(n+1)
      common/BLKV/vv(n,1m+1)
      common/BLKR/x(n)
      implicit none
      complex*16 x,a,vv,t
      integer ja,ia,j,p
      integer j,k,flopam,n

      flopam = 0
      do 100 j=1,n
         t=(0.0d0,0.0d0)
         do 99 k=ia(j),ia(j+1)-1
            t = t + a(k)*x(ja(k))
            flopam = flopam + 8
99      continue
         vv(j,p) = t
100    continue
      return
      end

```

VITA

Julie Renéé Spangler (Oates) was born on June 29, 1976 in Reno, Nevada. She moved to Pueblo, Colorado in 1981, where she attended school and graduated from Centennial High School in 1994. In the fall of 1994, she started undergraduate studies at the University of Southern Colorado. She graduated Suma cum Laude with a Bachelor of Science Degree in physics and mathematics in May 1998. That fall, she moved to Tennessee and began her graduate studies in mathematics at the University of Tennessee Space Institute. When not studying, she likes to read (mostly mysteries) and travel (when she has the time and can afford it).

After completing her Master of Science degree in May 2000, Julie plans to move to Charleston, South Carolina where she will work as a nuclear engineer