To the Graduate Council:
I am submitting herewith a dissertation written by Robert Wallace Tramel entitled "A Study of Domain Decomposition Methods Applied to the Discretized Helmholtz Equation". I have examined the final paper copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Engineering Science.

John E. Caruthers, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Vice Provost and Dean of Graduate Studies

A Study of Domain Decomposition Methods Applied to the Discretized Helmholtz Equation

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Robert Wallace Tramel
DECEMBER 2001

Considering the Snail

The snail pushes through a green

night, for the grass is heavy

with water and meets over

the bright path he makes, where rain

has darkened the earth's dark. He

moves in a wood of desire,

pale antlers barely stirring

as he hunts. I cannot tell

what power is at work, drenched there

with purpose, knowing nothing.

What is a snail's fury? All

I think is that if later

I parted the blades above

the tunnel and saw the thin

trail of broken white across

litter, I would never have

imagined the slow passion

to that deliberate progress.

Thom Gunn

Dedication

This dissertation is dedicated
to my wife
Terri Lynn Tramel
without whose love and support this
this work would have been impossible

and to my new son Mark Robert
for inspiration

Acknowledgments

The author would like to thank all the people who have shared so freely of their intellectual capital without which this work would have remained unfinished. Thanks are due to all the authors of the software used in this work:

1. R. W. Freund and N. M. Nacthigal (**QMRPACK**)
2. J. W. Demmel, J. R. Gilbert and X. S. Li (**SuperLU**)
3. All the members of the **lapack** team
4. The free software foundation (gcc, g77)
5. Sierra Pacific Corporation (vf90).

The author would like to acknowledge and thank Prof. Henry Rice of Trinity College, Dublin for suggesting that a restriction of the wave vectors used in the GFD expansion at a boundary was worth investigating. Thanks are due to Dr. K. C. Reddy for all the help and support he has given over many long years and for first introducing me to Krylov methods. Finally special thanks go out to Dr. John Caruthers for introducing me to the fascinating field of Green's Function Discretization and for encouraging me to work on this topic for my dissertation.

Abstract

In this work a domain decomposition based preconditioner of the additive Schwarz type is developed and tested on the linear systems which arise out of the application of the Green's Function/Wave Expansion Discretization (GFD/WED) method to Helmholtz's equation. In order to develop the additive Schwarz preconditioner, use is made of a class of one–sided Artificial Radiation Boundary Conditions (ARBC) developed during the course of this work. These ARBCs are computationally shown to be quite accurate for use on their own. The ARBC's are used to radiatively couple the various sub-domains which are naturally part of domain decomposition based methods in such a manner as to ensure that the system matrix, when restricted to the sub-domains, is non-singular. In addition, the inter-domain ARBC is constructed such that the solution to the global linear system is unaffected by the presence of the artificial boundaries. The efficacy and efficiency of the method is demonstrated on one, two, and three-dimensional test cases.

Contents

1 INTRODUCTION 1

2 THEORY 7

2.1 Introduction 7

2.2 Green's Function Discretization 9

2.3 GFD for the Helmholtz Equation 13

2.3.1 Two-dimensional Helmholtz Equation 14

2.3.2 Three-dimensional Helmholtz Equation 18

2.4 Radiation Boundary Conditions 19

2.5 Domain Decomposition Methods 21

2.6 Krylov Subspace Methods 27

2.6.1 Preconditioned Krylov Subspace Methods 31

2.7 QMR 32

3 One-Dimensional Test Cases 38

3.1 One-Dimensional GFD 38

3.2 Domain Decomposition Based Considerations 41

3.3 Numerical Examples 44

3.4 Spectral Analysis 45

4 Two-Dimensional Test Cases 52

4.1 Problem Description 52

4.2 Numerical Results 55

5 Three-Dimensional Test Cases **62**

5.1 Problem Description 62

5.2 Computational Cases 63

5.3 Numerical Results 64

5.4 Numerical Condition Numbers 78

6 Conclusions **80**

Bibliography **83**

Appendix **90**

A Source Codes **91**

A.1 QMR Driver Code 91

A.2 C++ Interface to SuperLU 104

A.3 C Sparse Matrix Factorization Routines 109

Vita **114**

List of Tables

4.1 Summary of various two-dimensional test cases. 54

4.2 Summary of cpu usage of the various two-dimensional test cases. 58

5.1 Summary of three-dimensional parametric test cases. 65

5.2 Summary of large scale cases. 66

5.3 Summary of three-dimensional hybrid test cases. 70

5.4 Summary of condition numbers. 79

List of Figures

1.1 Typical structure of inverse matrix for Laplace's equation. 4

1.2 Typical structure of inverse matrix for Helmholtz's equation, (20.0 points per wavelength). 4

1.3 Typical structure of inverse matrix for Helmholtz's equation, (3.33 points per wavelength). 5

2.1 Natural partitioning of a Matrix induced by domain decomposition. . . 24

3.1 One-dimensional overlap region ($overlap = 2$) 43

3.2 Convergence rate using symmetric block Gauss-Seidel scheme with radiation boundary interface condition ($\lambda = 4.0\delta$). 46

3.3 Convergence rate using symmetric block Gauss-Seidel scheme with radiation boundary interface condition ($\lambda = 4.03\delta$). 46

3.4 Convergence rate using QMR Krylov solver with additive Schwarz preconditioner ($\lambda = 4.0\delta$). 47

3.5 Convergence rate using QMR Krylov solver with additive Schwarz preconditioner ($\lambda = 4.03\delta$). 47

3.6 Structure plot of the preconditioned matrix. 49

3.7 Eigenspectrum of the preconditioned matrix using equation 3.18. 50

3.8 Spectral radii of the preconditioned matrices. 50

4.1 Problem setup for the two-dimensional test cases. 53

4.2 Convergence rates for various test cases. 56

4.3 Comparison of the convergence rates for test cases 9 and 10 vs. ppw. . . 56

4.4 Comparison of computed and exact solution along the line $y = 1$, 3ppw. 58

4.5 Comparison of computed and exact solution along the line $y = 1$, 15ppw. 59

4.6 Comparison of the convergence rates vs. the number of zones, 3ppw. . . 60

4.7 Comparison of the convergence rates vs. the number of zones, 15ppw. . 60

5.1 Comparison of the convergence rates vs. the number of zones ($yrange = -1.0$, $3ppw$). 66

5.2 Comparison of the convergence rates vs. the number of zones, ($yrange = -1.0$, $9ppw$). 67

5.3 Comparison of the convergence rates vs. the number of zones, ($yrange = -1.0$, $15ppw$). 67

5.4 Comparison of the convergence rates vs. the number of zones ($yrange = -0.6$, $3ppw$). 68

5.5 Comparison of the convergence rates vs. the number of zones, ($yrange = -0.6$,$9ppw$). 68

5.6 Comparison of the convergence rates vs. the number of zones, ($yrange = -0.6$, $15ppw$). 69

5.7 Comparison of computed and exact solutions ($3ppw$). 71

5.8 Comparison of computed and exact solutions ($9ppw$). 71

5.9 Comparison of computed and exact solutions ($15ppw$). 72

5.10 Comparison of hybrid yrange convergence rates (3ppw, $ND = 8$, $np = 17$). 73

5.11 Comparison of hybrid yrange convergence rates (3ppw, $ND = 1000$, $np = 5$). 73

5.12 Comparison of hybrid yrange convergence rates (9ppw, $ND = 8$, $np = 17$). 74

5.13 Comparison of hybrid yrange convergence rates (9ppw, $ND = 1000$, $np = 5$). 74

5.14 Comparison of hybrid yrange convergence rates (15ppw, $ND = 8$, $np = 17$). 75

5.15 Comparison of hybrid yrange convergence rates (15ppw, $ND = 1000$, $np = 5$). 75

5.16 Convergence rates for large scale test cases. 76

5.17 Comparison of solutions for large scale test cases. 77

Chapter 1

INTRODUCTION

The Helmholtz equation is the fundamental model for linear wave propagation in the frequency domain. As such it has important applications in areas such as aeroacoustics and electromagnetics. However the desire to obtain accurate solutions to the Helmholtz equation places severe demands on the numerical analyst. Accuracy considerations require that a sufficient number of grid points per wavelength be used in order to reduce the dispersion and dissipation errors associated with the discretized solution to a minimum. For standard finite difference techniques, 10-20 points per wavelength are required to reduce the truncation error to acceptable levels. However several new approaches to reducing the number of grid points required by using functions derived from plane waves have been advanced [5], [3], [24], [7], [8]. Of these methods the Green's Function Discretization (GFD) approach of Caruthers et al. [7], [8] has been particularly successful in that the method is able to reduce the number of grid points

required per wavelength to 2-3. This clearly goes a long way in helping to achieve the goal of obtaining accurate solutions in an efficient manner. However, even the strides provided by GFD are alone not enough to allow the treatment of practical engineering problems in aeroacoustics. This is because GFD, like all difference techniques for partial differential equations which contain an elliptical component, requires the solution of a large linear system. In aeroacoustic applications these systems can easily contain over 1 million unknowns. Thus approaches other than Gaussian elimination (or variants thereof) are needed. For example, consider a three-dimensional domain discretized using a structured grid with 100 points in each direction. If the unknowns are grouped by planes then the bandwidth of the system w is 10000. The operation count to perform a LU decomposition of a banded linear system with n unknowns requires on the order of $w^2 n$ floating point operations [35]. For the case described above ($n = 10^6$) this amounts to something on the order of 10^{14} floating point operations. On a machine capable of performing 10 Million Floating Point Operations per Second at a sustainable rate, approximately 120 days would be required to obtain a solution. However, given that direct inversion (or factorization) is impractical for use with this class of problems, it is also true that the use of iterative solution techniques is by no means straightforward. This is because the linear systems which arise when applying the GFD technique can be non-symmetric, non-positive definite, and ill-conditioned. In addition the inverses of these matrices are quite complicated in structure and the elements of the inverses do not display the same decay in magnitude away from the main diagonal as would be true

of the inverse matrices which arise from the discretization of typical elliptical systems. For example typical columns of the inverse matrices which arise when solving Laplace's equation as well as the Helmholtz equation using both 20.0 Points Per Wavelength (PPW) as well as 3.33 PPW are shown in the Figures 1.1 – 1.3.

In all of these cases the computational domain is a unit sided square discretized using 40 grid points in each direction. Dirichlet boundary conditions are applied on two sides of the square, while Neumann boundary conditions are applied on the other two sides. As can be seen in Figure 1.1, where the 800th column of the inverse matrix from a discretization of Laplace's is displayed, the elements of the inverse matrix in this example are well behaved and display a strong decay away from the diagonal. On the other hand, with Helmholtz's equation using 20 PPW the inverse matrix elements are now oscillatory and do not decay away from the diagonal as can be seen in Figure 1.2.

By the time one reaches 3.33 PPW, as can be in Figure 1.3 the inverse matrix elements become highly oscillatory and vary wildly from point to point.

These facts generally preclude the use of standard preconditioning approaches such as Symmetric Successive Over Relaxation (SSOR) [13] or Incomplete Lower Upper factorization with Thresholding (ILUT) [28] for accelerating the convergence rates of iterative solvers in spite of the fact that these methods have been shown to be very beneficial when applied to the solution of purely elliptical problems. However progress in this area in the 12-20 grid point per wave length regime has recently been demonstrated [34].

In this dissertation, a systematic study is made of the applicability and efficacy of

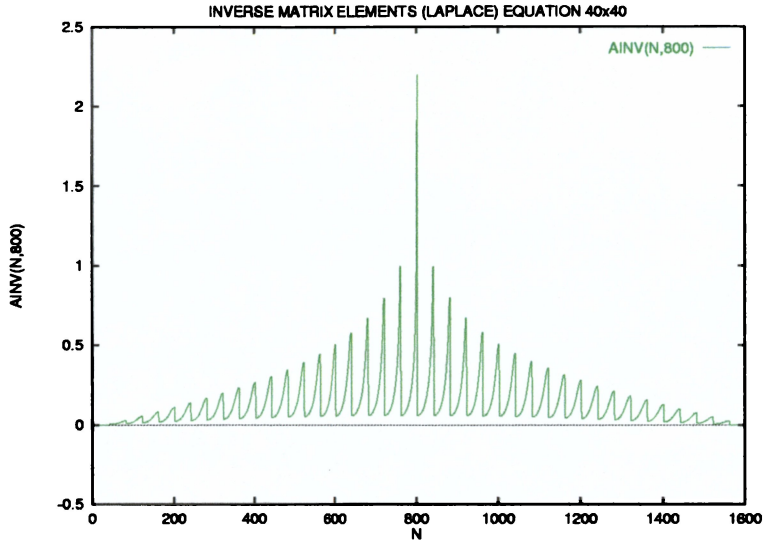


Figure 1.1: Typical structure of inverse matrix for Laplace's equation.

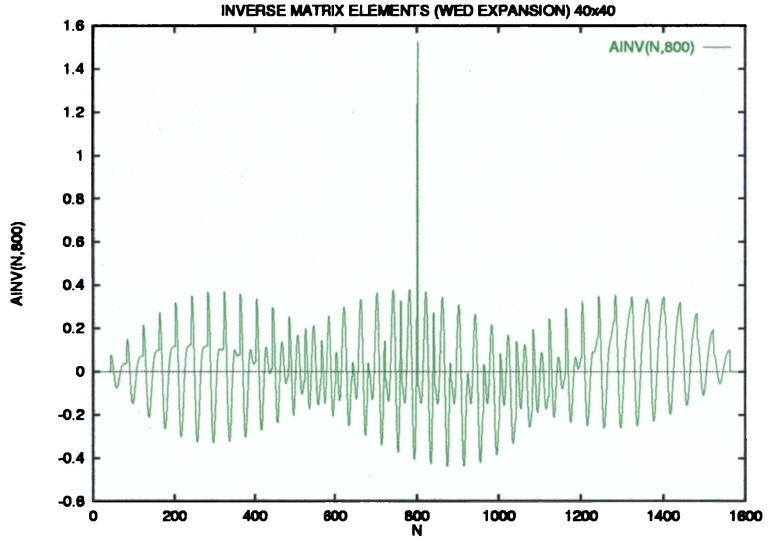


Figure 1.2: Typical structure of inverse matrix for Helmholtz's equation, (20.0 points per wavelength).

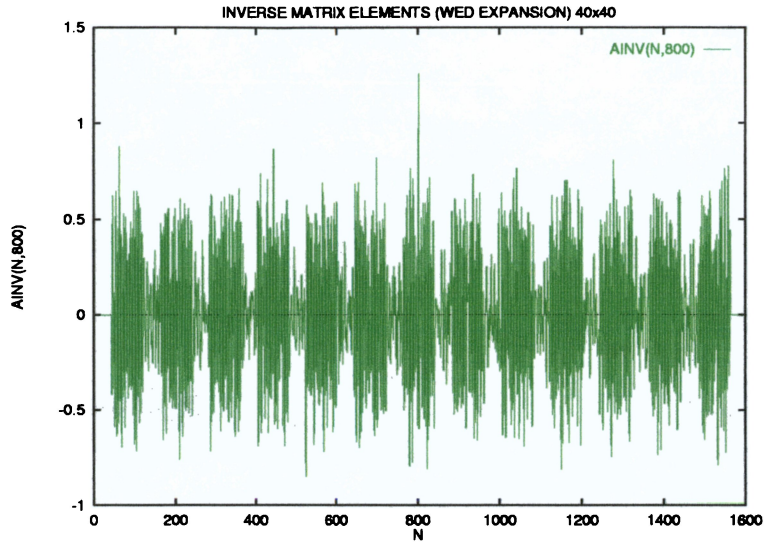


Figure 1.3: Typical structure of inverse matrix for Helmholtz's equation, (3.33 points per wavelength).

domain decomposition based preconditioners [33] to efficiently solve the linear systems which arise from application of the GFD technique. Other attempts to apply domain decomposition based approaches to the solution to Helmholtz's equation are presented in [11], [6] , [25], [36], and [2]. The outline of this dissertation is as follows.

First the relevant theoretical background of the methods used in this research are reviewed in Chapter 2. In Chapter 3 the GFD method is used to discretize a one-dimensional test problem. A domain decomposition based preconditioner is developed and is used to solve the resulting linear system using both a stationary block iterative scheme (Gauss-Seidel) [33], [29] as well as with a Krylov [29], [19] subspace method. In this and all future examples, the Krylov subspace methods used are taken from the **QMRPACK** [18] software package developed by Freund and Nachtigal. All the Krylov

methods coded for use in **QMRPACK** are variants of the Quasi Minimum Residual (QMR) method developed by Freund and Nachtigal [17]. QMR is selected over the more popular GMRES [31] algorithm since QMR requires less memory than GMRES while displaying similar convergence characteristics [4]. In addition, **QMRPACK** was selected since when this research was initiated, **QMRPACK** was, to the authors knowledge, the only freely available Krylov method based software package which was capable of treating complex valued linear systems. An Artificial Radiation Boundary condition (ARBC) is used to stabilize the block Gauss-Seidel iterative procedure. The convergence rates for both iterative methods are examined and a strong wave-number dependence is displayed. An eigenanalysis of the preconditioning matrix both with and without the use of the ARBC is used to explain and interpret the numerical results. In Chapter 4 numerical results are presented in which known point source (i.e., monopole) radiators are used to to specify a class of test problems. The computational domain in this case is a unit-sided square. Dirichlet boundary conditions are imposed on the two sides of the square closet to the point sources while ARBCs are used on the other two sides. The relevant driver subroutine from **QMRPACK** is modified to make use of the ARBC based domain decomposition preconditioner developed for this dissertation. This modified version of **QMRPACK** is then used to perform a variety of computational tests on this problem. The same computational experiments are repeated for three dimensions using a unit sided cube in Chapter 5. Conclusions and recommendations are presented in Chapter 6.

Chapter 2

THEORY

2.1 Introduction

Closed form, analytical solutions to linear Partial Differential Equations (PDEs) (such as the Helmholtz equation) are only obtainable in a small class of coordinate systems and boundary conditions. These coordinate systems are known as separable coordinate systems [26]. The Helmholtz equation is known to be separable in 11 different coordinate systems. However, if one wishes to solve the Helmholtz equation in more general situations, then one is forced to resort to the use of approximate solution methods. One of the classical approaches to the approximation problem is the finite-difference method [21]. In the finite difference method one first discretizes the computational domain into a finite subset of unique grid points and an approximation to the exact solution at each grid point. Next a set of difference equations is derived by first assuming that the exact solution is known at each of the individual grid points. These discrete point values are

then used to develop approximations to the differential operators which appear in the PDE. At every discrete point, these approximations are used to replace the differential operators in the PDES. This procedure produces a set of linear (difference) equations which is then solved by either direct or iterative methods. Another approach to the development of finite difference methods assumes that the approximate value at the "center" of a group of neighboring points is given as a linear combination of the approximate values at these neighbor points with a weight being assigned to each neighbor point. Then, in the resulting linear equation, the values of the neighboring points are expressed in terms of the center point by means of Taylor series expansions. The values of the weights are chosen so as to eliminate as many of the lower order terms as is possible in the Taylor series expansions. This method also produces a set of difference equations which must be solved by appropriate means.

In this dissertation the Helmholtz equation is solved by a novel approximation method known as the Green's Function Discretization (GFD). The Green's Function Discretization shares many similarities with traditional finite difference methods such as expressing the value of the approximate solution at the center of a stencil of points as a linear combination of the approximate solution values at the neighboring points in the stencil. Given the fact that the derivation of the weights used in the GFD is radically different from the traditional finite difference methods outlined above, this chapter will begin with an in depth review of the details of the Green's function discretization method. However, the development of the discretization coefficients by the

GFD method is only the first step needed to obtain the desired approximate solution to the Helmholtz equation. The resulting linear system of equations must still be solved by either direct factorization or by the use of some variety of iterative solution technique. The main goal of this dissertation is a systematic study of the applicability and the efficacy of domain decomposition based solution strategies for solving the linear system of equations which arise from application of the GFD. Thus, the GFD review is followed by a review of domain decomposition based approaches to the solution of linear systems. In this dissertation, domain decomposition based methods will mainly be used to develop preconditioners for use in conjunction with the QMR family of Krylov subspace methods developed by Freund and Nachtigal [17]. Therefore, this chapter closes with a brief review of Krylov subspace methods in general and QMR based Krylov subspace methods in particular.

2.2 Green's Function Discretization

A novel discretization method for the treatment of the Helmholtz equation was presented in [7] , [8] which reduced the number of grid points per wavelength required to accurately model wave phenomena from on the order of 20 for traditional 2nd order accurate finite difference methods to on the order of 2-3. The original derivation of the Green's Function Discretization method was based as much on intuition and the belief that numerical methods based on a physical understanding of wave phenomena would outperform any truncation error based scheme than on purely mathematical analysis. Indeed the authors

of [8] cite the following passage from [27] as inspiration for the development of the GFD: "...there is some historical precedent to believe that any scheme that incorporates some of the basic physical understanding of how waves propagate should have some intrinsic advantage over one based on brute force". To this end Caruthers et al. [8] developed the GFD based on the following interpolation scheme

$$\phi(\mathbf{r}) = \sum_{j=1}^{N} \gamma_j \psi_j(\mathbf{r}) \tag{2.1}$$

whereby a candidate solution $\phi(\mathbf{r})$ to a linear equation $\mathcal{L}\{\psi\} = 0$ is expressed as a linear combination of a set of N functions $\psi_j(\mathbf{r})$ which are themselves all solutions of the same linear equation. In equation 2.1 and for the remainder of this section the mathematical development and nomenclature used in [9] is closely followed. Equation 2.1 is assumed to hold in a neighborhood of a point $\mathbf{r}_0$ which contains a cluster of M discrete points $\mathbf{r}_i$; $i = 1, 2, ..., M$. This collection of points forms the computational "stencil" or "pencil" for $\mathbf{r}_0$. Next, the values of the weights γ_j are determined by forcing the series expansion to match the candidate solution at all of the M stencil points, i. e.,

$$\phi(\mathbf{r}_i) = \sum_{j=1}^{N} \gamma_j \psi_j(\mathbf{r}_i) \;\; ; \;\; i = 1, 2, ..., M \tag{2.2}$$

or in matrix notation,

$$\boldsymbol{\phi}^T = \boldsymbol{\gamma}^T \psi. \tag{2.3}$$

However, the solution for $\boldsymbol{\gamma}^T$ in terms of ϕ and ψ is unique only if $N = M$. In most

practical cases $N > M$ so that the solution is not unique (in fact an infinite number of solutions exist with $N - M$ free parameters). Out of this infinite set of possible solutions only one has an additional mathematical property to distinguish it from all the others. This property is the minimum norm property [35], i. e., the particular solution that minimizes the L_2 norm of the weights, $\|\boldsymbol{\gamma}\|_2$. This is the solution for the weights γ_j chosen in [8]. In order to determine the the minimum norm solution, first define the auxiliary variable $\boldsymbol{\alpha}$ as follows

$$\boldsymbol{\gamma}^T = \boldsymbol{\alpha}^T \boldsymbol{\psi}^H \tag{2.4}$$

where $\boldsymbol{\psi}^H$ is the complex conjugate transpose of $\boldsymbol{\psi}$. In terms of $\boldsymbol{\alpha}$ Equation 2.3 is now written as

$$\boldsymbol{\phi}^T = \boldsymbol{\alpha}^T \boldsymbol{\psi}^{\mathbf{H}} \boldsymbol{\psi}. \tag{2.5}$$

The matrix $\left[\boldsymbol{\psi}^H \boldsymbol{\psi}\right]$ is of full rank and is invertible, thus the solution for $\boldsymbol{\alpha}$ is given by

$$\boldsymbol{\alpha}^T = \boldsymbol{\phi}^T \left[\boldsymbol{\psi}^H \boldsymbol{\psi}\right]^{-1} \tag{2.6}$$

from which it follows that the solution for the weights $\boldsymbol{\gamma}$ is given by

$$\boldsymbol{\gamma}^T = \boldsymbol{\phi}^T \left[\boldsymbol{\psi}^H \boldsymbol{\psi}\right]^{-1} \boldsymbol{\psi}^H \tag{2.7}$$

Evaluating equation 2.3 at the center point of the stencil $\mathbf{r}_0$ yields

$$\phi(\mathbf{r}_0) = \sum_{j=1}^{N} \gamma_j \psi_j(\mathbf{r}_0) \tag{2.8}$$

or

$$\phi(\mathbf{r}_0) = \boldsymbol{\gamma}^T \boldsymbol{\psi}_0 = \boldsymbol{\phi}^T \left[\boldsymbol{\psi}^H \boldsymbol{\psi}\right]^{-1} \boldsymbol{\psi}^H \boldsymbol{\psi}_0 \tag{2.9}$$

which is the equation for the GFD first derived in [8]. The similarity between the GFD and standard finite differences can be seen by rearranging the result as follows

$$\phi(\mathbf{r}_0) = \boldsymbol{\phi}^T \mathbf{a} \tag{2.10}$$

where

$$\mathbf{a} = \left[\boldsymbol{\psi}^H \boldsymbol{\psi}\right]^{-1} \boldsymbol{\psi}^H \boldsymbol{\psi}_0. \tag{2.11}$$

This procedure is repeated at every discrete point in the computational domain. If there are N_p points in the domain then the discrete system for the entire domain is simply written as

$$\phi(\mathbf{r}_0^p) = \left(\boldsymbol{\phi}^p\right)^T \mathbf{a}^p \quad ; \; p = 1, 2, 3, ..., N_p \tag{2.12}$$

If the discrete points are distributed uniformly and the governing equation $\mathcal{L}\{\psi\} = 0$

is not a function of spatial position, then the $\mathbf{a}^p$ do not vary from point to point so that

$$\phi\left(\mathbf{r}_0^p\right) = \left(\boldsymbol{\phi}^p\right)^T \mathbf{a} \;\; ; \; p = 1, 2, 3, ..., N_p \tag{2.13}$$

At this point a fascinating and mathematically interesting property of the GFD is worth mentioning. As outlined above, the weights $\mathbf{a}$ were chosen to satisfy the minimum norm property. However these weights also imply a minimum error property as well. In [9] it was shown that of all possible choices of weights $\mathbf{a}$, for the weights defined in equation 2.11, the norm of the interpolation error over all the basis functions ψ_j is a minimum. The weights are thus also unique in that they interpolate the basis functions ψ_j as accurately as possible.

2.3 GFD for the Helmholtz Equation

In the case of the Helmholtz equation, $\mathcal{L}\{\psi\} = 0$ is given by

$$\nabla^2\psi + k^2\psi = 0 \tag{2.14}$$

and the basis functions ψ_j are taken to be propagating plane wave solutions, i.e.,

$$\psi_j(\mathbf{r}_i) = \frac{e^{-ik\hat{\mathbf{n}}_j\cdot\mathbf{r}_i}}{\sqrt{N}}; \;\; j = 1, 2, 3, ..., N \tag{2.15}$$

where the basis functions have all been normalized by the constant factor $\sqrt{N}$ and $\|\hat{\mathbf{n}}_j\| = 1$ for all j. Thus the elements, $\left[\psi^H\psi\right]_{m,i}$ of the $M \times M$ matrix $\psi^H\psi$ are given by

$$\left[\psi^H\psi\right]_{m,i} = \frac{1}{N}\sum_{j=1}^{N} e^{ik\hat{\mathbf{n}}_j\cdot(\mathbf{r}_m-\mathbf{r}_i)} \tag{2.16}$$

while the entries $\left[\psi^H\psi_0\right]_i$ of the $M \times 1$ vector $\psi^H\psi_0$ are given by

$$\left[\psi^H\psi_0\right]_i = \frac{1}{N}\sum_{j=1}^{N} e^{ik\hat{\mathbf{n}}_j\cdot(\mathbf{r}_i-\mathbf{r}_0)} \tag{2.17}$$

In addition, this specific version of the general GFD procedure is sometimes referred to as the Wave Expansion Discretization (WED) method.

2.3.1 Two-dimensional Helmholtz Equation

In two dimensions the $\hat{\mathbf{n}}_j$ vectors are assumed to be uniformly distributed in the $x-y$ plane as follows,

$$\begin{aligned} \hat{\mathbf{n}}_j &= \hat{\mathbf{i}}\cos(\theta_j) + \hat{\mathbf{j}}\sin(\theta_j) \quad ; \\ \theta_j &= (j-1)\tfrac{2\pi}{N}; \quad j = 1,2,3,...,N. \end{aligned} \tag{2.18}$$

In the case in which the vectors $\hat{\mathbf{n}}_j$ are continuously distributed, the elements of the matrix $\psi^H\psi$ are given by

$$\lim_{N\to\infty}\left[\psi^H\psi\right]_{m,i} = \lim_{N\to\infty}\frac{1}{N}\sum_{j=1}^{N} e^{ik\hat{\mathbf{n}}_j\cdot(\mathbf{r}_m-\mathbf{r}_i)} \tag{2.19}$$

In order to simplify the computations it can be assumed Without Loss Of Generality (WLOG) that a local coordinate system can always be introduced such that $\mathbf{r}_m - \mathbf{r}_i = |\mathbf{r}_m - \mathbf{r}_i|\,\hat{i}$ in which case

$$\begin{aligned}\lim_{N\to\infty}\left[\psi^H\psi\right]_{m,i} &= \lim_{N\to\infty}\frac{1}{2\pi}\sum_{j=1}^{N} e^{ik|\mathbf{r}_m-\mathbf{r}_i|\cos\theta_j}\,\Delta\theta \\ &= \frac{1}{2\pi}\int_0^{2\pi} e^{ik|\mathbf{r}_m-\mathbf{r}_i|\cos\theta}\,d\theta \\ &= J_0\left(k\,|\mathbf{r}_m-\mathbf{r}_i|\right) \qquad (2.20)\end{aligned}$$

A similar computation shows that the components $\left[\psi^H\psi_0\right]_i$ of the vector $\psi^H\psi_0$ are given (in the continuous limit) by

$$\lim_{N\to\infty}\frac{1}{N}\left[\psi^H\psi_0\right]_i = J_0\left(k\,|\mathbf{r}_i-\mathbf{r}_0|\right) \qquad (2.21)$$

Equations 2.20 – 2.21 represent the most general result available and as such are applicable to arbitrary stencils. However analytical solutions for the weights $\mathbf{a}$ are derived in [9] for the special case of a square, uniformly distributed 9 point stencil. Choosing the $\mathbf{r}_i$ with a normalized spacing of 1 allowed the authors of [9] to derive the following exact result for $\mathbf{a}$:

$$\frac{\mathbf{r}_i}{\delta} = \left\{ \begin{array}{ll} -\hat{\mathbf{i}} - \hat{\mathbf{j}} & ; i = 1 \\ -\hat{\mathbf{j}} & ; i = 2 \\ \hat{\mathbf{i}} - \hat{\mathbf{j}} & ; i = 3 \\ \hat{\mathbf{i}} & ; i = 4 \\ \hat{\mathbf{i}} + \hat{\mathbf{j}} & ; i = 5 \\ \hat{\mathbf{j}} & ; i = 6 \\ -\hat{\mathbf{i}} + \hat{\mathbf{j}} & ; i = 7 \\ -\hat{\mathbf{i}} & ; i = 8 \end{array} \right. \quad \Longrightarrow \quad \mathbf{a} = \left\{ \begin{array}{l} a_1 \\ a_2 \\ a_1 \\ a_2 \\ a_1 \\ a_2 \\ a_1 \\ a_2 \end{array} \right\} \tag{2.22}$$

where the solution to equation 2.11 was found to be

$$a_1 = -\frac{\begin{array}{c} -2J_0\left(k\delta\sqrt{2}\right)^2 - J_0\left(2k\delta\right) J_0\left(k\delta\sqrt{2}\right) - J_0\left(k\delta\sqrt{2}\right) \\ +2J_0\left(k\delta\right)^2 + 2J_0\left(k\delta\sqrt{5}\right) J_0\left(k\delta\right) \end{array}}{D} \tag{2.23}$$

$$a_2 = \frac{\begin{array}{c} -2J_0\left(k\delta\sqrt{2}\right) J_0\left(k\delta\sqrt{5}\right) - 2J_0\left(k\delta\sqrt{2}\right) J_0\left(k\delta\right) \\ +J_0\left(2k\delta\sqrt{2}\right) J_0\left(k\delta\right) + J_0\left(k\delta\right) + 2J_0\left(2k\delta\right) J_0\left(k\delta\right) \end{array}}{D} \tag{2.24}$$

and

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

Insight into the nature of the weights derived in equations 2.23 – 2.24 can be obtained by expanding the weights in a terms of a Taylor series expansion in k. The result is [9]

$$a_1 = \frac{1}{20} + \frac{17}{1000}\left(k\delta\right)^2 + \frac{801}{200\,000}\left(k\delta\right)^4 + \frac{76\,313}{90\,000\,000}\left(k\delta\right)^6 \tag{2.25}$$
$$+ \frac{1091\,144\,231}{6272\,000\,000\,000}\left(k\delta\right)^8 + O\left(\left(k\delta\right)^{10}\right)$$
$$a_2 = \frac{1}{5} + \frac{29}{500}\left(k\delta\right)^2 + \frac{2549}{200\,000}\left(k\delta\right)^4 + \frac{473\,849}{180\,000\,000}\left(k\delta\right)^6 \tag{2.26}$$
$$+ \frac{10\,086\,892\,607}{18\,816\,000\,000\,000}\left(k\delta\right)^8 + O\left(\left(k\delta\right)^{10}\right).$$

Taking the limit $k \to 0$ yields the classical 6^{th} order finite difference representation for Laplace's equation [21]

$$a_1 = \frac{1}{20} \;\; ; \;\; a_2 = \frac{1}{5}. \tag{2.27}$$

In [9] with the aid of a symbolic algebra program, 6^{th} order accuracy was proven for arbitrary k.

2.3.2 Three-dimensional Helmholtz Equation

In three-dimensions the $\hat{\mathbf{n}}_{i,j}$ vectors are assumed to be uniformly distributed in space as follows,

$$\hat{\mathbf{n}}_{i,j} = \hat{\mathbf{i}}\cos(\phi_i)\sin(\theta_j) + \hat{\mathbf{j}}\sin(\phi_i)\sin(\theta_j) + \hat{\mathbf{k}}\cos(\theta_j) \tag{2.28}$$

$$\begin{aligned} \phi_i &= (i-1)\frac{2\pi}{M} \quad ; \quad i = 1, 2, 3, ..., M \\ \theta_j &= \cos^{-1}(y_j) \quad ; \quad j = 1, 2, 3, ..., N, \end{aligned} \tag{2.29}$$

where

$$y_j = -1 + (j-1)\left(\frac{2}{N}\right) . \tag{2.30}$$

The inverse cosine distribution for the θ_j's is used to avoid the tendency toward the clustering of the $\hat{\mathbf{n}}_{i,j}$'s near the $\theta = 0$ and $\theta = \pi$ poles which occurs with the use of a uniform distribution. In the case in which the vectors $\hat{\mathbf{n}}_{i,j}$ are continuously distributed, the elements of the matrix $\psi^H\psi$ are given by

$$\lim_{M,N\to\infty}\left[\psi^H\psi\right]_{m,i} = \lim_{M,N\to\infty}\frac{1}{N}\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N} e^{ik\hat{\mathbf{n}}_{i,j}\cdot(\mathbf{r}_m - \mathbf{r}_i)} \tag{2.31}$$

Here again it can be assumed WLOG that local coordinates can always be introduced such that $\mathbf{r}_m - \mathbf{r}_i = |\mathbf{r}_m - \mathbf{r}_i|\,\hat{k}$ in which case

$$\lim_{M,N\to\infty}\left[\psi^H\psi\right]_{m,i} = \lim_{M,N\to\infty}\frac{1}{2\pi}\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N} e^{ik|\mathbf{r}_m-\mathbf{r}_i|\cos\theta_j}\Delta y\Delta\phi$$

$$= \frac{1}{4\pi}\int_0^{2\pi}\int_{-1}^{1} e^{ik|\mathbf{r}_m-\mathbf{r}_i|\cos\left(\cos^{-1}(y)\right)}d\phi dy$$

$$= \frac{1}{2}\int_0^{\pi} e^{ik|\mathbf{r}_m-\mathbf{r}_i|\cos(\beta)}\sin(\beta)\,d\beta$$

$$= \frac{\sin\left(k\,|\mathbf{r}_m-\mathbf{r}_i|\right)}{k\,|\mathbf{r}_m-\mathbf{r}_i|} \tag{2.32}$$

In three dimensions, the components $\left[\psi^H\psi_0\right]_i$ of the vector $\psi^H\psi_0$ are given (in the continuous limit) by

$$\lim_{N\to\infty}\frac{1}{N}\left[\psi^H\psi_0\right]_i = \frac{\sin\left(k\,|\mathbf{r}_i-\mathbf{r}_0|\right)}{k\,|\mathbf{r}_i-\mathbf{r}_0|} \tag{2.33}$$

Recently, analytical expressions have been derived for the weights in the case of a uniform 27 point stencil. However the expressions are quite cumbersome and will not be repeated here. In this dissertation, these analytical expressions were not used, instead equation 2.11 was inverted numerically.

2.4 Radiation Boundary Conditions

Within the context of the GFD the term radiation boundary condition is taken to mean that at a boundary with outward directed surface normal $\hat{\mathbf{n}}_b$, all the wave vectors $k\hat{\mathbf{n}}$ are exiting the domain i. e., $\hat{\mathbf{n}}_b\cdot\hat{\mathbf{n}}\geq 0$. In addition, the stencils are all restricted to lie within the computational domain. For example, in two-dimensions assume WLOG that

the outer boundary in question is a y constant surface such that $\hat{\mathbf{n}}_b = \hat{\mathbf{j}}$. In this case in order to apply the radiation boundary condition the distribution of the $\hat{\mathbf{n}}_j$ as defined in equation 2.18 is modified as follows

$$\hat{\mathbf{n}}_j = \hat{\mathbf{i}}\cos(\theta_j) + \hat{\mathbf{j}}\sin(\theta_j) \quad ; \quad \theta_j = (j-1)\frac{\pi}{N}; \quad j = 1, 2, 3, ..., N. \tag{2.34}$$

while the stencil of points as defined by the various $\mathbf{r}_i$ in equation 2.22 now becomes

$$\frac{\mathbf{r}_i}{\delta} = \left\{ \begin{array}{ll} -\hat{\mathbf{i}} - \hat{\mathbf{j}} & ; i = 1 \\ -\hat{\mathbf{j}} & ; i = 2 \\ \hat{\mathbf{i}} - \hat{\mathbf{j}} & ; i = 3 \\ \hat{\mathbf{i}} & ; i = 4 \\ -\hat{\mathbf{i}} & ; i = 5 \end{array} \right. \quad \Longrightarrow \quad \mathbf{a} = \left\{ \begin{array}{l} a_1 \\ a_3 \\ a_1 \\ a_2 \\ a_2 \end{array} \right\} \tag{2.35}$$

and the weights $a_1 - a_3$ are computed numerically using equation 2.11. In the case of a corner boundary, say for instance the intersection of a y constant surface ($\hat{\mathbf{n}}_b = \hat{\mathbf{j}}$) with a x constant surface ($\hat{\mathbf{n}}_b = \hat{\mathbf{i}}$), equation 2.34 becomes

$$\hat{\mathbf{n}}_j = \hat{\mathbf{i}}\cos(\theta_j) + \hat{\mathbf{j}}\sin(\theta_j) \quad ; \quad \theta_j = (j-1)\frac{\pi}{2N}; \quad j = 1, 2, 3, ..., N, \tag{2.36}$$

while the stencil of points $\mathbf{r}_i$ is now taken to be

$$\frac{\mathbf{r}_i}{\delta} = \left\{ \begin{array}{ll} -\hat{\mathbf{i}} - \hat{\mathbf{j}} & ; i = 1 \\ -\hat{\mathbf{j}} & ; i = 2 \\ -\hat{\mathbf{i}} & ; i = 3 \end{array} \right. \Longrightarrow \quad \mathbf{a} = \left\{ \begin{array}{l} a_1 \\ a_2 \\ a_2. \end{array} \right\} \tag{2.37}$$

Again the weights a_1 and a_2 are determined numerically using equation 2.11.

The corresponding face, edge and corner radiation conditions are also easily derived for the three-dimensional case. However, as opposed to the two-dimensional case,where the the wave vectors are taken to range over all angles consistent with the radiation condition $\hat{\mathbf{n}}_b \cdot \hat{\mathbf{n}} \geq 0.$, in three dimensions the variables y_j as defined by equation 2.30 are redefined to be

$$y_j = 1 + (j-1)\left(\frac{yrange}{N}\right). \tag{2.38}$$

In the computational experiments detailed in chapter 5 *yrange* is taken to be -1.0 and -0.6. The case $yrange = -1$ corresponds to $\theta_j \in [0, 90^o]$ while $yrange = -0.6$ corresponds to the case $\theta_j \in [0, 66^o]$.

2.5 Domain Decomposition Methods

Having outlined the GFD method in the preceding sections, the question now turns to the issue of the numerical solution of the linear system which arises as the result of applying the GFD. If one were only interested in solving one or modestly sized two-dimensional systems, then one could use highly optimized numerical libraries such as **lapack** [1] or **SuperLU** [10] in order to efficiently solve the system by directly factoring

the coefficient matrix into L and U factors. However, for three-dimensional problems, such an approach is prohibitively expensive as outlined in the work estimates given in chapter 1. In terms of efficient solution techniques for large linear systems, domain decomposition algorithms, which date back the the work of Schwarz [32], in the 1890's are again receiving much attention over one hundred years later[33]. This is motivated by several reasons, the most important being the shift in the emphasis in large scale scientific computing from vector machines with a few very fast, very expensive processors to massively parallel machines with hundreds or even thousands of inexpensive, relatively fast processors [37]. As pointed out in detail by Smith, et al., [33] domain decomposition algorithms are naturally suited for implementation on parallel computers because of their high degree of data locality and associated low communication overhead. However, even on single-processor serial machines domain decomposition methods have tremendous potential to reduce the work required in solving large linear systems. In this section domain decomposition methods as applied to the solution of problems on single grids are reviewed very briefly. A comprehensive treatment of domain decomposition methods including the more general case of overlapping grids may be found in [33].

Consider the following linear system

$$Au = f. \tag{2.39}$$

Assume without loss of generality that the vector of unknowns u is separated into two subsets u_1 and u_2 where, in general, u_1 and u_2 contain common elements. The dimension

of the linear system in question is increased by the number of unknowns in the overlap region as the unknowns are taken to be independent. In general, if equation 2.39 represents the discretization of a partial differential equation on some domain Ω, then the partition of the unknowns into subsets u_1 and u_2 corresponds to a decomposition of the domain Ω into (possibly) overlapping sub-domains Ω_1 and Ω_2. Hence the name domain decomposition. The vector of unknowns is now written as

$$u = \left(u_{1\backslash 2}, u_1 \cap u_2, u_{2\backslash 1}\right)^T \tag{2.40}$$

where $u_{1\backslash 2} = u_1 - u_1 \cap u_2$ etc. This grouping of the unknowns also introduces a natural partitioning of the matrix A as shown in figure 2.1. This allows equation 2.39 to be rewritten as follows

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \tag{2.41}$$

If equation 2.41 were solved using a block stationary iterative method such as block Gauss-Seidel then the following systems of equations would be solved

$$\begin{aligned} A_{11}u_1^n &= f_1 - A_{12}u_2^{n-1} \\ A_{22}u_2^n &= f_2 - A_{21}u_1^n. \end{aligned} \tag{2.42}$$

If one adds and subtracts u_1^{n-1} and u_2^{n-1} from the first and second equations above then

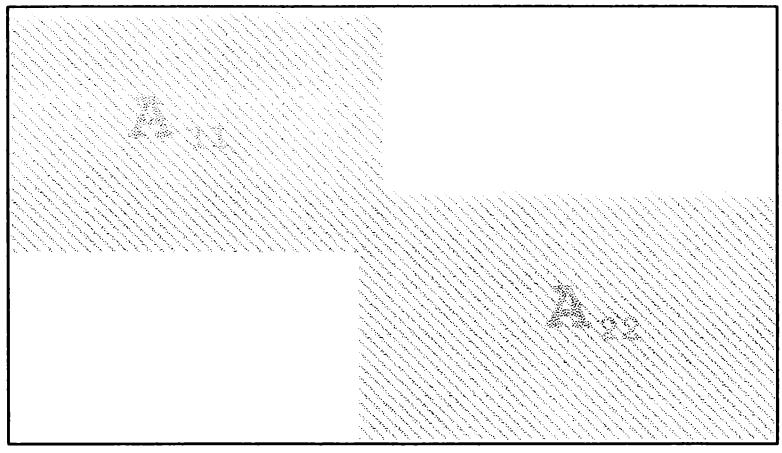


Figure 2.1: Natural partitioning of a Matrix induced by domain decomposition.

the block Gauss–Seidel iterative procedure is written as

$$\begin{aligned} u_1^n &= u_1^{n-1} + A_{11}^{-1}(f_1 - A_{11}u_1^{n-1} - A_{12}u_2^{n-1}) \\ u_2^n &= u_2^{n-1} + A_{22}^{-1}(f_2 - A_{21}u_1^{n} - A_{22}u_2^{n-1}). \end{aligned} \tag{2.43}$$

Now, equation 2.43 looks very similar to a simple iteration procedure [19] where a simple iteration procedure for equation 2.39 is written as follows

$$u^n = u^{n-1} + B^{-1}(f - Au^{n-1}). \tag{2.44}$$

In equation 2.44 B^{-1} is a so called preconditioning matrix. The definition of a good preconditioning matrix is hardware and software dependent, but for the purposes of this work a good preconditioning matrix, B is defined to be a non-singular $N \times N$ matrix which approximates A in the sense that the condition number of $B^{-1}A$ is lower than

that of A, and is relatively easy to invert. i. e., it is relatively easy to solve equations of the type:

$$B\mathbf{y} = \mathbf{c}. \tag{2.45}$$

In addition matrices for which the solution of equation 2.45 displays a good deal of natural parallelism are also considered desirable. In fact, equation 2.43 can be written in the form of equation 2.44 as follows. First define the matrix R_1, whose dimension is say $m \times n$, by its action on a vector u of dimension $n \times 1$ as follows

$$u_1 = R_1 u = \begin{pmatrix} I_{m,m} & 0_{m,n-m} \end{pmatrix} \begin{pmatrix} u_1 \\ \\ u_{2\backslash 1} \end{pmatrix} \tag{2.46}$$

with R_2 defined in a similar manner. Then the Gauss–Seidel iteration procedure given in equation 2.43 becomes

$$\begin{aligned} u^{n+\frac{1}{2}} &= u^n + R_1^T (R_1 A R_1^T)^{-1} R_1 (f - A u^n) \\ u^{n+1} &= u^{n+\frac{1}{2}} + R_2^T (R_2 A R_2^T)^{-1} R_2 (f - A u^{n+\frac{1}{2}}). \end{aligned} \tag{2.47}$$

If one defines

$$B_i = R_i^T (R_i A R_i^T)^{-1} R_i \tag{2.48}$$

where $i = 1, 2$ then the block Gauss-Seidel iteration procedure can be written as follows

$$u^{n+1} = u^n + (B_1 + B_2 - B_2 A B_1)(f - A u^n) \tag{2.49}$$

where in this case the preconditioning matrix is given by

$$B^{-1} = (B_1 + B_2 - B_2 A B_1). \tag{2.50}$$

The preconditioning matrix above is said to be of the **multiplicative Schwarz** type since the matrices B_1 and B_2 multiply each other. In addition one can define matrices of the **additive Schwarz** type by changing the block Gauss-Seidel iteration into a block Jacobi iteration. This is done by rewriting equation 2.43 as follows

$$\begin{aligned} u_1^n &= u_1^{n-1} + A_{11}^{-1}(f_1 - A_{11}u_1^{n-1} - A_{12}u_2^{n-1}) \\ u_2^n &= u_2^{n-1} + A_{22}^{-1}(f_2 - A_{21}u_1^{n-1} - A_{22}u_2^{n-1}). \end{aligned} \tag{2.51}$$

where now equation 2.49 becomes

$$u^{n+1} = u^n + (B_1 + B_2)(f - Au^n) \tag{2.52}$$

and in this case the preconditioning matrix is given by

$$B^{-1} = (B_1 + B_2). \tag{2.53}$$

In closing this section, one final comment is in order. Now that we have defined preconditioning matrices of the additive and multiplicative Schwarz type we have the option of using them not as only as preconditioners in simple iterations but we can also use them

in conjunction with Krylov [29], [19] subspace accelerator schemes. Both approaches are used in the numerical examples discussed in the following chapters.

2.6 Krylov Subspace Methods

Application of the GFD method produces a linear system which is sparse (i. e., possesses a relatively small number of non-zero entries) and highly structured. Given the fact that direct factorization is inappropriate in a large number of cases it seems natural to seek a class of solution methods which take advantage of the sparsity in the GFD matrix. Also, given the fact the cheapest operation that can be performed with a sparse matrix is multiplication of the matrix with a vector, it would be advantageous to use a method which only requires matrix vector products. Such methods do indeed exist, the most popular class in recent times coming under the general heading of Krylov subspace methods [19], [29]. Krylov subspace methods all attempt to solve the linear system

$$A\mathbf{x} = \mathbf{b} \tag{2.54}$$

where $A \in \mathcal{C}^{N \times N}$ and $\mathbf{x}, \mathbf{b} \in \mathcal{C}^N$ by producing a sequence of approximate solutions $\mathbf{x}_n$ to $A^{-1}b$ of the form

$$\mathbf{x}_n \in \mathbf{x}_0 + \mathcal{K}_n(\mathbf{r}_0, A), \quad n = 1, 2, \ldots \tag{2.55}$$

where $\mathbf{x}_0$ is the initial guess for the solution to equation 2.54, $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ is the starting residual, and $\mathcal{K}_n(\mathbf{r}_0, A)$ is the n^{th} Krylov subspace generated by $\mathbf{r}_0$ and A. In

general, for any $\mathbf{c} \in \mathcal{C}^N$ and $B \in \mathcal{C}^{N \times N}$ the m^{th} Krylov subspace of $\mathcal{C}^N$ is defined to be

$$\mathcal{K}_m(\mathbf{c}, B) = \text{span}\{\mathbf{c}, B\mathbf{c}, \ldots, B^{m-1}\mathbf{c}\}. \tag{2.56}$$

Krylov subspace methods may be thought of as seeking to find low-dimensional (as compared to N) polynomial approximations to A^{-1}, i. e.,

$$A^{-1} \approx \phi(A) \; ; \; \phi \in \mathcal{P}_{n-1} \tag{2.57}$$

where $\mathcal{P}_n$ is used to denote the set of complex polynomials of order n i. e.,

$$\mathcal{P}_n = \{\phi(\lambda) \equiv \sigma_0 + \sigma_1\lambda + \ldots + \sigma_n\lambda^n \mid \sigma_0, \sigma_1, \ldots, \sigma_n \in \mathcal{C}\}$$

To this end note that

$$\mathcal{K}_n(\mathbf{r}_0, A) = \{\phi(A)\mathbf{r}_0 \mid \phi \in \mathcal{P}_{n-1}\}. \tag{2.59}$$

Thus the approximate solution $\mathbf{x}_n$ in equation 2.55 can be written as

$$\mathbf{x}_n = \mathbf{x}_0 + \phi(A)\mathbf{r}_0 \tag{2.60}$$

from which it may be easily inferred that the condition for convergence ($\mathbf{x}_n = \mathbf{x}$) is $\phi(A) = A^{-1}$. However, the above argument, while instructive as to how the Krylov subspace methods operate, provides no insight into how to successfully determine the

polynomial $\phi(A)$. In order to do this, it is more instructive to look at the operation of the Krylov solver in terms of the residuals $\mathbf{r}_n$. In particular, it can be shown [16] that

$$\mathbf{r}_n = \mathbf{b} - A\mathbf{x}_n = \psi_n(A)\mathbf{r}_0 \tag{2.61}$$

where $\psi_n \in \mathcal{P}_n$ with $\psi_n(0) = 1$. Given that the goal of all iterative methods is to drive the norm of the residual, $\|\mathbf{r}_n\|$, to zero, all the various Krylov solvers attempt to construct, at each iteration, a polynomial $\psi_n(A)$ which sets $\mathbf{r}_n \approx 0$ in some sense. Some Krylov solvers attempt to minimize an appropriate norm of the residual:

$$\begin{aligned} \|\mathbf{r}_n\| &= \min_{\mathbf{x}_n \in \mathbf{x}_0 + \mathcal{K}_n(\mathbf{r}_0, A)} \|\mathbf{b} - A\mathbf{x}_n\| \\ &= \min_{\psi \in \mathcal{P}_n : \psi(0)=1} \|\psi(A)\mathbf{r}_0\| \end{aligned} \tag{2.62}$$

where $\|\cdot\|$ is a vector norm in $\mathcal{C}^N$. Other Krylov methods operate by imposing a Galerkin-type condition:

$$\mathbf{s}^H \mathbf{r}_n = 0 \text{ for all } \mathbf{s} \in \mathcal{S}_n, \tag{2.63}$$

where $\mathcal{S}_n$ is an n-dimensional subspace of $\mathcal{C}^N$.

Perhaps the most powerful and well known of all the Krylov solver methods is the Conjugate Gradient (CG) Algorithm for positive definite Hermitian matrices due to Hestenes and Stiefel [20]. The CG method possesses two outstanding features which all other Krylov methods attempt to emulate in some form or another. The first feature

deals with the minimization of the residuals [16]

$$\|\mathbf{b} - A\mathbf{x}_n\|_{A^{-1}} = \min_{\mathbf{x}_n \in \mathbf{x}_0 + \mathcal{K}_n(\mathbf{r}_0, A)} \|\mathbf{b} - A\mathbf{x}_n\|_{A^{-1}}, \tag{2.64}$$

where $\|\mathbf{c}\|_{A^{-1}} \equiv \sqrt{\mathbf{c}^H A^{-1}\mathbf{c}}$, while the second desirable quality possessed by the CG algorithm is the fact the the iterates $\mathbf{x}_n$ are generated compactly and efficiently using three-term recurrences, i. e., one only needs to allocate storage for three iterates in order to use the algorithm. Thus the ultimate Krylov solver for non-Hermitian indefinite matrices should share the above properties. However, for general matrices, $\|\cdot\|_{A^{-1}}$ no longer necessarily defines a norm and another norm must be used to define the minimal residual property. Unfortunately [14] it appears that CG–like algorithms exist only for the class of shifted and rotated Hermitian matrices. Thus when designing Krylov methods for non-Hermitian matrices emphasis has been placed on either

(i) the minimal residual property, or

(ii) the short recurrence property

Type (i) based Krylov subspace methods when applied to non-Hermitian matrices suffer from the drawback that their storage requirements grow linearly with the iteration count n. In practice this drawback is alleviated by periodically restarting the algorithm. This can, however, lead to poor convergence on certain classes of difficult problems. Perhaps the most popular of the type (i) algorithms is the GMRES algorithm of Saad and Schultz [30], [31]. Type (ii) based Krylov subspace methods have low storage requirements at the

price of sometimes displaying wildly erratic convergence behavior. In this dissertation the Quasi-Minimal Residual (QMR) method of Freund and Nachtigal is used. The QMR algorithm is of type (ii) and is reviewed in the next section.

2.6.1 Preconditioned Krylov Subspace Methods

In practice, the performance of Krylov subspace methods is found to be strongly dependent on the eigen-structure of A. In order to obtain reasonable results it is often found necessary to use the Krylov subspace methods in conjunction with a good preconditioning matrix. Within the context of Krylov subspace methods, a preconditioning matrix is used to transform the linear system in equation 2.54 into the following system

$$A'\mathbf{x}' = \mathbf{b}' \tag{2.65}$$

where

$$A' = B^{-1}A \ , \ \mathbf{x}' = \mathbf{x} \ , \text{and } \mathbf{b}' = B^{-1}b.$$

A Krylov subspace method applied to to A' generates the following sequence of approximate iterates:

$$\mathbf{x}_n \in \mathbf{x}_0 + \mathcal{K}_n \left(B^{-1}\mathbf{r}_0, B^{-1}A\right), \quad n = 1, 2, \ldots \ . \tag{2.67}$$

In this dissertation, B is always taken to be an additive Schwarz preconditioner of the kind defined in equation 2.53. The additive Schwarz preconditioner is chosen over multiplicative Schwarz preconditioning because the additive Schwarz preconditioners are

naturally parallel. In addition, computational experiments performed using multiplicative Schwarz preconditioners indicated that any improvement in convergence obtained using multiplicative Schwarz preconditioning was always negated by the increase in operation count incurred by the method.

2.7 QMR

The Krylov subspace method chosen for use in this dissertation is the QMR method developed by Freund and Nachtigal [17]. The QMR method uses dual Krylov subspaces generated by the nonsymmetric Lanczos algorithm [22], [23]. The method is closely related to the BiConjugate Gradient (BCG) algorithm of Fletcher [15]. However QMR replaces the original Galerkin type minimization criteria used by the BCG method with a quasi-optimality condition for minimizing the residual within the Krylov subspace. The QMR method was designed to retain the short recurrences of BCG while avoiding the wildly erratic convergence behavior sometimes observed with BCG. Also, to remedy the breakdowns (either exact or numerical) which can occur within the Lanczos algorithm, so-called 'look-ahead' methods have been developed [16] which are used to skip over any non-existent Lanczos vectors. The QMR algorithm produces a sequence of iterates x_n such that

$$\mathbf{x}_n \in \mathbf{x}_0 + \mathcal{K}_n(\mathbf{r}_0, A) \tag{2.68}$$

where $\mathbf{r}_0$ is the starting residual and the basis for the Krylov space $\mathcal{K}_n(\mathbf{r}_0, A)$ is generated using the nonsymmetric Lanczos algorithm. The nonsymmetric Lanczos algorithm is a

procedure which can be viewed as building two sets of biorthogonal basis vectors

$$\{\mathbf{v}_j\}_{j=1}^n \text{ and } \{\mathbf{w}_j\}_{j=1}^n$$

such that for $i = 1, 2, \ldots, n$

$$\mathbf{v}_i \in \mathcal{K}_n(\mathbf{v}_1, A) \tag{2.70}$$

$$\mathbf{w}_i \in \mathcal{K}_n\left(\mathbf{w}_1, A^H\right).$$

The biorthogonality relation implies that

$$\langle \mathbf{w}_i, \mathbf{v}_j \rangle \equiv \mathbf{v}_i^H \mathbf{w}_j = \delta_{ij}. \tag{2.71}$$

The nonsymmetric Lanczos algorithm is briefly summarized as follows:

NONSYMMETRIC LANCZOS ALGORITHM.

Given $\mathbf{r}_0$ and $\hat{\mathbf{r}}_0$ with $\langle \mathbf{r}_0, \hat{\mathbf{r}}_0 \rangle \neq 0$, set $\mathbf{v}_1 = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|}$ and $\mathbf{w}_1 = \frac{\hat{\mathbf{r}}_0}{\langle \hat{\mathbf{r}}_0, \mathbf{v}_1 \rangle}$. Set $\mathbf{v}_0 = \mathbf{w}_0 = 0$ and $\beta_0 = \gamma_0 = 0$.

for $n = 1, 2, \ldots$

Compute $A\mathbf{v}_n$ and $A^H \mathbf{w}_n$

set $\alpha_n = \langle A\mathbf{v}_n, \mathbf{w}_n \rangle$

set $\tilde{\mathbf{v}}_{n+1} = A\mathbf{v}_n - \alpha_n \mathbf{v}_n - \beta_{n-1}\mathbf{v}_{n-1}$

set $\tilde{\mathbf{w}}_{n+1} = A^H \mathbf{w}_n - \overline{\alpha}_n \mathbf{w}_n - \gamma_{n-1}\mathbf{w}_{n-1}$

set $\gamma_n = \|\tilde{\mathbf{v}}_{n+1}\|$ and $\mathbf{v}_{n+1} = \frac{\tilde{\mathbf{v}}_{n+1}}{\gamma_n}$

set $\beta_n = \langle \tilde{\mathbf{w}}_{n+1}, \mathbf{v}_{n+1} \rangle$ and $\mathbf{w}_{n+1} = \frac{\tilde{\mathbf{w}}_{n+1}}{\beta_n}$

The coefficients α_n, β_n and γ_n are chosen such that $\|\mathbf{v}_n\| = 1$ and to ensure that the biorthogonality constraint (equation 2.71) is satisfied.

The Lanczos algorithm can break down in two different situations. The first occurs when $\tilde{\mathbf{v}}_{n+1} = 0$ or $\tilde{\mathbf{w}}_{n+1} = 0$. If $\tilde{\mathbf{v}}_{n+1} = 0$ then the right Lanczos vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ form an A–invariant subspace, while if $\tilde{\mathbf{w}}_{n+1} = 0$ then the left Lanczos vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n$ form an A^H–invariant subspace. These cases are referred to as regular termination. The second breakdown, referred to as serious breakdown by Wilkinson [38], occurs when $\beta_n = 0$ but neither $\tilde{\mathbf{v}}_{n+1} = 0$ or $\tilde{\mathbf{w}}_{n+1} = 0$. *Look-ahead* Lanczos methods [16] have been developed to treat the serious breakdowns. However with the domain decomposition based preconditioners used in this work serious breakdowns are not observed and so the look-ahead Lanczos methods will not be reviewed here.

The recurrence relations expressed in the above Lanczos algorithm for the $\mathbf{v}_i$ can be written compactly using matrix notation by first defining the matrix V_n where the

columns of V_n are the given by the Lanczos vectors $\mathbf{v}_j$, $j = 1, 2, \ldots, n$, i. e.,

$$V_n = [\mathbf{v}_1 \ \mathbf{v}_2 \ \ldots \ \mathbf{v}_n] \ . \tag{2.74}$$

Also define the matrices H_n^e and H_n as follows

$$H_n^e = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \gamma_1 & \alpha_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_{n-1} \\ \vdots & & \ddots & \gamma_{n-1} & \alpha_n \\ 0 & \cdots & \cdots & 0 & \gamma_n \end{pmatrix} \in C^{N+1 \times N} \tag{2.75}$$

and

$$H_n = [I_n^e \ 0] \, H_n^e, \tag{2.76}$$

i. e., the matrix H_n is equal to H_n^e minus the last row. Then the recurrences in the Lanczos process can be expressed as

$$AV_n = V_{n+1} H_n^e \ . \tag{2.77}$$

Similar relations exist for the $\mathbf{w}_i$. Once the above information is produced by the Lanczos process, the QMR iterates are computed by first expressing the residual $\mathbf{r}_n$ for

the n^{th} iterate

$$\mathbf{x}_n = \mathbf{x}_0 + V_n \mathbf{z}_n, \text{ where } \mathbf{z}_n \in C^N \tag{2.78}$$

as

$$\mathbf{r}_n = \mathbf{b} - A\mathbf{x}_n = \mathbf{r}_0 - AV_n\mathbf{z}_n = V_{n+1}\left(\mathbf{d}_n - H_n^e \mathbf{z}_n\right) \tag{2.79}$$

where

$$\mathbf{d}_n = [\|\mathbf{r}_0\|_2 \; 0 \; \ldots \; 0]^T \; \in \mathcal{R}^{N+1}. \tag{2.80}$$

The residual is quasi-minimized by choosing the $\mathbf{z}_n$ which satisfies the following least squares problem:

$$\|\mathbf{d}_n - H_n^e \mathbf{z}_n\|_2 = \min_{\mathbf{z}_n \in C^N} \|\mathbf{d}_n - H_n^e \mathbf{z}_n\|_2 \; . \tag{2.81}$$

The residual is said to be quasi-minimized as opposed to minimized (as in the GMRES algorithm) as the columns of V_{n+1} are not orthogonal and so

$$\|V_{n+1}\left(\mathbf{d}_n - H_n^e \mathbf{z}_n\right)\|_2 \neq \|\mathbf{d}_n - H_n^e \mathbf{z}_n\|_2 \; . \tag{2.82}$$

Finally, a theorem due to Freund and Nachtigal [17] is reviewed as it will be useful in interpreting some of the computational experiments in the next chapter

Theorem 2.1: *If the $N \times N$ matrix H_n generated by n steps of the Lanczos algorithm is diagonalizable, then the residual vectors of the QMR algorithm satisfy*

$$\|\mathbf{r}_n\|_2 \leq \|\mathbf{r}_0\|_2 \kappa\left(H_n\right) \sqrt{n+1} \epsilon^{(n)} \tag{2.83}$$

where

$$\epsilon^{(n)} = \min_{p(0)=1, \lambda \in \lambda(A)} \max |p(\lambda)|, \tag{2.84}$$

p is a polynomial of degree n, $\lambda(A)$ denotes the eigen spectrum of A and $\kappa(\cdot)$ denotes the condition number with respect to the 2-norm.

In the above theorem the constant $\epsilon^{(n)}$ is a measure of how well the function zero can be approximated on the spectrum of A by polynomials of degree n normalized to one at zero. Thus if the spectrum of A is clustered around a few values, then $\epsilon^{(n)}$ will be small implying fairly rapid convergence of the QMR algorithm. In particular, if A is a rank p perturbation of I then A has at most p+1 distinct eigenvalues, which implies that the QMR algorithm will converge in at most p+1 steps.

Chapter 3

One-Dimensional Test Cases

3.1 One-Dimensional GFD

The one-dimensional Helmholtz equation is given by

$$\frac{d^2\phi}{d^2x} + k^2\phi = 0. \tag{3.1}$$

Fundamental solutions to the one-dimensional Helmholtz equation are given by Ae^{ikx} and Be^{-ikx}. Here $i = \sqrt{-1}$, $k = \frac{2\pi}{\lambda}$, Ae^{ikx} is a right running wave with amplitude A, and Be^{-ikx} is a left running wave with amplitude B. These fundamental solutions to the Helmholtz equation are used to develop a one-dimensional version of the GFD method outlined in chapter 2 as follows. First assume that the solution in the interval

$[x_{i-1}, x_{i+1}]$ can be written as a combination of the fundamental solutions as shown below

$$\phi = Ae^{ikx} + Be^{-ikx} \tag{3.2}$$

The coefficients A and B are determined at each grid point x_i by forcing equation 3.2 to the match the solution at nodes x_{i-1} and x_{i+1} as follows

$$\begin{aligned} \phi_{i-1} &= Ae^{-ik\delta} + Be^{ik\delta}, \\ \phi_{i+1} &= Ae^{ik\delta} + Be^{-ik\delta}. \end{aligned} \tag{3.3}$$

In equation 3.3 it has been assumed without loss of generality that the grid is uniform with grid spacing δ. The set of equations given in equation 3.3 is easily solved to yield

$$\begin{aligned} A &= \frac{e^{-ik\delta}\phi_{i-1} - e^{ik\delta}\phi_{i+1}}{e^{-2ik\delta} - e^{2ik\delta}}, \\ B &= \frac{e^{ik\delta}\phi_{i-1} - e^{-ik\delta}\phi_{i+1}}{e^{2ik\delta} - e^{-2ik\delta}}. \end{aligned} \tag{3.4}$$

These expressions are inserted into equation 3.2 which, when evaluated at $x = 0$, yields the following expression for ϕ_i

$$\phi_i = \left(\frac{e^{-ik\delta} - e^{ik\delta}}{e^{-2ik\delta} - e^{2ik\delta}}\right)\phi_{i-1} + \left(\frac{e^{-ik\delta} - e^{ik\delta}}{e^{-2ik\delta} - e^{2ik\delta}}\right)\phi_{i+1} \tag{3.5}$$

which can be simplified to yield the desired discretization formula

$$2cos(k\delta)\phi_i = \phi_{i-1} + \phi_{i+1}. \tag{3.6}$$

Radiation boundary conditions of the kind derived in section 2.4 are also easily derived in this context by assuming that at an x_{max} boundary

$$\phi = Ae^{ikx} \Rightarrow \phi_{i-1} = Ae^{-ik\delta} \Rightarrow -e^{ik\delta}\phi_{i-1} + \phi_i = 0. \tag{3.7}$$

A similar relation is easily derived for an x_{min} boundary.

In the numerical examples presented in section 3.3, equation 3.1 is solved subject to the following Dirichlet boundary conditions

$$\begin{aligned} \phi(0) &= 1 \\ \phi(1) &= 0. \end{aligned} \tag{3.8}$$

This leads to the following tridiagonal linear system

$$\begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ -1 & 2cos(k\delta) & -1 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & -1 & 2cos(k\delta) & -1 \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{np-1} \\ \phi_{np} \end{pmatrix} = \begin{pmatrix} 1.0 \\ 0.0 \\ \vdots \\ 0.0 \\ 0.0 \end{pmatrix}. \tag{3.9}$$

In this case $\delta = \frac{1}{np-1}$ and $x_i = (i-1)\delta$.

3.2 Domain Decomposition Based Considerations

Before proceeding to discuss the computations, some special considerations in applying domain decomposition techniques to the Helmholtz equation must be addressed.

The submatrices

$$R_i A R_i^T \tag{3.10}$$

which appear in the definition of the preconditioning matrices B_i in equation 2.48 in section 2.5 present certain problems when applied to the discretized Helmholtz equation. This is because these submatrices all have Dirichlet boundary conditions on the artificial boundaries which separate the various domains. However, the Laplace operator possesses eigen-solutions

$$\frac{d^2\phi}{d^2x} = -\lambda^2\phi$$

in any geometry under the assumption of Dirichlet boundary conditions. This implies that the Helmholtz operator associated with the subproblem is undefined for those wave numbers k which correspond to eigenspectrum of the Laplace operator, i. e.,

$$\begin{aligned} \tfrac{d^2\phi}{d^2x} + k^2\phi &= 0 \\ \text{or} & \\ -\lambda^2\phi + k^2\phi &= 0 \end{aligned}$$

which implies that

$$0\phi = 0$$

for $k = \lambda$. In other words the sub-problems contains resonant modes or in the language of linear algebra the null space of $R_i A R_i^T$ is non-empty. This fact holds true even when A in non-singular and hence should have a unique solution. This can be a problem for both block iterative methods as well as Krylov subspace methods which use the B_i matrices as part of a preconditioning matrix. In addition, when using block iterative methods the Dirichlet boundary conditions on the sub-domain faces causes waves to be reflected at the artificial boundaries. Thus energy becomes trapped within each domain and the solution diverges. As was first demonstrated by Després [11], the key to ensuring the successful application of domain decomposition based methods to the Helmholtz equation is that each sub-domain matrix $R_i A R_i^T$ have no resonant modes. One way to do this is to use some form of radiation boundary condition at the artificial (interface) boundaries. Within the context of 1D-GFD this idea is implemented as follows. First assume that the global domain has been partitioned into ND overlaping sub-domains and that each sub-domain contains ns points. In addition define ϕ_j^N to be the jth point in sub-domain N where points in the global domain are related to points in the sub-domains as follows :

$$\phi_j^N = \phi_{j_G} \; ; \; j_G = [j + (N-1) * (ns - overlap)] \tag{3.14}$$

where each domain overlaps its neighboring domains by *overlap* points. Also the unknowns in the overlap region between say zones N and $N+1$ for the case $overlap = 2$

are related as follows:

$$\phi^N_{ns-1} = \phi^{N+1}_1 \quad and \quad \phi^N_{ns} = \phi^{N+1}_2.$$

However all unknowns in the overlap regions are treated as independent. This situation is depicted graphically in figure 3.1. For sake of brevity only the max boundary for domain 1 is treated. The other boundaries are all treated by analogy. If no change is made to equation 3.5, then ϕ^1_{ns} satisfies the following equation

$$-\phi^1_{ns-1} + 2cos(k\delta)\phi^1_{ns} = \phi^2_3. \tag{3.16}$$

In general one could move ϕ^2_3 onto the right hand side of the matrix equation. However, as mentioned earlier this method leads to very rapidly diverging solutions. However, as the unknowns in the overlap region are treated as independent variables, one can write

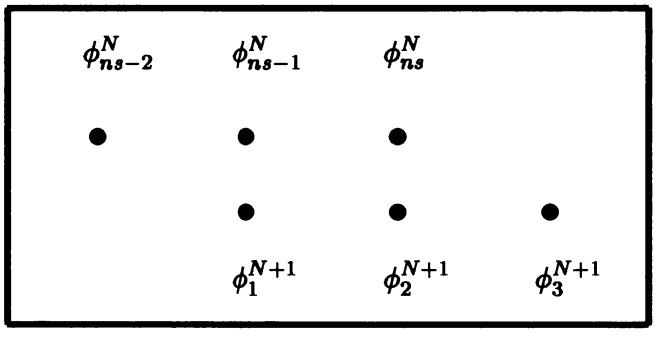


Figure 3.1: One-dimensional overlap region ($overlap = 2$) .

the following difference equation for ϕ^1_{ns}

$$-e^{ik\delta}\phi^1_{ns-1} + \phi^1_{ns} = \left(\frac{e^{-ik\delta} - e^{ik\delta}}{e^{-2ik\delta} - e^{2ik\delta}}\right)\phi^2_1 + \left(\frac{e^{-ik\delta} - e^{ik\delta}}{e^{-2ik\delta} - e^{2ik\delta}}\right)\phi^2_3 - e^{ik\delta}\phi^2_1. \qquad (3.17)$$

In equation 3.17 ϕ^1_{ns} is coupled to domain 2 using equation 3.5 and domain 1 is coupled to domain 2 by using the radiation boundary condition developed in equation 3.7. This boundary condition suffices to ensure that $R_1 A R_1^T$ has no resonant modes as well as guaranteeing that the solution for ϕ^1_{ns} is the same as if the artificial boundary were not present. Hence this boundary condition should be efficient for use with either block iterative methods or Krylov subspace methods. Finally note that since ϕ^2_2 still satisfies equation 3.5, equation 3.17 can be simplified to

$$e^{ik\delta}\phi^1_{ns-1} - \phi^1_{ns} = e^{ik\delta}\phi^2_1 - \phi^2_2. \qquad (3.18)$$

This is the form of the ARBC used in all the computations.

3.3 Numerical Examples

In this section equation 3.9 is solved on a grid composed of 92 ($= np$) grid points. In each case the unknowns are partitioned into 10 sub-domains with 11 points in each domain. This corresponds to a 2 point overlap.

First, solutions to equation 3.9 using a block symmetric Gauss-Seidel iterative scheme which makes use of equation 3.18 are presented. The convergence rate for $\lambda = 4.0\delta$ is

shown in figure 3.2 while the convergence rate for $\lambda = 4.03\delta$ is shown in figure 3.3. Note the dramatic drop off in convergence rate as λ approaches the value 4.04444δ at which A becomes singular.

Next, solutions obtained using the QMR Krylov subspace solver taken from the **QMRPACK** software package of Freund and Nachtigal [18], used in conjunction with an additive Schwarz preconditioner, are presented. Here the preconditioner is given by

$$B^{-1} = B_1 + B_2 \cdots + B_9 + B_{10} \tag{3.19}$$

where the B_i are defined as in equation 2.48. In this case equation 3.18 is not used. For the case where $\lambda = 4.0\delta$, which is shown in figure 3.4, the preconditioned QMR algorithm shows good convergence up to the point at which the algorithm breaks down. This case corresponds to a situation in which the submatrices are numerically singular. In contrast the $\lambda = 4.03\delta$ case, which is shown in figure 3.5, shows outstanding convergence. This convergence is characteristic of all wave numbers which are removed from those wave numbers where the submatrices are singular or very ill conditioned. This is also true regardless of the size of delta.

3.4 Spectral Analysis

In order to get some perspective of these results it is useful to review the eigenspectrum of the preconditioned matrices, $B^{-1}A$ used in these examples. First however, it is

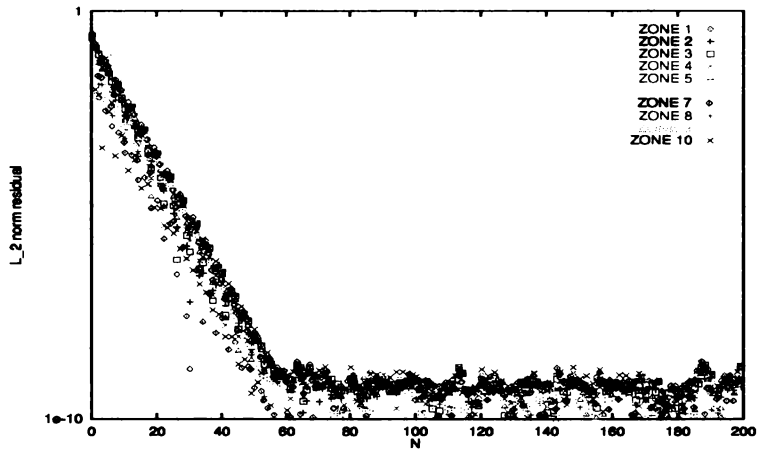


Figure 3.2: Convergence rate using symmetric block Gauss-Seidel scheme with radiation boundary interface condition ($\lambda = 4.0\delta$).

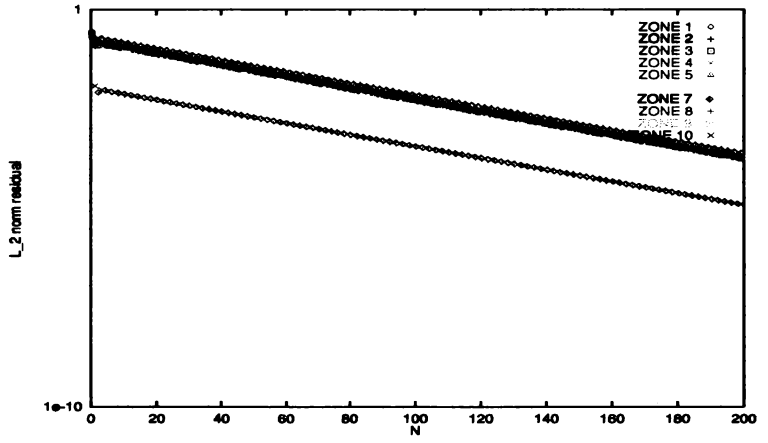


Figure 3.3: Convergence rate using symmetric block Gauss-Seidel scheme with radiation boundary interface condition ($\lambda = 4.03\delta$).

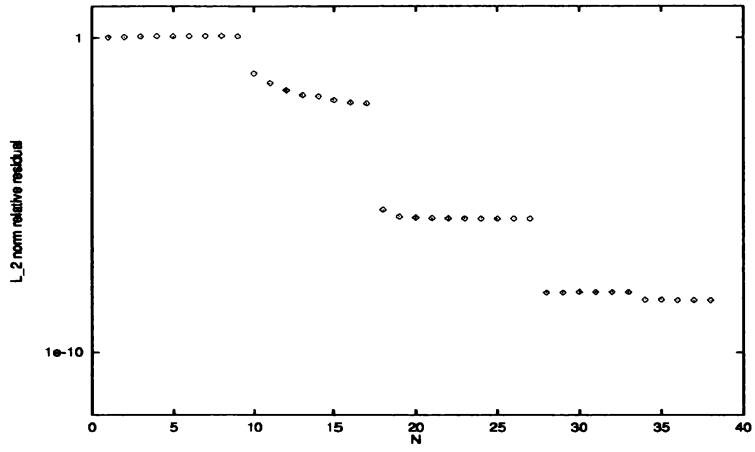


Figure 3.4: Convergence rate using QMR Krylov solver with additive Schwarz preconditioner ($\lambda = 4.0\delta$).

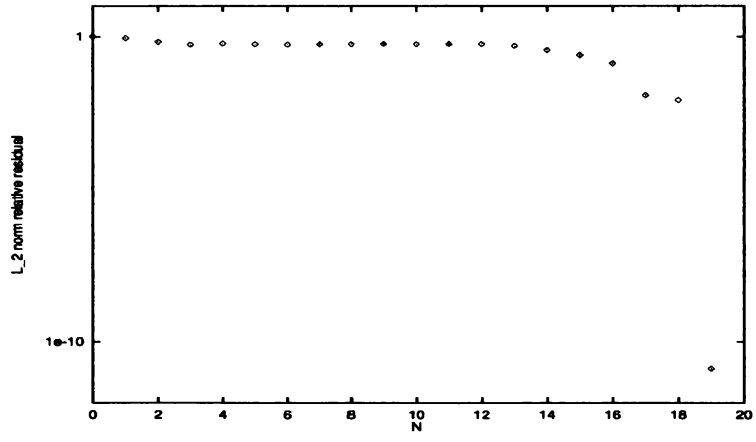


Figure 3.5: Convergence rate using QMR Krylov solver with additive Schwarz preconditioner ($\lambda = 4.03\delta$).

instructive to look at a generic structure plot (i. e., the pattern of non-zero entries) common to all of of these matrices. This pattern is shown in figure 3.6. All the elements on the diagonals of $B^{-1}A$ are equal to one. The fact the the preconditioned matrix is a rank p perturbation of the identity matrix I is easily seen by an examination of the structure plot. In this case $p = 36$. This pattern is the same whether equation 3.18 is used at the domain boundaries or not. Next, a graph of the magnitudes of the eigenspectrum of the the preconditioned matrices corresponding to $\lambda = 4.0\delta$, and $\lambda = 4.03\delta$ are shown in figure 3.7. In this case, the ARBC is used. The block Gauss-Seidel iterative method is convergent as long as the magnitudes of the eigenspectrum of $B^{-1}A$ is contained in the interval $[0.0, 2.0]$ [19], or in other words, all of the eigenvalues of $B^{-1}A$ must lie in the unit disk centered at $(1.0, 0.0)$ in the complex plane. As can be seen in figure 3.7 the ends of the spectrum are moving closer to the extreme values of 0.0 and 2.0 in the $\lambda = 4.03\delta$ case. This explains the observed drop-off in the convergence rate observed when using the block Gauss-Seidel iterative method.

Finally, a plot is shown in figure 3.8 of the spectral radii vs. the number of points per wavelength of both the preconditioned matrices which use the ARBC as well as those that do not. As can be seen in figure 3.8, the spectral radii of the matrices which use the ARBC are largely unaffected by the number of points used in the computations, while those that do not show a large variation with the PPW used and can become severely ill-conditioned for several wave number ranges. However, the QMR algorithm still converges quite well without the use of the ARBC modified preconditioner.

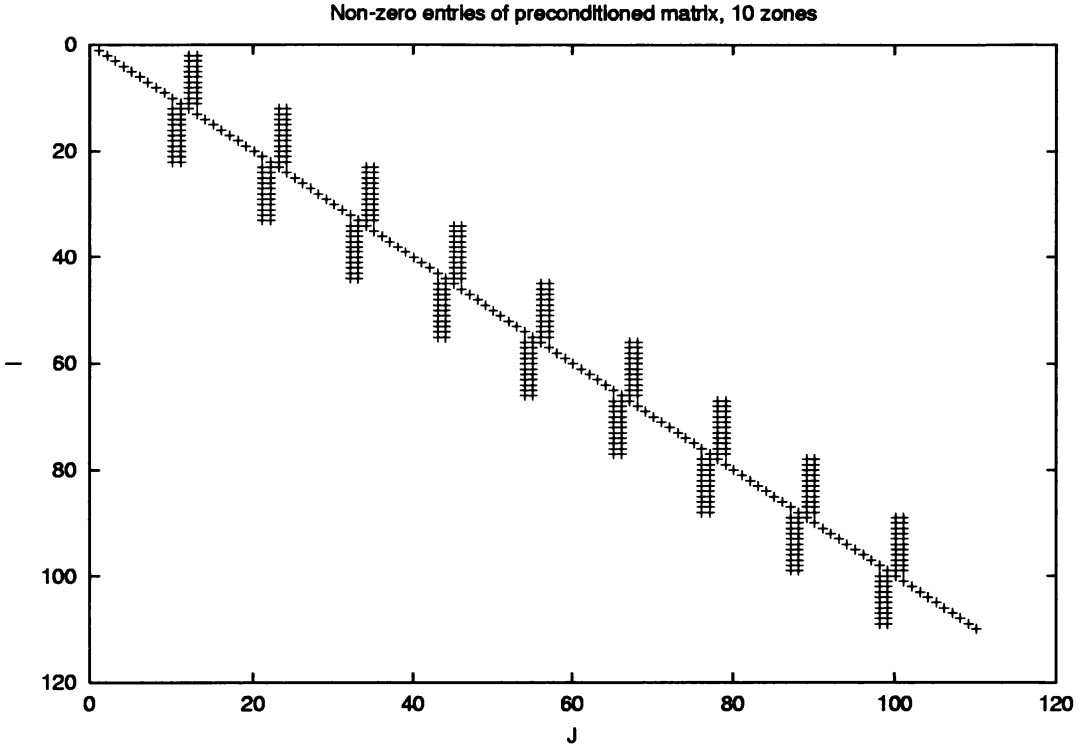


Figure 3.6: Structure plot of the preconditioned matrix.

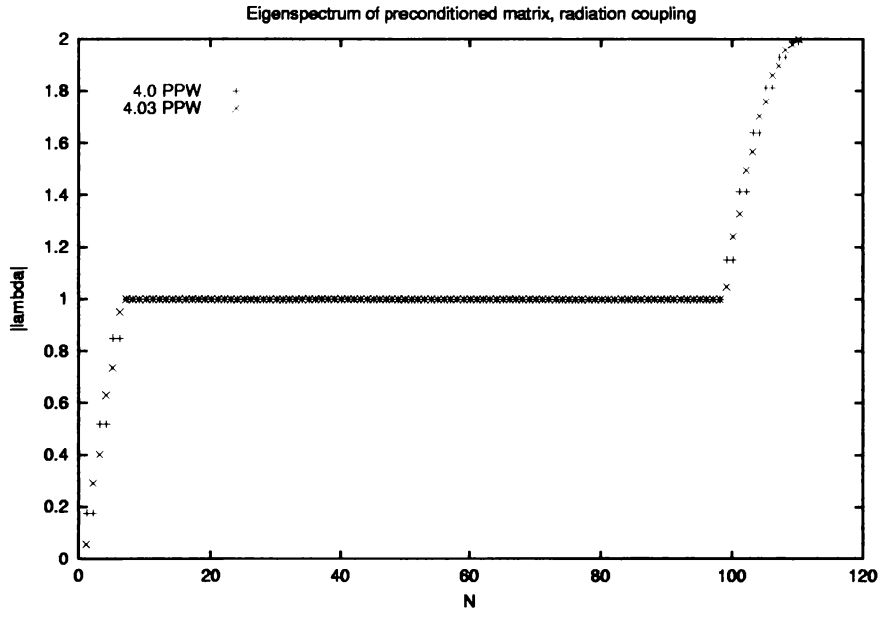


Figure 3.7: Eigenspectrum of the preconditioned matrix using equation 3.18.

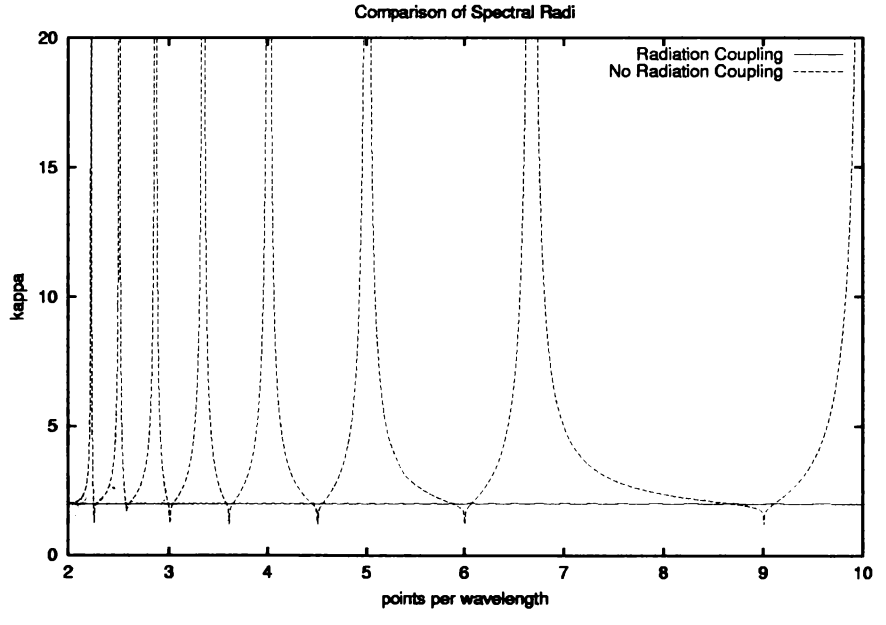


Figure 3.8: Spectral radii of the preconditioned matrices.

This can be understood in light of Theorem 2.1. Given that the preconditioned matrix is a rank p perturbation of I, convergence is always guaranteed in less than $p+1$ steps.

However, in multiple dimensions p can be quite large and the additional improvement in the conditioning of $B^{-1}A$ obtained when use is made of equation 3.18 is necessary in order to obtain good convergence. This is the approach used in all of the multi-dimensional computations.

Chapter 4

Two-Dimensional Test Cases

4.1 Problem Description

In this chapter, Helmholtz's equation for the acoustic potential ϕ

$$\nabla^2\phi + k^2\phi = 0 \tag{4.1}$$

is solved on a two-dimensional computational domain $\mathcal{D}$. In this case, $\mathcal{D}$ is taken to be the unit square $[0,1] \times [0,1]$ in the $x-y$ plane. The test problem is chosen to be that of the free space sound field created by three point source radiators located at $\mathbf{r}_1 = (-0.25, -0.25)$, $\mathbf{r}_2 = (0.0, -0.25)$, and $\mathbf{r}_3 = (0.25, -0.25)$. The problem setup is shown in figure 4.1.

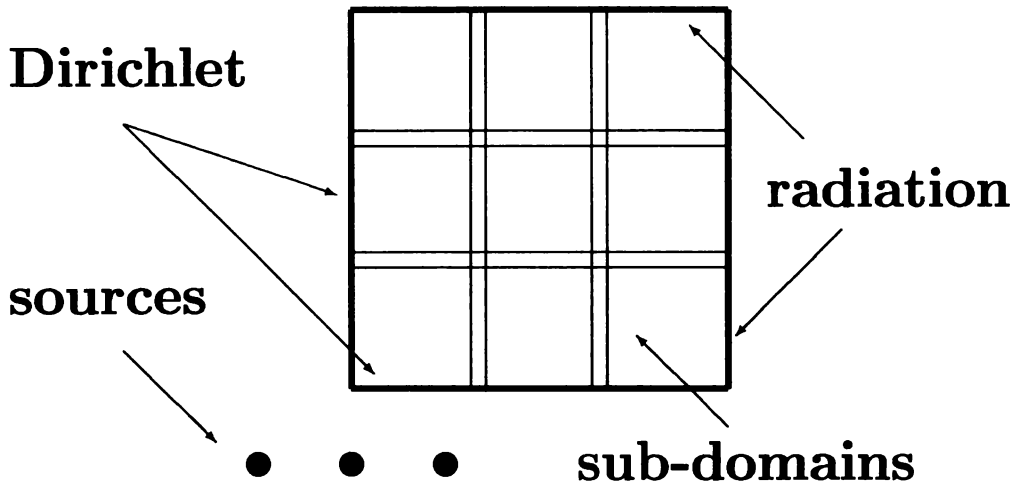


Figure 4.1: Problem setup for the two-dimensional test cases.

The analytical solution [12] to equation 5.1 in this case is given by

$$\phi(\mathbf{r}) = \sum_{i=1}^{3} \frac{\sqrt{-1}}{4} H_0^2(k\|\mathbf{r} - \mathbf{r}_i\|) \tag{4.2}$$

The problem is discretized using the GFD method outlined in chapter 2. The computational domain $\mathcal{D}$ is discretized using $np = 98$ points in each direction and $(x_i, y_j) = ((i-1)\delta, (j-1)\delta)$ where $\delta = \frac{1}{np-1}$. Dirichlet boundary conditions are applied on the $x = 0$ and $y = 0$ boundaries, while the one-sided 5 point (edge) and 3 point (corner) ARBCs developed in section 2.4 are used on the $x = 1$ and $y = 1$ edges and the $(x, y) = (1, 1)$ corner respectively. The linear system produced by the GFD process is solved using the QMR solver from **QMRPACK** [18]. Computations are performed using additive Schwarz preconditioners using differing number of subdomains and points per wavelength as outlined in table 4.1. In table 4.1, ND_x is the number of subdomains

Table 4.1: Summary of various two-dimensional test cases.

Case	ND_X	ND_y	np_x	np_y	ppw	preconditioning
1	1	1	98	98	3	none
2	1	1	98	98	15	none
3	2	1	50	98	3	additive Schwarz
4	2	1	50	98	15	additive Schwarz
5	2	2	50	50	3	additive Schwarz
6	2	2	50	50	15	additive Schwarz
7	4	4	26	26	3	additive Schwarz
8	4	4	26	26	15	additive Schwarz
9	12	12	10	10	3	additive Schwarz
10	12	12	10	10	15	additive Schwarz

used in the x direction, np_x is the number of points used in each subdomain in the x direction, and ND_y and np_y are the corresponding quantities in the y direction. The preconditioners used are again of the type

$$B^{-1} = \sum_{i=1}^{ND_X \times ND_y} B_i \tag{4.3}$$

where the B_i are defined as in equation 2.48. The submatrices $R_i A R_i^T$ are directly factored using the efficient fill-reducing sparse matrix LU factorization routines taken from the **SuperLU** [10] software package. A special C++ programming language routine written by the author to drive the factorization and back-substitution routines from **SuperLU** is used in conjunction **QMRPACK**. This version of **QMRPACK** was also modified by the author to make use of additive Schwarz preconditioning.

In addition, in the cases where multiple subdomains are used, the one-sided ARBCs derived in section 2.4 are used to develop the corresponding two-dimensional inter-

domain boundary conditions corresponding to equation 3.18. Let $\mathcal{D}_i$ be a subdomain a portion of whose boundary $\mathcal{E}_{ij} \subset \mathcal{B}(\mathcal{D}_i)$ (which is here assumed without loss of generality to be an edge) overlaps with subdomain $\mathcal{D}_j$. In addition, let ϕ_I be the potential at a point I on $\mathcal{E}_{ij}$. Then in order to develop an inter-domain boundary condition which prevents the submatrices $R_i A R_i^T$ from possessing resonant modes, the following discretization formula is used at I

$$\boldsymbol{\phi}_I^T \mathbf{a}^e - \phi_I = \boldsymbol{\phi}_J^T \mathbf{a}^e - \phi_J, \tag{4.4}$$

where ϕ_J is the potential at the point J in $\mathcal{D}_j$ which coincides with the point I in $\mathcal{D}_i$. In addition $\mathbf{a}^e$ are the weights defined in equation 2.35. The corresponding formula for corner points is also easily derived.

4.2 Numerical Results

In order to assess how well the additive Schwarz preconditioner performs, plots are shown in figures 4.2 and 4.3 of the convergence rates of the L_2 norm of the relative residuals $\frac{\|\mathbf{r}_n\|_2}{\|\mathbf{r}_0\|_2}$ for cases 1 and 2 and 9 and 10. Cases 1 and 2 are single grid cases using 3 and 15 PPW for which the unpreconditioned QMR algorithm is used while cases 9 and 10 are 144 zone cases and for which the additive Schwarz preconditioner is used.

As can be seen the additive Schwarz preconditioner accelerates the convergence of the QMR method dramatically in this case. A comparison of the convergence rates for cases 9 and 10 is shown in figure 4.3. The additive Schwarz preconditioner in this case appears to be highly insensitive to the points per wavelength used or equivalently the

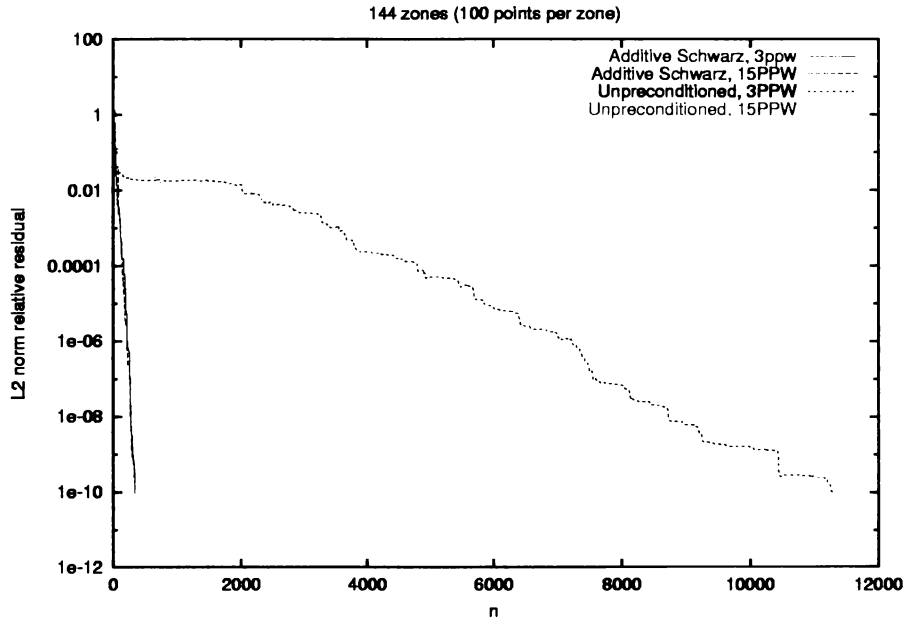


Figure 4.2: Convergence rates for various test cases.

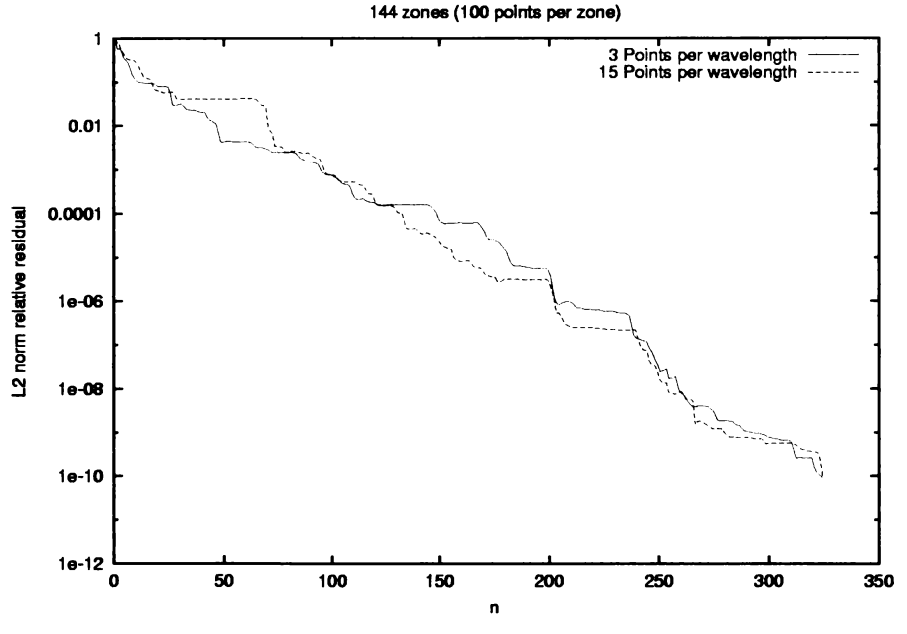


Figure 4.3: Comparison of the convergence rates for test cases 9 and 10 vs. ppw.

wave number k.

Plots of comparisons of the computed and the exact solutions are shown along the line $y = 1$ in figures 4.4 and 4.5 for the 3 ppw and 15 ppw cases respectively. The ARBC developed in section 2.4 is seen to produce quite accurate results for both resolutions.

Comparisons of the convergence rates vs. the number of zones are shown for the 3 ppw cases in figure 4.6 and for the 15 ppw cases in figure 4.7. A comparison of the cpu times required to reduce the L2 norm of the relative residual 10 orders of magnitude for the various cases are shown in table 4.2. All the cpu times were obtained using a DELL inspiron 4000 laptop computer with an intel 900 MHz pentium III cpu running red hat linux 7.0 using the native gnu g77 and gcc compilers. Also displayed in table 4.2 is a column of the elapsed cpu time divided by the number of zones. These numbers represent the amount of speedup that would be obtainable on a perfectly parallel computer. It can clearly be seen that the reduction in convergence rate inherent in increasing the number of zones is offset by the amount of wall clock time reduction that could be achieved by running the computation in parallel. On a real parallel system, the optimal number of zones would be a function of the communication overhead, the amount of memory per processor, et cetera rather than being simply a matter of using as many zones as possible. In all cases, however, the convergence rates appear to be quite reasonable.

As a closing note it should be mentioned that an effort was made to assess how standard preconditioners perform on these test problems Computations were performed using the ILUT preconditioner [28] which is included in the standard release of **QM-**

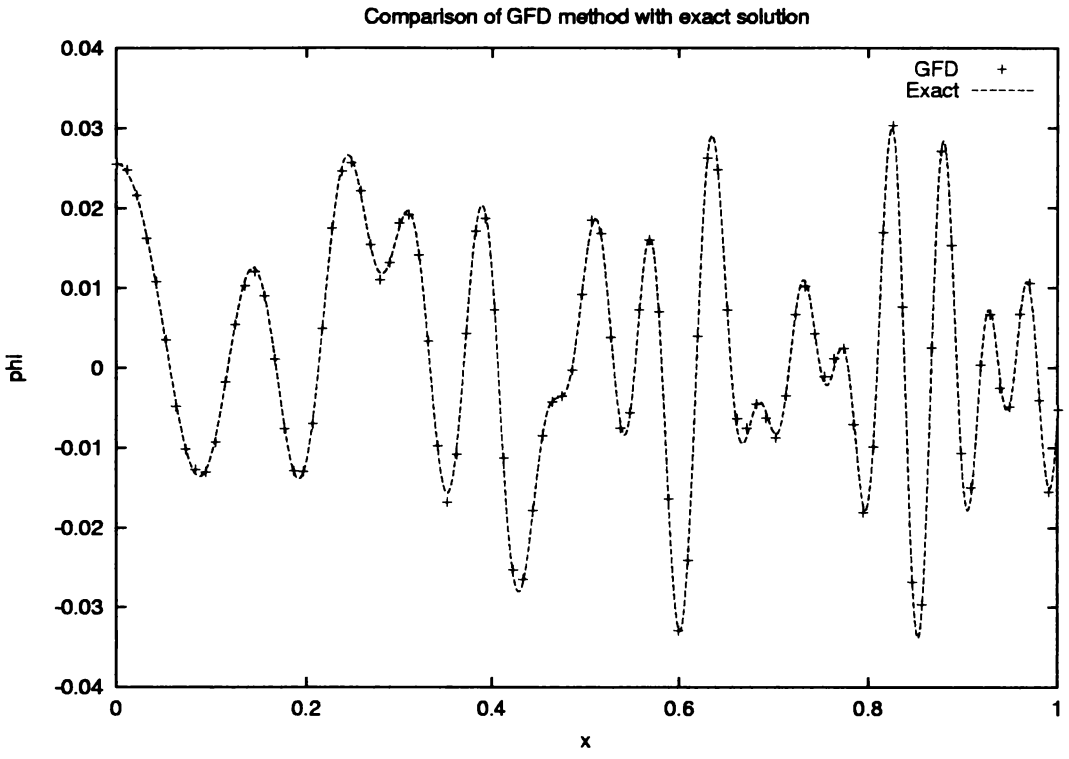


Figure 4.4: Comparison of computed and exact solution along the line $y = 1$, 3ppw.

Table 4.2: Summary of cpu usage of the various two-dimensional test cases.

Case	CPU time (sec)	$\frac{\text{cpu time}}{\text{No. Zones}}$
1	545	545
2	255	255
3	30.47	15.24
4	31.56	15.78
5	23.98	5.99
6	25.33	6.33
7	34.7	2.17
8	38.8	2.43
9	65	0.451
10	65	0.451

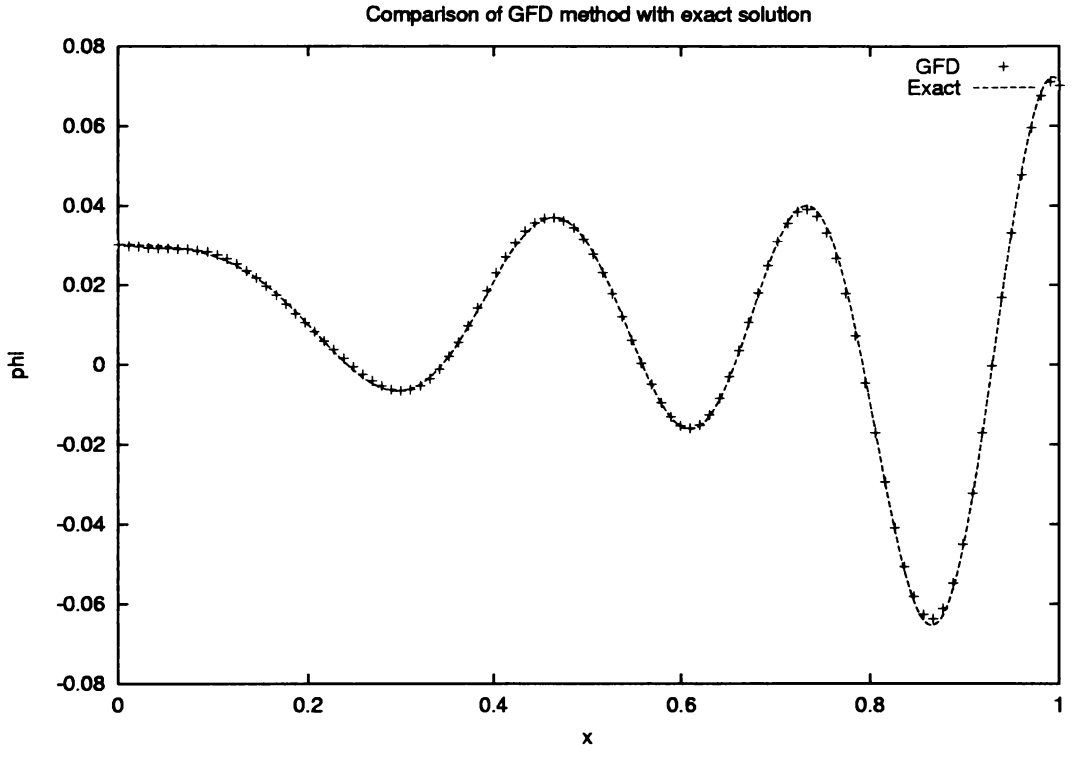


Figure 4.5: Comparison of computed and exact solution along the line $y = 1$, 15ppw.

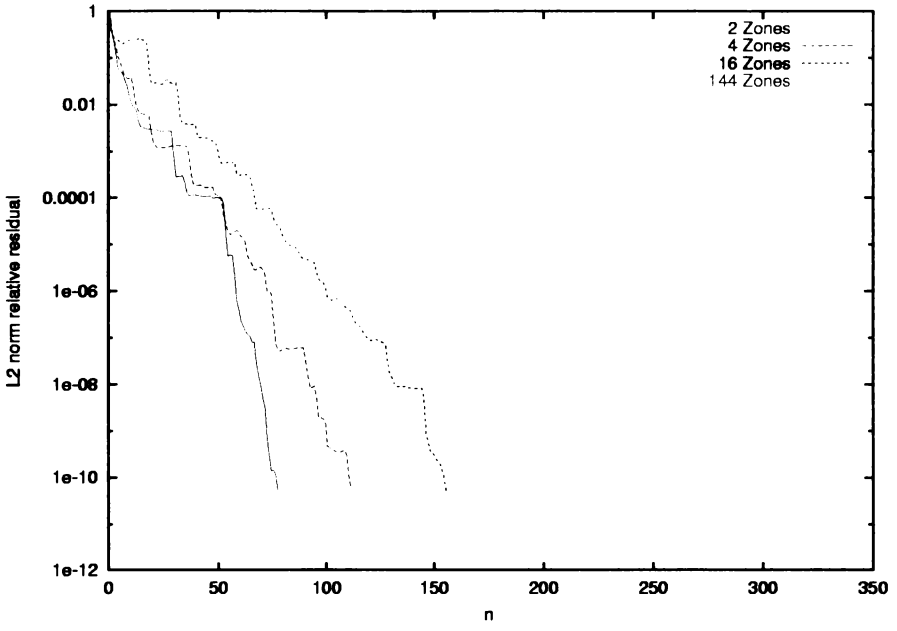


Figure 4.6: Comparison of the convergence rates vs. the number of zones, 3ppw.

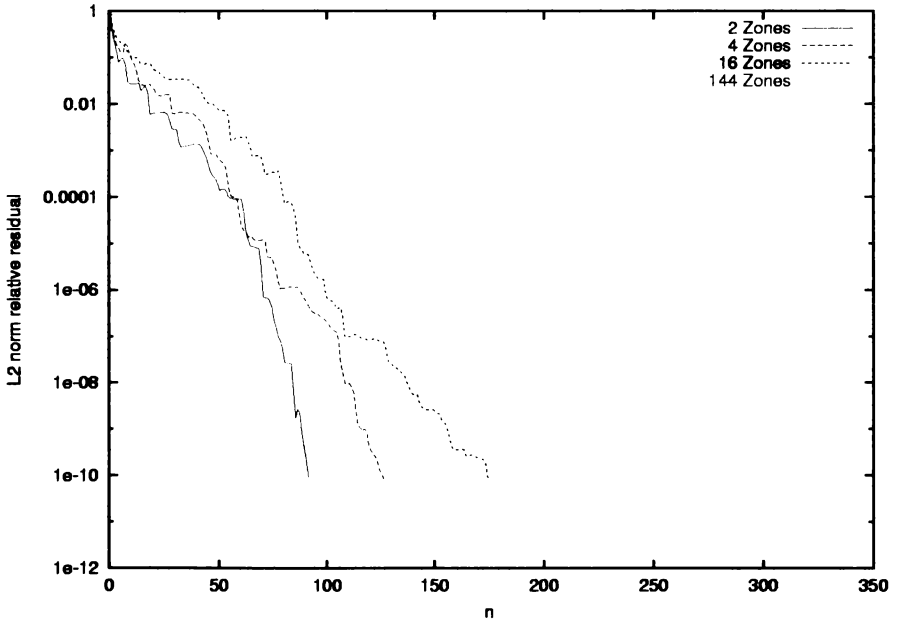


Figure 4.7: Comparison of the convergence rates vs. the number of zones, 15ppw.

RPACK. The computations were performed with fill amounts varying from 10 to 50 and drop tolerances varying from 0.0 to $1.0e-04$. In all cases $10,000$ iterations was insufficient to obtain convergence.

Chapter 5

Three-Dimensional Test Cases

5.1 Problem Description

In this chapter, Helmholtz's equation for the acoustic potential ϕ

$$\nabla^2\phi + k^2\phi = 0 \tag{5.1}$$

is solved on a three-dimensional computational domain $\mathcal{D}$. In this case, $\mathcal{D}$ is taken to be the unit cube $[0,1] \times [0,1] \times [0,1]$ in three-dimensional space. The test problem is chosen to be that of the sound field created by three point source radiators located at $\mathbf{r}_1 = (-0.25, -0.25, -0.25)$, $\mathbf{r}_2 = (0.0, -0.25, -0.25)$, and $\mathbf{r}_3 = (0.25, -0.25, -0.25)$.

The analytical solution [12] to equation 5.1 in this case is given by

$$\phi(\mathbf{r}) = \sum_{i=1}^{3} \frac{\exp(-\imath k\|\mathbf{r} - \mathbf{r}_i\|)}{4\pi\|\mathbf{r} - \mathbf{r}_i\|} \tag{5.2}$$

The problem is discretized using the GFD method outlined in chapter 2. The computational domain $\mathcal{D}$ is discretized using a variety of points np in each direction. Dirichlet boundary conditions are applied on the $x = 0$, $y = 0$, and $z = 0$ planes while the appropriate one-sided ARBCs developed in section 2.4 are used on all other boundaries. The linear system produced by the GFD process is solved using the transpose free QMR solver from **QMRPACK** [18]. The numerical tools outlined in the previous chapter are used to perform parametric studies using ADditive Schwarz preconditioning (ADS) in conjunction with differing number of sub-domains ND, points np, points per wavelength (PPW), and *yranges*. The sub-domains are again coupled using equations of the type

$$\boldsymbol{\phi}_I^T \mathbf{a}^B - \phi_I = \boldsymbol{\phi}_J^T \mathbf{a}^B - \phi_J, \tag{5.3}$$

where the vector $\mathbf{a}^B$ contains the weights from the appropriate boundary stencil. As outlined in section 2.4, recall that *yrange* is a parameter which controls the angular dependence of the wave vectors used in the computation of the boundary weights $\mathbf{a}^B$. The case $yrange = -1$ corresponds to $\theta_j \in [0, 90^o]$ while $yrange = -0.6$ corresponds to the case $\theta_j \in [0, 66^o]$.

5.2 Computational Cases

The first series of computations is designed to assess the sensitivity of the approach to changes in the number of sub-domains ND, the number of points per zone np, PPW, and yrange for a series of problems which all correspond to the case of a $32 \times 32 \times 32$

point grid in the case of a single domain . Here $ND = ND_x \times ND_y \times ND_z$ is the total number of sub-domains. The various cases are outlined in table 5.1.

Next, results are presented for a relatively large problem which corresponds to a $51 \times 51 \times 51$ point grid in the case of a single domain. These cases are cataloged in table 5.2.

Finally, the Singular Value Decomposition (SVD) routines from **lapack** [1] are used to help interpret some of the computational results presented in this chapter.

5.3 Numerical Results

The L_2 norms of convergence rates of the relative residuals $\frac{\|\mathbf{r}_n\|_2}{\|\mathbf{r}_0\|_2}$ for cases 1 thru 15 ($yrange = -1.0$) are summarized in figures 5.1 - 5.3. An examination of the figures reveals that the convergence rates in three dimensions are more sensitive to changes in resolution (PPW) than the corresponding one and two–dimensional cases. However, in all cases, the behavior of the convergence rates with respect to changes in ND is still quite good with the number of iterations required to reduce the residuals by a given amount being proportional to $ND^{\frac{1}{3}}$.

On the other hand, the $yrange = -0.6$ cases which are displayed in figures 5.4 - 5.6 show greater sensitivity to changes in PPW and ND than the $yrange = -1.0$ cases with the degradation in convergence rate being especially bad for the the 15 PPW cases. In addition, the $yrange = -0.6$ cases in general require more overall iterations to reduce the relative residual 9 orders of magnitude as compared with the $yrange = -1.0$ cases.

Table 5.1: Summary of three-dimensional parametric test cases.

Case	ND_x	ND_y	ND_z	np_x	np_y	np_z	PPW	*yrange*
1	2	2	2	17	17	17	3	-1
2	2	2	2	17	17	17	9	-1
3	2	2	2	17	17	17	15	-1
4	3	3	3	12	12	12	3	-1
5	3	3	3	12	12	12	9	-1
6	3	3	3	12	12	12	15	-1
7	5	5	5	8	8	8	3	-1
8	5	5	5	8	8	8	9	-1
9	5	5	5	8	8	8	15	-1
10	6	6	6	7	7	7	3	-1
11	6	6	6	7	7	7	9	-1
12	6	6	6	7	7	7	15	-1
13	10	10	10	5	5	5	3	-1
14	10	10	10	5	5	5	9	-1
15	10	10	10	5	5	5	15	-1
16	2	2	2	17	17	17	3	-0.6
17	2	2	2	17	17	17	9	-0.6
18	2	2	2	17	17	17	15	-0.6
19	3	3	3	12	12	12	3	-0.6
20	3	3	3	12	12	12	9	-0.6
21	3	3	3	12	12	12	15	-0.6
22	5	5	5	8	8	8	3	-0.6
23	5	5	5	8	8	8	9	-0.6
24	5	5	5	8	8	8	15	-0.6
25	6	6	6	7	7	7	3	-0.6
26	6	6	6	7	7	7	9	-0.6
27	6	6	6	7	7	7	15	-0.6
28	10	10	10	5	5	5	3	-0.6
29	10	10	10	5	5	5	9	-0.6
30	10	10	10	5	5	5	15	-0.6

Table 5.2: Summary of large scale cases.

Case	ND_x	ND_y	ND_z	np_x	np_y	np_z	ppw	preconditioner
37	1	1	1	51	51	51	3	none
38	1	1	1	51	51	51	15	none
38	7	7	7	9	9	9	3	ADS
40	7	7	7	9	9	9	15	ADS

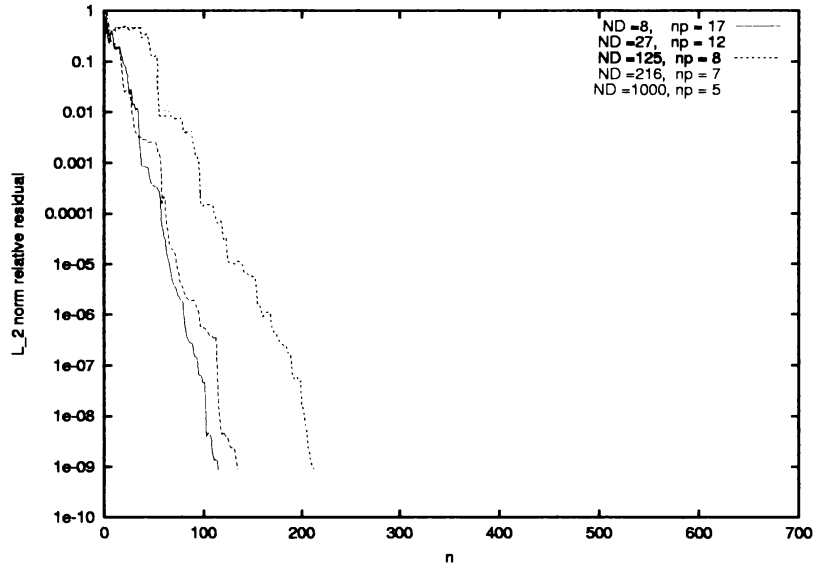


Figure 5.1: Comparison of the convergence rates vs. the number of zones ($yrange = -1.0, 3ppw$).

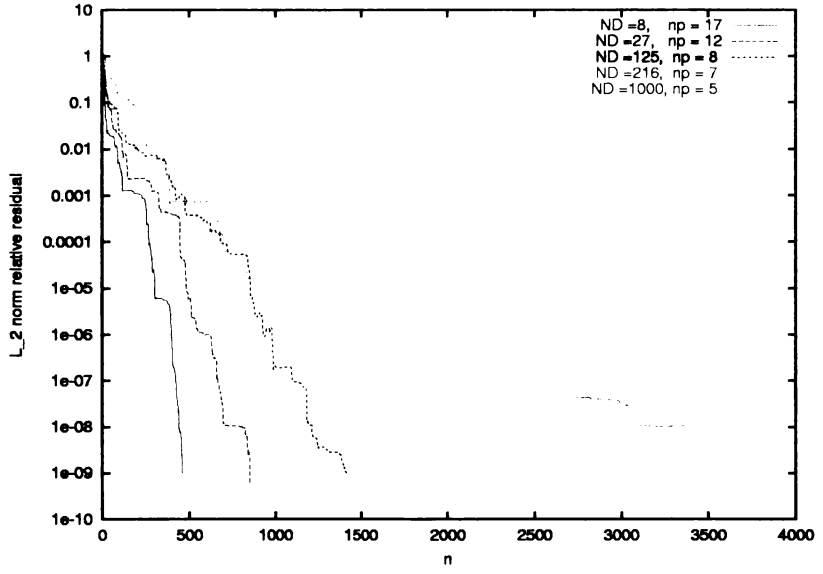


Figure 5.2: Comparison of the convergence rates vs. the number of zones, ($yrange = -1.0$, $9ppw$).

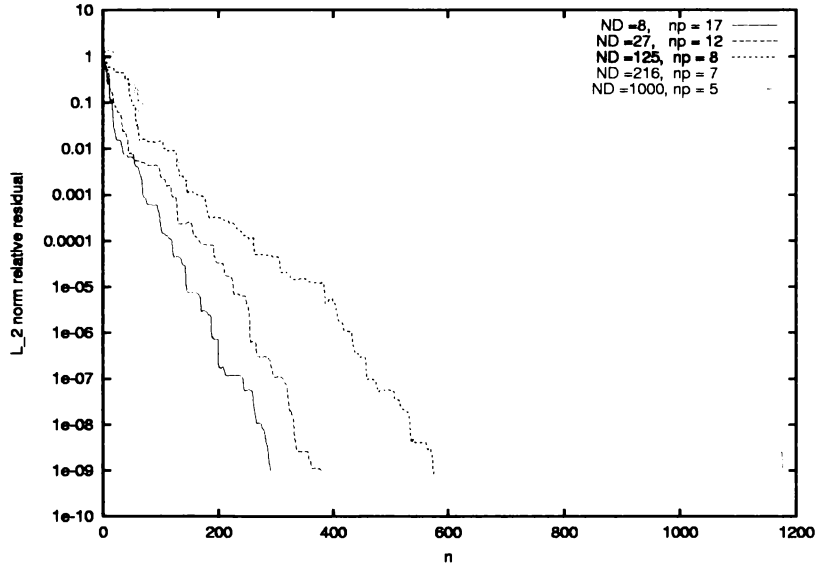


Figure 5.3: Comparison of the convergence rates vs. the number of zones, ($yrange = -1.0$, $15ppw$).

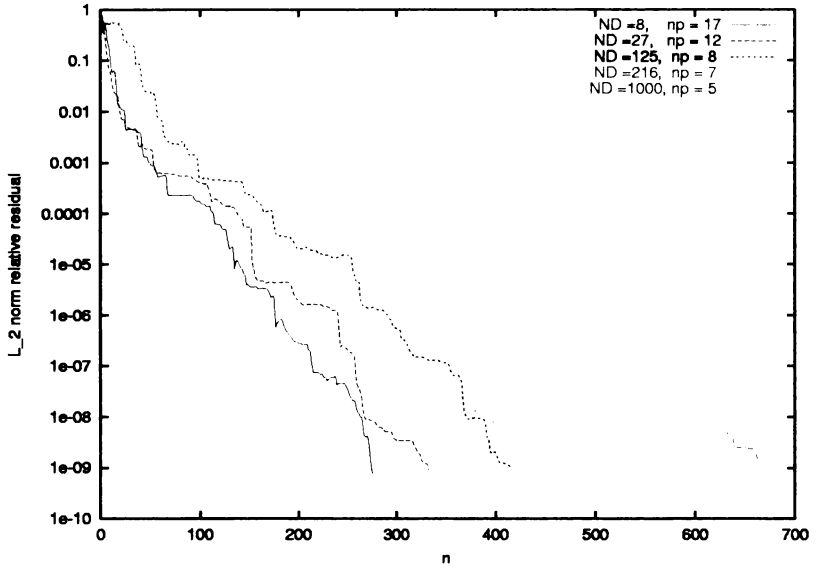


Figure 5.4: Comparison of the convergence rates vs. the number of zones ($yrange = -0.6$, $3ppw$).

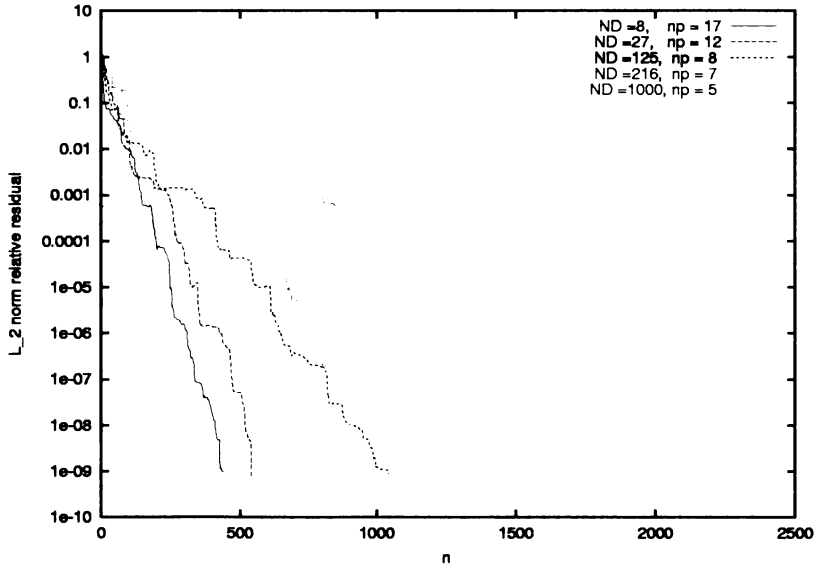


Figure 5.5: Comparison of the convergence rates vs. the number of zones, ($yrange = -0.6$,$9ppw$).

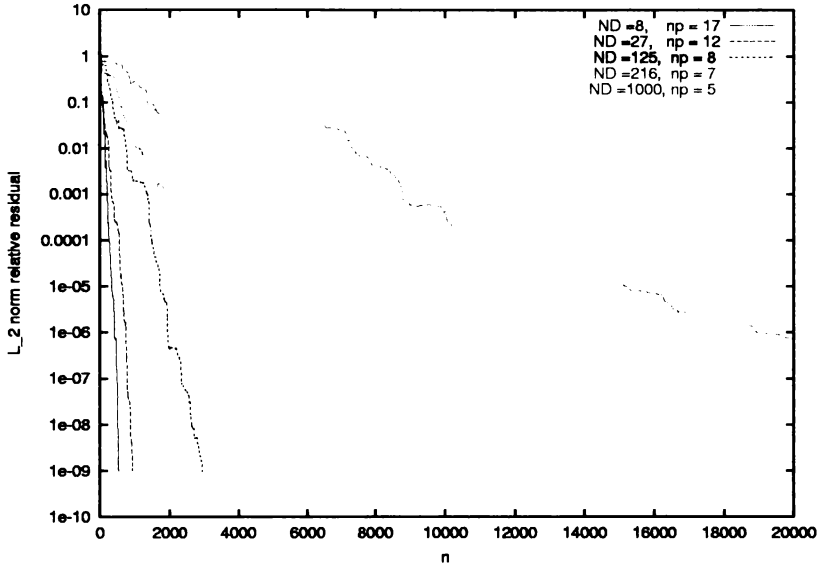


Figure 5.6: Comparison of the convergence rates vs. the number of zones, ($yrange = -0.6, 15ppw$).

Having first compared the convergence characteristics of the $yrange = -1.0$ and $yrange = -0.6$ test cases, the effect of $yrange$ on solution accuracy is now examined. A comparison of the numerically computed solutions with the exact solutions along the $x \in [0,1], y = 0.5, z = 1.0$ line are shown in figures 5.7 - 5.9. As can clearly be seen in these figures the solutions for the $yrange = -0.6$ cases are always more accurate than the $yrange = -1.0$ cases.

Given that setting $yrange = -1.0$ produces better convergence properties than $yrange = -0.6$ while setting $yrange = -0.6$ produces more accurate solutions it seems desirable to combine the best of both methods. This is possible by using $yrange = -1.0$ when computing the weights $\mathbf{a}^B$ which are used at artificial boundaries (equation 5.3) while $yrange = -0.6$ is used to compute the $\mathbf{a}^B$ vectors at the real physical boundaries of $\mathcal{D}$. In order to test whether this is a viable option or not, selected cases from the first set of computations are repeated using the hybrid approach. This set of test cases is summarized in table 5.3.

A comparison of the convergence rates using the hybrid yrange approach is com-

Table 5.3: Summary of three-dimensional hybrid test cases.

Case	ND_x	ND_y	ND_z	np_x	np_y	np_z	PPW	$yrange$
31	2	2	2	17	17	17	3	hybrid
32	2	2	2	17	17	17	9	hybrid
33	2	2	2	17	17	17	15	hybrid
34	10	10	10	5	5	5	3	hybrid
35	10	10	10	5	5	5	9	hybrid
36	10	10	10	5	5	5	15	hybrid

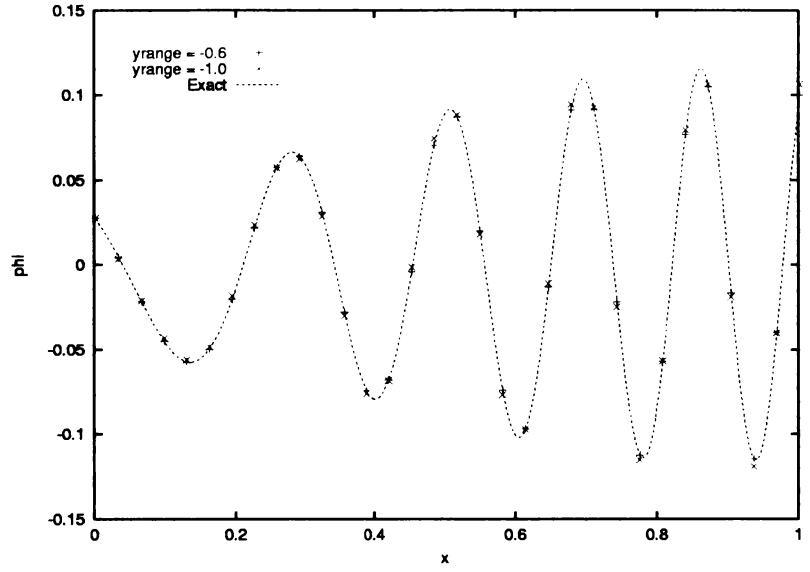


Figure 5.7: Comparison of computed and exact solutions (3*ppw*).

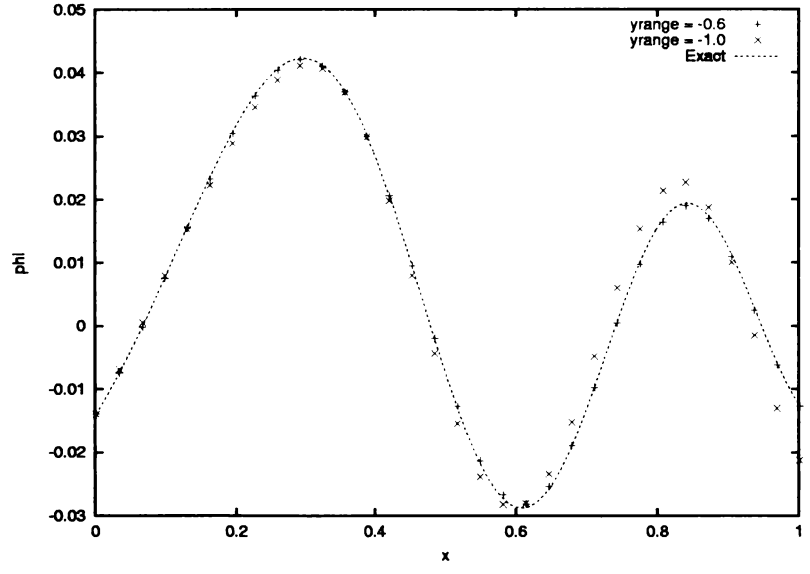


Figure 5.8: Comparison of computed and exact solutions (9*ppw*).

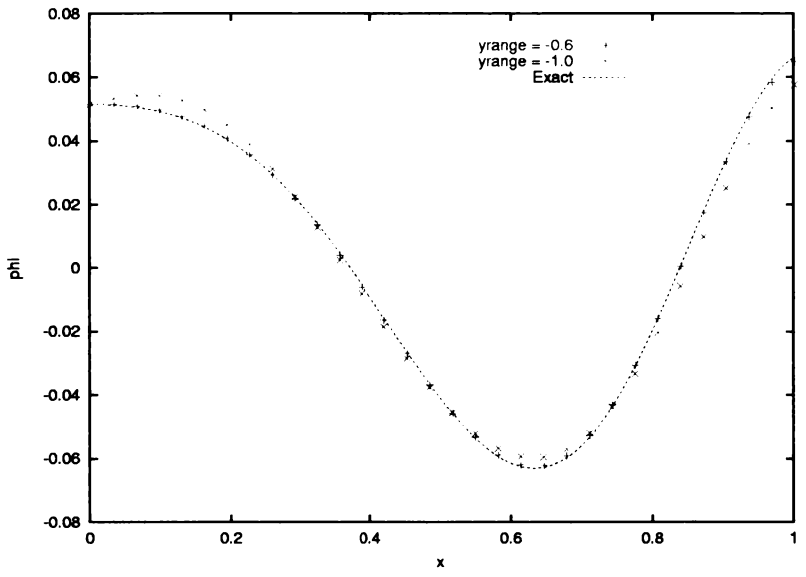


Figure 5.9: Comparison of computed and exact solutions (15*ppw*).

pared with the original results in figures 5.10 – 5.15. The convergence rates are largely unaffected as compared to the original $yrange = -1.0$ cases. The solutions (not shown here) agree with the $yrange = -0.6$ cases.

This section is concluded by applying the ADS preconditioner to a relatively large three-dimensional test case. The cases of this section are outlined in table 5.2. The convergence rate for the L2 norm of the relative residual for the 3*ppw* and 15*ppw* cases for the ADS method are compared with the unpreconditioned cases in figure 5.16. In both cases the ADS method converges. However the non-preconditioned case fails to converge for the 15 ppw case while converging fairly well for the 3 ppw case. A comparison of the numerical and exact solutions along the $x \in [0, 1], y = 0.5, z = 1.0$ line are shown in figure 5.17. The agreement is quite good in both cases.

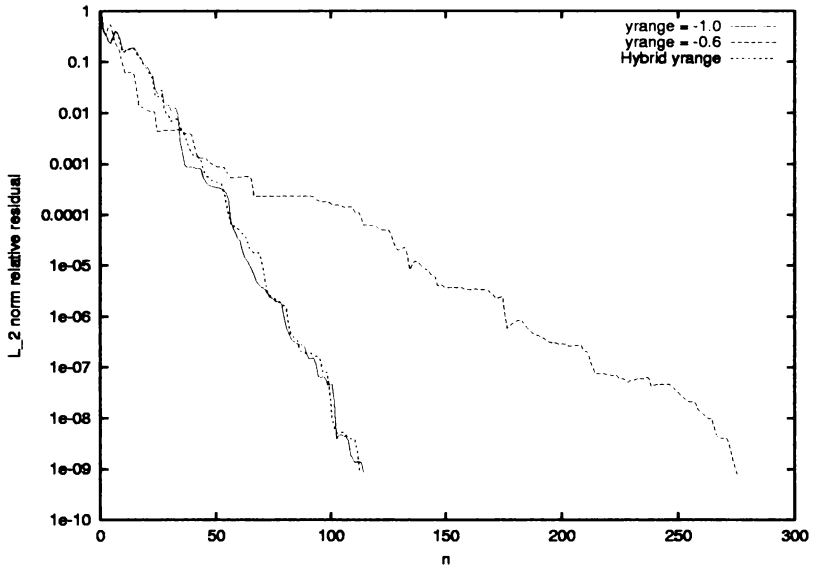


Figure 5.10: Comparison of hybrid yrange convergence rates ($3ppw$, $ND = 8$, $np = 17$).

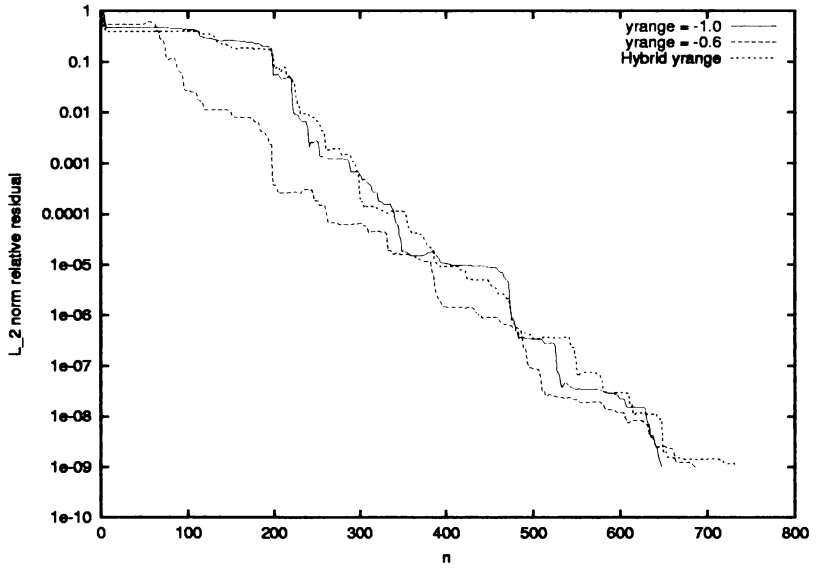


Figure 5.11: Comparison of hybrid yrange convergence rates ($3ppw$, $ND = 1000$, $np = 5$).

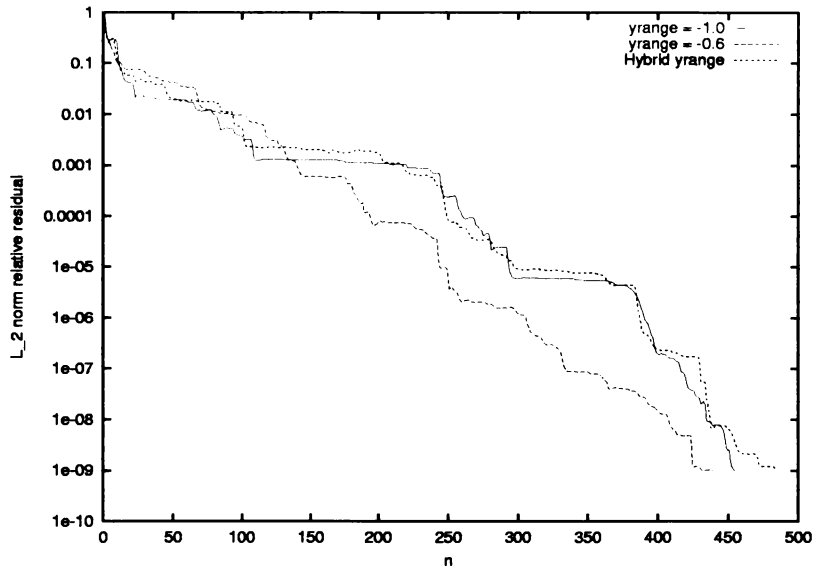


Figure 5.12: Comparison of hybrid yrange convergence rates ($9ppw$, $ND = 8$, $np = 17$).

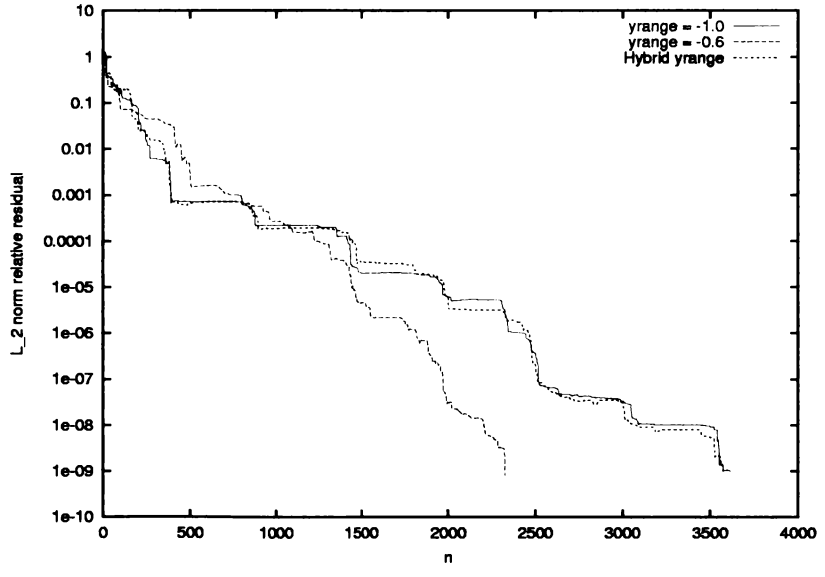


Figure 5.13: Comparison of hybrid yrange convergence rates ($9ppw$, $ND = 1000$, $np = 5$).

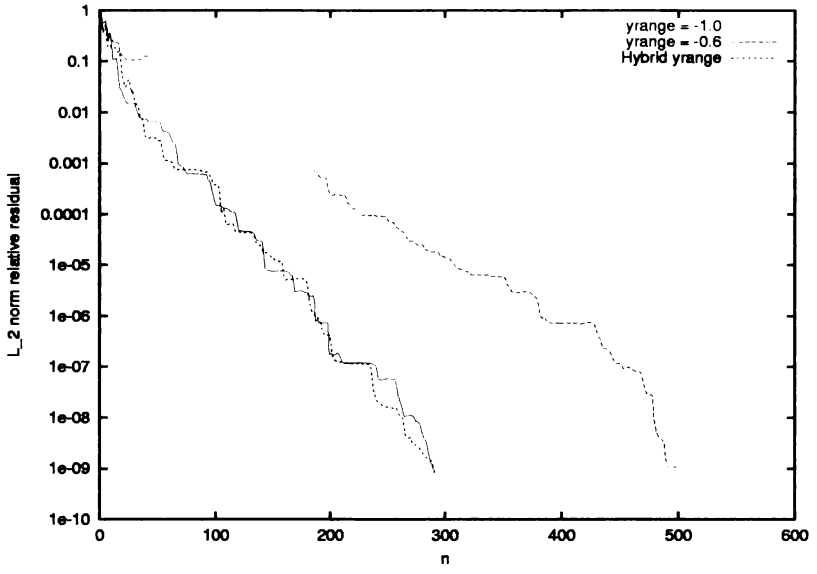


Figure 5.14: Comparison of hybrid yrange convergence rates (15*ppw*, $ND = 8$, $np = 17$).

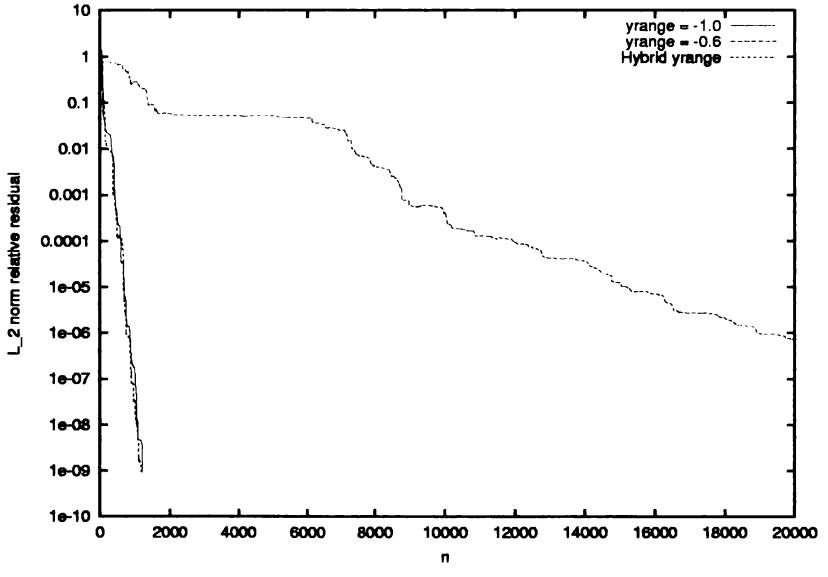
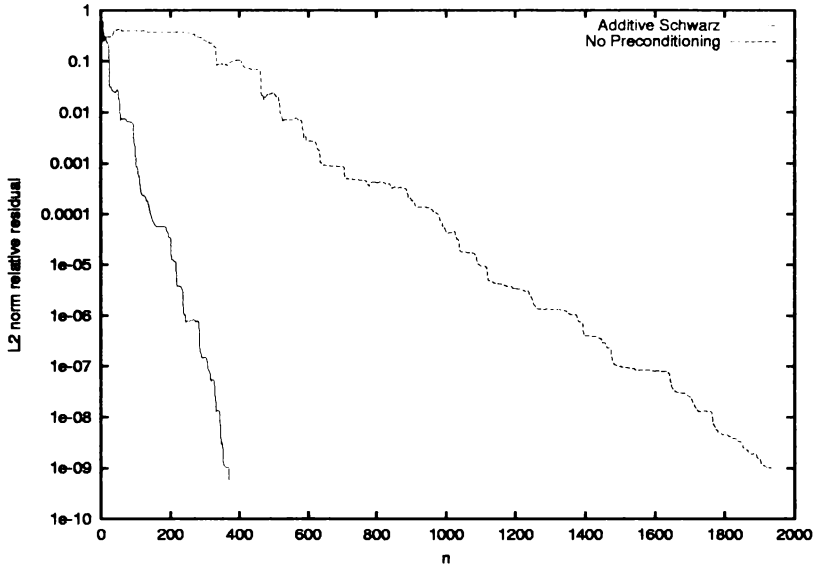
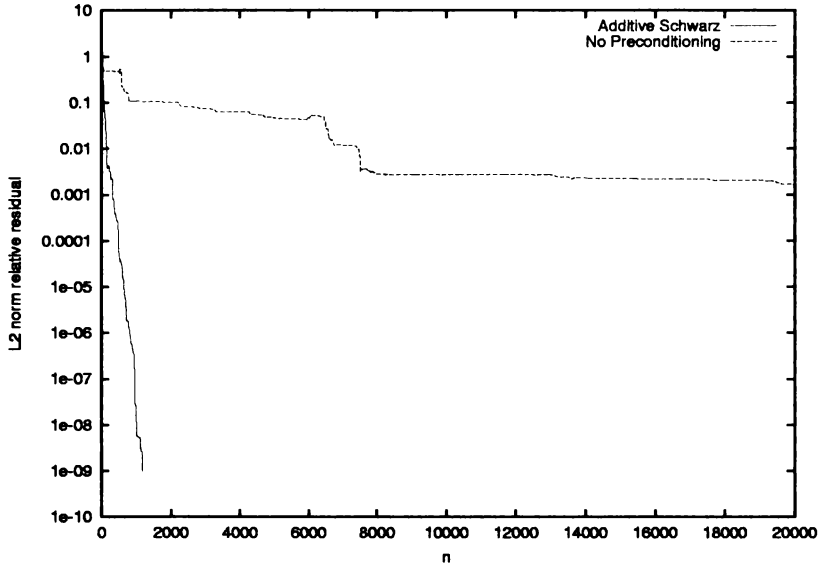


Figure 5.15: Comparison of hybrid yrange convergence rates (15*ppw*, $ND = 1000$, $np = 5$).

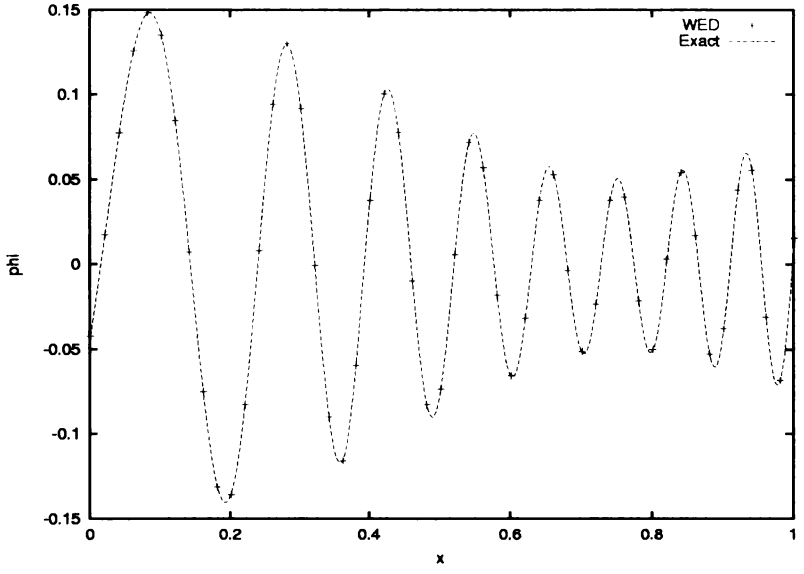


(a) $3ppw$

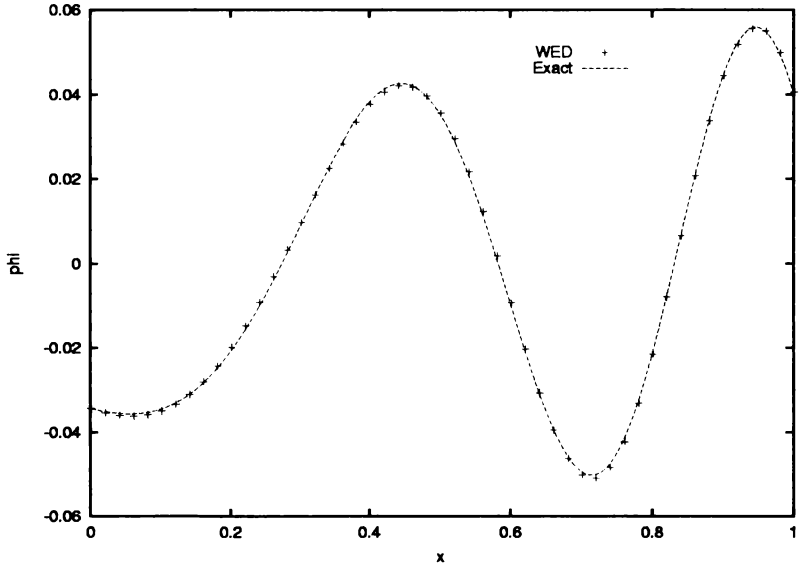


(b) $15ppw$

Figure 5.16: Convergence rates for large scale test cases.



(a) 3*ppw*



(b) 15*ppw*

Figure 5.17: Comparison of solutions for large scale test cases.

5.4 Numerical Condition Numbers

This chapter concludes by attempting to shed some light on the counter-intuitive fact that for all the methods considered in this chapter, convergence rates tended to degrade as the points per wavelength were increased, i. e. with increasing wave length. In order to do this the SVD routines from **lapack** [1] were used to compute the distribution of singular values for a single zone case which corresponds to $np = 11$. The setup is the same as for all the other test cases in the chapter. These results are summarized in table 5.4. Here $error_1 = |A\phi_e - b|_\infty$ is a measure of how well the exact solution ϕ_e satisfies the discretized system, while $error_2 = |\phi_e - \phi_n|_\infty$ is a measure of how well the numerical solution to the linear system $\phi_n = A^{-1}b$ is a match for the exact solution. An examination of this table clearly reveals how large the condition number becomes in the case of 15 ppw with the condition number for the $yrange = -0.6$ being substantially smaller than the $yrange = -1.0$ case. On the other hand, the condition numbers are quite comparable for the 3ppw case. In all cases the $error_1$ and $error_2$ measures are smaller for $yrange = -0.6$. These results are all consistent with and provide a partial explanation for the observed degradation in convergence rate at larger ppw.

Table 5.4: Summary of condition numbers.

$yrange$	ppw	$ConditionNumber$	$error_1$	$error_2$
-1	3	108.517619	0.0102602061	0.00945632914
-0.6	3	101.247241	0.00263828117	0.00204463795
-1	6	806.716649	0.000652438739	0.0246450029
-0.6	6	466.047815	0.00026499494	0.003625183
-1	9	17956.5071	0.000105331996	0.141022979
-0.6	9	2735.64993	6.4276586E-05	0.0175395103
-1	12	21936.1622	2.94477303E-05	0.0336047634
-0.6	12	5187.75731	2.50787446E-05	0.00786908079
-1.0	15	37889.0165	1.14461019E-05	0.0315706532
-0.6	15	24475.099	1.20936868E-05	0.0256515755

Chapter 6

Conclusions

In this work a domain decomposition based preconditioner of the additive Schwarz type was developed and tested on the linear systems which arise out of the application of the Green's Function/Wave Expansion Discretization (GFD/WED) method. In order to develop the additive Schwarz preconditioner, use was made of a class of one–sided Artificial Radiation Boundary Conditions developed during the course of this work. These ARBCs were computationally shown to be quite accurate for use on their own. The ARBC's were used to radiatively couple the various sub-domains which are naturally part of domain decomposition based methods in such a manner as to ensure that the system matrix, when restricted to the sub-domains, was non-singular. In addition, the inter-domain ARBC was constructed such that the solution to the global linear system was unaffected by the presence of the artificial boundaries.

The method was analyzed on a one-dimensional example where it was demonstrated that the use of the newly developed ARBC

(i) greatly improved the condition number of the preconditioned matrix,

(ii) removed the situations in which the preconditioned matrix became numerically singular, and

(iii) clustered the spectrum of the preconditioned matrix in the unit disk centered at $(1.0, 0.0)$ in the complex plane.

The additive Schwarz preconditioner was then combined with the QMR Krylov subspace method and was used to solve a two-dimensional test case in which known point source (i.e., monopole) radiators were used to to specify a class of test problems. The computational domain was taken to be a unit-sided square. Dirichlet boundary conditions were imposed on the two sides of the square closest to the point sources while ARBCs were used on the other two sides. The method was shown to yield highly accurate solutions with rapidly converging residuals. A study of the convergence rates and CPU times required using various numbers of zones indicated that the approach should be quite efficient when applied on parallel computer platforms.

The same computational setting was used to test the performance of the algorithm in three dimensions. A series of parametric computations were performed in which the sensitivity of the algorithm to changes in the number of sub-domains, points per sub-domain, and points per wavelength was assessed. The ADS based method still

performed quite well but displayed greater sensitivity to resolution (ppw) than it did in two-dimensions, with convergence rates degrading with increasing resolution. In addition, the effects of varying the angular ranges of the wave vectors $\vec{k} = k\hat{n}$ in the set of basis functions used in the WED expansion at boundaries was demonstrated. It was shown that restricting the wave vectors slightly produced more accurate solutions than the non-restricted cases while the convergence rate of the algorithm tended to decrease when the wave vectors were restricted. A hybrid method was proposed in which the weights derived using the restricted wave vectors were used on global boundaries of the computational domain while the weights derived using the full range of wave vectors were used on the artificial boundaries. The performance and accuracy of the hybrid method were demonstrated on two relatively large scale test cases. In addition, the condition numbers for a few selected test cases were computed. These computations showed that the condition numbers of the matrices which arise from the application of the WED method in three dimensions tended to increase rapidly with increasing resolution (ppw). This result provided some evidence of why the convergence rates displayed greater sensitivity in the three dimensional case. This phenomena is interesting in its own right and worthy of a separate investigation. However, such an effort is beyond the scope of this dissertation and is left for future research.

Bibliography

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. Mckenny, S. Ostrouchov, and D. Sorensen. *Lapack Users' Guide.* SIAM, Philadelphia PA, 1997. **http://www.netlib.org/lapack.**

[2] H. M. Atassi and R. F. Susan-Resiga. A domain decomposition method for time-harmonic aeroacoustic problems. *AIAA paper 99-0355*, May, 1998.

[3] I. Babuška and J. M. Melenk. The partition of unity finite element method. *Int. J. Numer. Meths.*, 40:727–758, 1997.

[4] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, Philadelphia PA, 1997. **http://www.netlib.org/templates.**

[5] A. DE La Bourdonnaye. Une méthode de discrétisation microlocale et son application à un problème de diffraction. *C. R. Acad. Sci. Paris*, 318:385–388, 1994.

[6] X.-C. Cai, M. Casarin, F. Elliott, and O. B. Widlund. Overlapping schwarz algorithms algorithms for solving Helmholtz's equation. In X.-C. Cai, C. Fahrat, and J. Mandel, editors, *Proc. 10th Int. Conf. on Domain Decomposition Methods*, Providence, RI, 1998. AMS.

[7] J. E. Caruthers, R. C. Engels, and G. K.Raviprakash. A wave expansion computational method for discrete frequency acoustics within inhomogeneous flows. *AIAA paper 96-1684*, May, 1998.

[8] J. E. Caruthers, J. E. French, and G. K.Raviprakash. Green's function discretization for numerical solution of the Helmholtz equation. *Journal of Sound and Vibration*, 187(4):553–568, 1995.

[9] J. E. Caruthers, J. S. Steinhoff, and R. C. Engels. An optimal finite difference representation for a class of linear pde's with application to the Helmholtz equation. *Journal of Computational Acoustics*, 7(4):245–252, 1999.

[10] J. W. Demmel, J. R. Gilbert, and X. S. Li. Superlu users' guide. 1999. **http://www.nersc.gov/ xiaoye/SuperLU/index.html.**

[11] B. Després. Décomposition de domaine et problème de Helmholtz. *C. R. Acad. Sci. Paris*, 311:313–316, 1990.

[12] A. P. Dowling and J. E. Ffowcs Williams. *Sound and Sources of Sound.* Halsted Press, New York, 1983.

[13] S. C. Eisenstat. Efficient implementation of a class of preconditioned conjugate gradient methods. *SIAM J. Sci. Statist. Comput.*, 2:1–4, 1981.

[14] V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.*, 21:352–362, 1984.

[15] R. Fletcher. Conjugate gradient methods for indefinite systems. In G. Watson, editor, *Numerical Analysis Dundee 1975*, Berlin, New York, 1976. Springer Verlag.

[16] R. Freund, G. H. Golub, and N. M. Nachtigal. Iterative solution of linear systems. *Acta Numerica*, pages 1–44, 1992.

[17] R. W. Freund and N. M. Nachtigal. QMR: A quasi-minimal residual method for non-hermitian linear systems. *Numer. Math.*, 60(4):315–339, 1991.

[18] R. W. Freund and N. M. Nachtigal. QMRPACK: a package of qmr algorithms. *ACM Trans. Math. Software*, 22:46–77, 1996. **http://www.netlib.org/linalg/qmrpack.tgz.**

[19] A. Greenbaum. *Iterative Methods for Solving Linear Systems.* SIAM, Philadelphia PA, 1997.

[20] M. R. Hestenes and E Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49:33–53, 1952.

[21] F. B. Hildebrand. *Finite-Difference Equations and Simulations.* Pretince Hall Inc., Englewood Cliffs, NJ, 1968.

[22] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand.*, 45:255–282, 1950.

[23] C. Lanczos. Solution of linear equations by minimized iterations. *J. Res. Natl. Bur. Stand.*, 49:409–436, 1952.

[24] P. Mayer and J. Mandel. The finite element ray method for the Helmholtz equation of scattering: first numerical experiments. Technical Report UCD/CCM report 111, Center for Computational Mathematics, Univerity of Colorado at Denver, Denver, CO, October 1997.

[25] L. C. McInnes, R. F. Susan-Resiga, D. E. Keyes, and H. M. Atassi. Parallel solution of Helmholtz problems using additive schwarz methods. In J. A. Desanto, editor, *Mathematical and Numerical aspects of wave propagation*, Philadelphia PA, 1998. SIAM.

[26] P. Morse and H. Feschbach. *Methods of Theoretical Physics, Vol. 1.* McGraw-Hill, New York, NY, 1953.

[27] A. D. Pierce. The Helmholtz-Kirchhoff integral relation as a framework for developing algorithms for sound propagation through inhomogeneous moving media. In D. Lee, A. Carmack, and R. Vichnevetsky, editors, *Computational Acoustics: Ocean-Acoustic Models and Supercomputing, Proceedings of the 2nd IMCAS Symposium on Computational Acoustics*, Amsterdam, 1990. Elsevier.

[28] Y. Saad. ILUT:A Dual Threshold Incomplete LU Factorization. Technical Report UMSI 92/38, University of Minnesota Supercomputer Institute, Minneapolis, MN, 1992.

[29] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, Boston MA, 1995.

[30] Y. Saad and M. H. Schultz. Conjugate gradient type algorithms for solving non-symmetric linear systems. *Math. Comp.*, 44:417–424, 1985.

[31] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving non-symmetric systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[32] H. A. Schwarz. *Gesammelte Mathematische Abhandlungen*, volume 2. Springer, Berlin, 1890.

[33] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain Decomposition.* Cambridge University Press, Cambridge, 1996.

[34] J. Spangler. Iterative Solution of the Discretized Helmholtz Equation. Master's thesis, The University of Knoxville, Tennessee, Knoxville, TN, 2000.

[35] G. Strang. *Linear Algebra and its Applications.* Wellesley–Cambridge Press, Wellesley MA, third edition, 1993.

[36] R. F. Susan-Resiga and H. M. Atassi. A domain decomposition method for the exterior Helmholtz equation. *J. Comp. Physics*, 147:338–401, 1998.

[37] P. Wesseling, editor. *High Performance Computing in Fluid Dynamics.* Kluwer Academic Publishers, Boston MA, 1996.

[38] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Oxford University Press, Oxford, 1965.

Appendix

Appendix A

Source Codes

A.1 QMR Driver Code

```
C**********************************************************************
C
C     Copyright (C) 1992  Roland W. Freund and Noel M. Nachtigal
C     All rights reserved.
C
C     This code is part of a copyrighted package.  For details, see the
C     file "cpyrit.doc" in the top-level directory.
C
C     ******************************************************************
C     ANY USE OF  THIS CODE CONSTITUTES ACCEPTANCE OF  THE TERMS OF THE
C                              COPYRIGHT NOTICE
C     ******************************************************************
C
C**********************************************************************
C
      SUBROUTINE CSRSYS2(a,ia,ja,rhs,
     &                   ib,ie,jb,je,zndx,
     &                   nrow,npp,nzone,
     &                   nsparse,TYPE)
C
```

```
C     This is an example driver code for the solution of linear systems
C     with coefficient  matrices stored  in the  compressed  sparse row
C     format. The original code was written by Noel Nachtigal. It was
C     modified by Robert Tramel in October 1999 to handle additive
C     Schwarz preconditioning.
C
C
C     The driver will ask the user for various run-time parameters, and
C     will expect to find the starting guess  in a file called "x.dat".
C     Note that the  file name is in  lower case (for systems where the
C     case is significant, such as Unix).  All the vectors  are assumed
C     to be of length NROW.  It is  the responsibility  of the  user to
C     ensure that there is enough data in the files.
C
C     NOTE: This code uses an input format that, while valid on the Sun
C     and Cray compilers, might not be accepted  by other compilers. In
C     particular, the construct
C        WRITE (6,'(A30,$)') 'This is a test'
C     is used to prevent  the output processor from  moving to the next
C     line after writing the text.  If this construct  is not supported
C     by the compiler, remove the dollar sign ($).
C     Also, the user input in this code is not bullet-proof. It assumes
C     that the user will provide valid inputs, i.e., numbers where they
C     are expected, etc. Providing invalid input type will likely crash
C     the program.
C
C     Noel M. Nachtigal
C     March 9, 1992
C
C*********************************************************************
C
      EXTERNAL ZUIL1I, ZUIL1T, ZUIL2I, ZUIL2T, ZUILST
      EXTERNAL ZUS2SI, ZUS2ST, ZUSS1I, ZUSS1T, ZUSS2I, ZUSS2T, ZUSSST
      EXTERNAL ZAXPBY, ZUCPL, ZUCPX, ZUQBG, ZUQMR, ZUQMX, ZUTFX
      EXTERNAL ZSPAXB, ZSPATX, ZSPRAS, ZSPRRW
C
C*********************************************************************
C
C     Parameters for the data format and the preconditioners.
C
      INTEGER NZLMAX, NZUMAX
C
```

```
C     Parameters for some of the solvers.
C
      INTEGER M, MAXN, MAXPQ, MAXVW
      PARAMETER (MAXN = 10000, MAXPQ = 3, MAXVW = 3)
      PARAMETER (M = MAXVW+MAXPQ+2)
C
C     Miscellaneous parameters.
C
      DOUBLE COMPLEX ZONE, ZZERO
      PARAMETER (ZONE = (1.0D0,0.0D0),ZZERO = (0.0D0,0.0D0))
c
c    input data (matrix in csr format)
c
      INTEGER IB(nzone)  , IE(nzone)
      INTEGER JB(nzone)  , JE(nzone)
      INTEGER ZNDX(NZONE)
      INTEGER IA(NROW+1) , JA(NSPARSE)
      DOUBLE COMPLEX A(NSPARSE)
      DOUBLE COMPLEX RHS(NROW)
C
C     Variables for the data format and the preconditioners.
C
      LOGICAL, dimension(:), allocatable :: JT
      INTEGER, dimension(:), allocatable :: IDA, IL, IU
      INTEGER, dimension(:), allocatable :: JI,  JL, JU
      DOUBLE COMPLEX,   dimension(:), allocatable :: AINV, DR, DUINV
      DOUBLE COMPLEX,   dimension(:), allocatable :: L, U
      DOUBLE PRECISION, dimension(:), allocatable :: DN, DS
      CHARACTER TYPE*3
      INTEGER NCOL, NROW, PRECON
      INTEGER FACTOR, SOLVE, TRANSPOSE
      DOUBLE PRECISION OMEGA
C
C     Variables common to all solvers.
C     All solvers use: NROW, NLEN, NLIM, VECS, TOL, and INFO.
C
      INTEGER INFO(4), NLEN, NLIM
      DOUBLE COMPLEX, dimension(:,:), allocatable :: VECS
      DOUBLE COMPLEX, dimension(:),   allocatable :: XZR
      DOUBLE PRECISION TOL
C
C     Variables specific to only some of the solvers.
```

```
C
      INTEGER IDX(6 ,MAXN), IWK(M,13 ), MVEC
      DOUBLE COMPLEX ZWK(M,8*M+7)
      DOUBLE PRECISION DWK(M,MAXN+4 ), NORMS(2)
C
C     Variables used by reverse communication.
C
      INTEGER CB, CX, IERR, REVCOM
C
C     Local driver variables.
C
      INTEGER ALG, I, MAXXPQ, MAXXVW, OLDPRE, PRE
      LOGICAL NEWMAT
      SAVE    NEWMAT
      CHARACTER ANS*1, FNAME*72
      DOUBLE COMPLEX, dimension(:), allocatable :: XTMP
      DATA NEWMAT /.TRUE./
c
c     allocate memory for the main work arrays
c     preconditioner specific memory is allocated
c     below on a case by case method
c
      ncol_vecs = 5*(MAXPQ+MAXVW)+ 4
      allocate( vecs(nrow,ncol_vecs) )
      allocate( xzr(nrow) )
      allocate( xtmp(nrow) )
      ncol = nrow
      write(6,*) 'nrow,ncol = ',nrow,ncol
      write(6,*) 'nsparse = ',nsparse
      nlen = nrow
      do i=1,nrow
       vecs(i,2) = rhs(i)
       xzr(i)    = rand(i)
      enddo
C
      NLIM = MIN(MAXN-2,NLIM)
      IF (NLEN.EQ.0) GO TO 100

      PRE    = 0
      NEWMAT = .TRUE.

C
```

```
C     Select an algorithm.
C
      WRITE (6,'(A38)')  'Choices of algorithm         : 1 = CPL'
      WRITE (6,'(A38)')  '                               2 = CPX'
      WRITE (6,'(A38)')  '                               3 = QBG'
      WRITE (6,'(A38)')  '                               4 = QMR'
      WRITE (6,'(A38)')  '                               5 = QMX'
      WRITE (6,'(A38)')  '                               6 = TFX'
      WRITE (6,'(A31,$)')  'Select an algorithm          : '
      READ (5,*) ALG
      IF ((ALG.LT.1).OR.(ALG.GT.6)) GO TO 100
C
C     Get the convergence tolerance.
C
      WRITE (6,'(A31,$)') 'Enter convergence tolerance  : '
      READ (5,*) TOL
C
C     Get the maximum number of iteration steps.
C
      WRITE (6,'(A31,$)') 'Maximum number of steps NLIM : '
      READ (5,*) NLIM
      IF (NLIM.GT.MAXN-2) THEN
         NLIM = MAXN-2
         WRITE (6,'(A31,I10)') 'NLIM too large, adjusted to  : ', NLIM
      END IF
C
C     Solver-specific initialization.
C
      IF (ALG.EQ. 1 ) THEN
         WRITE (6,'(A31,$)') 'Enter estimated norm for P&Q : '
         READ (5,*) NORMS(1)
         WRITE (6,'(A31,$)') 'Enter estimated norm for V&W : '
         READ (5,*) NORMS(2)
C
C     Set the maximum block and storage information.
C
         MAXXPQ = MAXPQ
         MAXXVW = MAXVW
         MVEC   = MAXPQ + MAXVW
      ELSE IF (ALG.EQ. 4 ) THEN
         WRITE (6,'(A31,$)') 'Enter estimated matrix norm  : '
         READ (5,*) NORMS(1)
```

```
C
C     Set the maximum block and storage information.
C
         MAXXVW = MAXVW
      END IF
      IF (NLEN.EQ.0) GO TO 100
C
C     Select a preconditioner.
C
      OLDPRE = PRE
      PRE    = 0
      WRITE (6,'(A42)') 'Choices of preconditioner    : 0 = no prec'
      WRITE (6,'(A44)') '                               1 = Left ILUT'
      WRITE (6,'(A45)') '                               2 = Right ILUT'
      WRITE (6,'(A49)')
     $       '                                  3 = Two-sided ILUT'
      WRITE (6,'(A44)') '                               4 = Left SSOR'
      WRITE (6,'(A45)') '                               5 = Right SSOR'
      WRITE (6,'(A49)')
     $       '                                  6 = Two-sided SSOR'
      WRITE (6,'(A49)')
     $       '                                  7 = Addit. Schwarz'
      WRITE (6,'(A31,$)') 'Select a preconditioner      : '
      READ (5,*) PRECON
      IF ((PRECON.EQ.1).OR.(PRECON.EQ.2).OR.(PRECON.EQ.3)) THEN
         PRE = 1
      ELSE IF ((PRECON.EQ.4).OR.(PRECON.EQ.5).OR.(PRECON.EQ.6)) THEN
         PRE = 2
      ELSE IF ((PRECON.EQ.7)) THEN
         PRE = 3
      ELSE
         PRECON = 0
         OLDPRE = PRE + 1
      END IF
C
C     Initialize the preconditioner.
C
      IF ((.NOT.NEWMAT).AND.(PRE.EQ.OLDPRE)) THEN
         WRITE (6,'(A31,$)') 'Reuse the preconditioner (Y/N) '
         READ (5,'(A1)') ANS
         NEWMAT = (ANS.EQ.'N').OR.(ANS.EQ.'n')
      END IF
```

```
      IF (NEWMAT.OR.(PRE.NE.OLDPRE)) THEN
         IF (PRE.EQ.1) THEN
            nzlmax=30*nrow
            nzumax=30*nrow
            allocate( l(nzlmax) )
            allocate( il(nrow+1) )
            allocate( jl(nzlmax) )
            allocate( u(nzumax) )
            allocate( iu(nrow+1) )
            allocate( ju(nzumax) )
            allocate( jt(nrow) )
            allocate( ji(nrow) )
            allocate( dn(nrow) )
            allocate( dr(nrow) )
            allocate( ds(nrow) )
            allocate( duinv(nrow) )
            CALL ZUILST (NROW,A,IA,JA,L,IL,JL,U,IU,JU,JT,JI,DN,DR,DS,
     $                   DUINV,NZLMAX,NZUMAX,IERR)
         ELSE IF (PRE.EQ.2) THEN
            allocate( ainv(nrow) )
            allocate( ida(nrow) )
            CALL ZUSSST (NLEN,A,IA,JA,AINV,IDA,OMEGA,PRECON)
         ELSE IF (PRE.EQ.3) THEN
            write(6,*) 'setting up the Schwarz preconditioner'
CRWT        Lets set "POINTers" into the precon arrays
            call point(ib,ie,jb,je,nrow,npp,nzone)
            factor    = 1
            solve     = 0
            transpose = 1
            call sp_solve( nrow, nsparse, a, ja, ia, ib, ie, jb, je,
     $                     zndx,transpose,
     $                     XTMP, XTMP, nzone, factor, solve, info )
         END IF
      END IF
C
C     Compute the modified right hand side.
C
      CALL ZSPAXB (NLEN,A,IA,JA,TYPE,XZR,XTMP)
      CALL ZAXPBY (NLEN,VECS( 1 ,2) ,ZONE,VECS( 1 ,2) ,-ZONE,XTMP)
      IF (PRE.EQ.1) THEN
         CALL ZUIL1I (NROW,L,IL,JL,U,IU,JU,DUINV,PRECON,VECS( 1 ,2) )
      ELSE IF (PRE.EQ.2) THEN
```

```
         CALL ZUSS1I (NLEN,A,IA,JA,AINV,IDA,OMEGA,PRECON,VECS( 1 ,2) )
       ELSE IF (PRE.EQ.3) THEN
          factor    = 0
          solve     = 1
          transpose = 1
          call sp_solve( nrow, nsparse, a, ja, ia, ib, ie, jb, je,
      $                  zndx,transpose,
      $                  vecs(1,2), XTMP, nzone, factor, solve, info )
          CALL ZAXPBY (NLEN, VECS(1,2),ZONE,XTMP,ZZERO,XTMP)
       END IF
C
C      Set up call to linear systems solver.
C      Compute true residual norms, generate second starting vector.
C
       INFO(2) = 0
       INFO(1) = 011106
C
C      Open the output files.
C
       OPEN (11,FILE = 'res.out')
C
C      Call the solver.
C
 80    IF (ALG.EQ. 1 ) THEN
          CALL ZUCPL (NROW,NLEN,NLIM,MAXXPQ,MAXXVW,M,MVEC,NORMS,ZWK,DWK,
      $               IDX,IWK,VECS,TOL,INFO)
       ELSE IF (ALG.EQ. 2 ) THEN
          CALL ZUCPX (NROW,NLEN,NLIM,VECS,TOL,INFO)
       ELSE IF (ALG.EQ. 3 ) THEN
          CALL ZUQBG (NROW,NLEN,NLIM,VECS,TOL,INFO)
       ELSE IF (ALG.EQ. 4 ) THEN
          CALL ZUQMR (NROW,NLEN,NLIM,MAXXVW,M,NORMS(1),ZWK,DWK,IDX,IWK,
      $               VECS,TOL,INFO)
       ELSE IF (ALG.EQ. 5 ) THEN
          CALL ZUQMX (NROW,NLEN,NLIM,VECS,TOL,INFO)
       ELSE IF (ALG.EQ. 6 ) THEN
          CALL ZUTFX (NROW,NLEN,NLIM,VECS,TOL,INFO)
       END IF
C
C      Perform matrix-vector multiplications when needed.
C
       IERR   = INFO(1)
```

```
      REVCOM = INFO(2)
      CX     = INFO(3)
      CB     = INFO(4)
C
C     Multiply VECS(1,CX) with the preconditioned matrix.
C
      IF (REVCOM.EQ.1) THEN
         IF (PRE.EQ.0) THEN
            CALL ZSPAXB (NROW,A,IA,JA,TYPE,VECS(1,CX),VECS(1,CB))
         ELSE IF (PRE.EQ.1) THEN
            CALL ZAXPBY (NLEN,XTMP,ZONE,VECS(1,CX),ZZERO,XTMP)
            CALL ZUIL2I (NROW,L,IL,JL,U,IU,JU,DUINV,PRECON,XTMP)
            CALL ZSPAXB (NROW,A,IA,JA,TYPE,XTMP,VECS(1,CB))
            CALL ZUIL1I (NROW,L,IL,JL,U,IU,JU,DUINV,PRECON,VECS(1,CB))
         ELSE IF ((PRECON.EQ.4).OR.(PRECON.EQ.5)) THEN
            CALL ZAXPBY (NLEN,XTMP,ZONE,VECS(1,CX),ZZERO,XTMP)
            CALL ZUSS2I (NLEN,A,IA,JA,AINV,IDA,OMEGA,PRECON,XTMP)
            CALL ZSPAXB (NROW,A,IA,JA,TYPE,XTMP,VECS(1,CB))
            CALL ZUSS1I (NLEN,A,IA,JA,AINV,IDA,OMEGA,PRECON,VECS(1,CB))
         ELSE IF (PRECON.EQ.6) THEN
            CALL ZAXPBY (NLEN,VECS(1,CB),ZONE,VECS(1,CX),ZZERO,XTMP)
            CALL ZUS2SI (NROW,A,IA,JA,AINV,IDA,OMEGA,PRECON,
     $                   VECS(1,CB),XTMP)
         ELSE IF (PRECON.EQ.7) THEN
            CALL ZSPAXB (NROW,A,IA,JA,TYPE,VECS(1,CX),XTMP)
            factor    = 0
            solve     = 1
            transpose = 1
            call sp_solve( nrow, nsparse, a, ja, ia, ib, ie, jb, je,
     $                     zndx,transpose,
     $                     XTMP, vecs(1,CB), nzone, factor, solve, info )
         END IF
         GO TO 80
C
C     Multiply VECS(1,CX) with the preconditioned transpose.
C
      ELSE IF (REVCOM.EQ.2) THEN
         IF (PRE.EQ.0) THEN
            CALL ZSPATX (NLEN,A,IA,JA,TYPE,VECS(1,CX),VECS(1,CB))
         ELSE IF (PRE.EQ.1) THEN
            CALL ZAXPBY (NLEN,XTMP,ZONE,VECS(1,CX),ZZERO,XTMP)
            CALL ZUIL1T (NROW,L,IL,JL,U,IU,JU,DUINV,PRECON,XTMP)
```

```
            CALL ZSPATX (NLEN,A,IA,JA,TYPE,XTMP,VECS(1,CB))
            CALL ZUIL2T (NROW,L,IL,JL,U,IU,JU,DUINV,PRECON,VECS(1,CB))
         ELSE IF ((PRECON.EQ.4).OR.(PRECON.EQ.5)) THEN
            CALL ZAXPBY (NLEN,XTMP,ZONE,VECS(1,CX),ZZERO,XTMP)
            CALL ZUSS1T (NLEN,A,IA,JA,AINV,IDA,OMEGA,PRECON,XTMP)
            CALL ZSPATX (NLEN,A,IA,JA,TYPE,XTMP,VECS(1,CB))
            CALL ZUSS2T (NLEN,A,IA,JA,AINV,IDA,OMEGA,PRECON,VECS(1,CB))
         ELSE IF (PRECON.EQ.6) THEN
            CALL ZAXPBY (NLEN,VECS(1,CB),ZONE,VECS(1,CX),ZZERO,XTMP)
            CALL ZUS2ST (NROW,A,IA,JA,AINV,IDA,OMEGA,PRECON,
     $                   VECS(1,CB),XTMP)
         ELSE IF (PRECON.EQ.7) THEN
            factor    = 0
            solve     = 1
            transpose = 0
            call sp_solve( nrow, nsparse, a, ja, ia, ib, ie, jb, je,
     $                     zndx,transpose,
     $                     VECS(1,CX),XTMP, nzone, factor, solve, info )
            CALL ZSPATX (NLEN,A,IA,JA,TYPE,XTMP,VECS(1,CB))
         END IF
         GO TO 80
      END IF
C
C     Close the output files.
C
      CLOSE (11)
C
C     Check why the solver stopped (this could be more compact).
C
      IF (IERR.EQ. 0 ) THEN
         WRITE (6,'(A32)') 'The residual norm has converged.'
         GO TO 90
      ELSE IF (IERR.EQ. 1 ) THEN
         WRITE (6,'(A35)') 'Invalid reverse communication call.'
         GO TO 90
      ELSE IF (IERR.EQ. 2 ) THEN
         WRITE (6,'(A27)') 'Invalid inputs encountered.'
         GO TO 90
      ELSE IF (IERR.EQ. 4 ) THEN
         WRITE (6,'(A31)') 'The algorithm did not converge.'
         GO TO 90
      END IF
```

```
 IF (ALG.EQ. 1 ) THEN
    IF (IERR.LT.0) THEN
       WRITE (6,'(A40,I5)')
$'Error encountered in the ZSVDC routine: ',
$            -IERR
       GO TO 90
    ELSE IF (IERR.EQ. 8 ) THEN
       WRITE (6,'(A35)') 'The last block could not be closed.'
       GO TO 90
    ELSE IF (IERR.EQ. 16 ) THEN
       WRITE (6,'(A39)') 'An A-invariant subspace has been found.'
       GO TO 90
    ELSE IF (IERR.EQ. 32 ) THEN
       WRITE (6,'(A41)')
$'An A^T-invariant subspace has been found.'
       GO TO 90
    ELSE IF (IERR.EQ. 48 ) THEN
       WRITE (6,'(A41)')
$'Both invariant subspaces have been found.'
       GO TO 90
    ELSE IF (IERR.EQ. 64 ) THEN
       WRITE (6,'(A30)') 'Insufficient memory allocated.'
       GO TO 90
    ELSE IF (IERR.EQ. 128 ) THEN
       WRITE (6,'(A32)') 'Cannot convert regular to inner.'
       GO TO 90
    END IF
 ELSE IF (ALG.EQ. 2 ) THEN
    IF (IERR.EQ. 8 ) THEN
       WRITE (6,'(A25)') 'The algorithm broke down.'
       GO TO 90
    ELSE IF (IERR.EQ. 16 ) THEN
       WRITE (6,'(A39)') 'An A-invariant subspace has been found.'
       GO TO 90
    ELSE IF (IERR.EQ. 32 ) THEN
       WRITE (6,'(A41)')
$'An A^T-invariant subspace has been found.'
       GO TO 90
    ELSE IF (IERR.EQ. 48 ) THEN
       WRITE (6,'(A41)')
$'Both invariant subspaces have been found.'
       GO TO 90
```

```
    END IF
 ELSE IF (ALG.EQ. 3 ) THEN
    IF (IERR.EQ. 8 ) THEN
       WRITE (6,'(A25)') 'The algorithm broke down.'
       GO TO 90
    END IF
 ELSE IF (ALG.EQ. 4 ) THEN
    IF (IERR.LT.0) THEN
       WRITE (6,'(A40,I5)')
$'Error encountered in the ZSVDC routine: ',
$            -IERR
       GO TO 90
    ELSE IF (IERR.EQ. 8 ) THEN
       WRITE (6,'(A35)') 'The last block could not be closed.'
       GO TO 90
    ELSE IF (IERR.EQ. 16 ) THEN
       WRITE (6,'(A39)') 'An A-invariant subspace has been found.'
       GO TO 90
    ELSE IF (IERR.EQ. 32 ) THEN
       WRITE (6,'(A41)')
$'An A^T-invariant subspace has been found.'
       GO TO 90
    ELSE IF (IERR.EQ. 48 ) THEN
       WRITE (6,'(A41)')
$'Both invariant subspaces have been found.'
       GO TO 90
    END IF
 ELSE IF (ALG.EQ. 5 ) THEN
    IF (IERR.EQ. 8 ) THEN
       WRITE (6,'(A25)') 'The algorithm broke down.'
       GO TO 90
    ELSE IF (IERR.EQ. 16 ) THEN
       WRITE (6,'(A39)') 'An A-invariant subspace has been found.'
       GO TO 90
    ELSE IF (IERR.EQ. 32 ) THEN
       WRITE (6,'(A41)')
$'An A^T-invariant subspace has been found.'
       GO TO 90
    ELSE IF (IERR.EQ. 48 ) THEN
       WRITE (6,'(A41)')
$'Both invariant subspaces have been found.'
       GO TO 90
```

```
         END IF
      ELSE IF (ALG.EQ. 6 ) THEN
         IF (IERR.EQ. 8 ) THEN
            WRITE (6,'(A25)') 'The algorithm broke down.'
            GO TO 90
         END IF
      END IF
      WRITE (6,'(A19,I5)') 'Unknown INFO code: ', IERR
C
C     Compute the unpreconditioned solution.
C
 90   IF (PRE.EQ.1) THEN
         CALL ZUIL2I (NROW,L,IL,JL,U,IU,JU,DUINV,PRECON,VECS(1,1) )
      ELSE IF (PRE.EQ.2) THEN
         CALL ZUSS2I (NLEN,A,IA,JA,AINV,IDA,OMEGA,PRECON,VECS(1,1) )
      END IF
      CALL ZAXPBY (NLEN,XZR,ZONE,XZR,ZONE,VECS(1,1) )
      DO I=1,NROW
        RHS(I) = XZR(I)
      ENDDO
 100  CONTINUE
      IF(PRE .EQ. 1) THEN
         deallocate( l )
         deallocate( il )
         deallocate( jl )
         deallocate( u )
         deallocate( iu )
         deallocate( ju )
         deallocate( jt )
         deallocate( ji )
         deallocate( dn )
         deallocate( dr )
         deallocate( ds )
         deallocate( duinv )
      ELSEIF(PRE .EQ. 2) THEN
         deallocate( ainv )
         deallocate( ida )
      ENDIF
      deallocate( vecs )
      deallocate( xzr )
      deallocate( xtmp )
C
```

```
C     Done.
C
      RETURN
      END
C
C*********************************************************************
```

A.2 C++ Interface to SuperLU

```
/*
 * -- SuperLU routine (version 2.0) --
 * Univ. of California Berkeley, Xerox Palo Alto Research Center,
 * and Lawrence Berkeley National Lab.
 * November 15, 1997
 * new routine created by Robert Tramel
 * June 8, 2000.
 *
 */
#include <stdlib.h>
#include <stdio.h>

#include "zsp_defs.h"
#include "util.h"
#include "Cnames.h"



extern "C" {


extern void
factor(SuperMatrix *A, int *perm_c, int *perm_r, SuperMatrix *L,
       SuperMatrix *U, int *info );


void
sp_solve__(int *n, int *nnz, doublecomplex *values,
          int *ja, int *ia, int *ib, int *ie, int *jb, int *je,
```

```
        int *zndx,  int *transpose,
        doublecomplex *bin, doublecomplex *bout,
        int *nzone, int *fac, int* solve, int *info)

{
    static SuperMatrix *L=NULL, *U=NULL;
    if(L == NULL)
       L = new SuperMatrix[27];
    if(U == NULL)
       U = new SuperMatrix[27];
    SCformat *Lstore;
    NCformat *Ustore;
    static int **perm_r=NULL; /* row permutations from partial pivoting */
    static int **perm_c=NULL; /* column permutation vector */
    if(perm_r == NULL) {
       perm_r = new int*[27];
       int nz;
       for (nz = 0; nz < 27; ++nz) {
            perm_r[nz]=NULL;
       }
    }
    if(perm_c == NULL) {
       perm_c = new int*[27];
        int nz;
        for (nz = 0; nz < 27; ++nz) {
             perm_c[nz]=NULL;
        }
    }
    int *ia_loc;
    int *ja_loc;
    int nrow_loc;
    doublecomplex *v_loc;
    doublecomplex *bloc;
    int      panel_size, permc_spec, i, ldb, nrhs, nz;
    int i1, i2, j1, j2;
    int n1, n2, nc, nsp;
    int nrowc;
    int nsp_pre;
    mem_usage_t   mem_usage;
    char     trans[1];
```

```
/* Adjust to 0-based indexing */
for (i = 0; i < *nnz; ++i) --ja[i];
for (i = 0; i <= *n; ++i) --ia[i];
for (nz = 0; nz < *nzone; ++nz) --ib[nz];
for (nz = 0; nz < *nzone; ++nz) --ie[nz];
for (nz = 0; nz < *nzone; ++nz) --jb[nz];
for (nz = 0; nz < *nzone; ++nz) --je[nz];
for (nz = 0; nz < *nzone; ++nz) --zndx[nz];

nsp_pre = 0;

if ( *fac == 1) {
   for (nz = 0; nz < *nzone; ++nz) {
       if(perm_r[zndx[nz]]==NULL) {
       SuperMatrix A;
       i1 = ib[nz];
       i2 = ie[nz];
       j1 = jb[nz];
       j2 = je[nz];
       nrow_loc = ie[nz] - ib[nz] + 1;
       nsp = 0;
       for( i = i1; i <= i2; i++) {
          n1 = ia[i];
          n2 = ia[i+1] -1;
          for (nc = n1; nc <= n2; nc++ ) {
            if(ja[nc] >= j1 && ja[nc] <= j2 ) nsp ++;
          }
       }
       printf(" NSP = %d\n",nsp);
       if ( !(v_loc  = doublecomplexMalloc(nsp)) ) {
          ABORT("Malloc fails for v_loc.");
       }
       if ( !(ia_loc = intMalloc(nrow_loc+1)) ) {
          ABORT("Malloc fails for ia_loc.");
       }
       if ( !(ja_loc = intMalloc(nsp)) ) {
          ABORT("Malloc fails for ja_loc.");
       }
       ia_loc[0] = 0;
       nsp = 0;
       for( i = i1; i <= i2; i++) {
          n1 = ia[i];
```

```
   n2 = ia[i+1] -1;
   nrowc = 0;
   for (nc = n1; nc <= n2; nc++ ) {
     if(ja[nc] >= j1 && ja[nc] <= j2 ) {
        nrowc++;
        v_loc[nsp]  = values[nc];
        ja_loc[nsp] = ja[nc]-j1;
        nsp ++;
     }
   }
   ia_loc[i-i1+1] = ia_loc[i-i1] + nrowc;
}
zCreate_CompRow_Matrix(&A, nrow_loc, nrow_loc, nsp, v_loc,
                       ja_loc, ia_loc, NR, _Z, GE);

if ( !(perm_r[zndx[nz]] = intMalloc(nrow_loc)) )
    ABORT("Malloc fails for perm_r[].");
if ( !(perm_c[zndx[nz]] = intMalloc(nrow_loc)) )
    ABORT("Malloc fails for perm_c[].");

/*
 * Get column permutation vector perm_c[], according to permc_spec:
 *   permc_spec = 0: use the natural ordering
 *   permc_spec = 1: use minimum degree ordering A'*A
 *   permc_spec = 2: use minimum degree ordering A'+A
 *   permc_spec = 3: use approximate minimum degree column ordering
 */
permc_spec = 3;
get_perm_c(permc_spec, &A, perm_c[zndx[nz]]);

panel_size = sp_ienv(1);

factor(&A, perm_c[zndx[nz]], perm_r[zndx[nz]],
       &L[zndx[nz]], &U[zndx[nz]], info);


if ( *info == 0 ) {

    Lstore = (SCformat *) L[zndx[nz]].Store;
    Ustore = (NCformat *) U[zndx[nz]].Store;
    printf("No of nonzeros in factor L = %d\n", Lstore->nnz);
    printf("No of nonzeros in factor U = %d\n", Ustore->nnz);
```

```
            printf("No of nonzeros in L+U = %d\n", Lstore->nnz + Ustore->nnz);
            nsp_pre += Lstore->nnz + Ustore->nnz;

            zQuerySpace(&L[zndx[nz]], &U[zndx[nz]], panel_size, &mem_usage);
            printf("L\\U MB %.3f\ttotal MB needed %.3f\texpansions %d\n",
                   mem_usage.for_lu/1e6, mem_usage.total_needed/1e6,
                   mem_usage.expansions);

        } else {
            printf("factor() error returns INFO= %d\n", *info);
            if ( *info <= *n ) { /* factorization completes */
                zQuerySpace(&L[zndx[nz]], &U[zndx[nz]], panel_size,
                            &mem_usage);
                printf("L\\U MB %.3f\ttotal MB needed %.3f\texpansions %d\n",
                       mem_usage.for_lu/1e6, mem_usage.total_needed/1e6,
                       mem_usage.expansions);
            }
        }
        Destroy_SuperMatrix_Store(&A);
        free(v_loc);
        free(ia_loc);
        free(ja_loc);
        }
    }
    printf("Total Number of non-zero elements in the preconditioner is %d\n"
           ,nsp_pre);
}

if ( *solve == 1 ) {
    for (nz = 0; nz < *nzone; ++nz) {
        SuperMatrix B;
        i1 = ib[nz];
        i2 = ie[nz];
        nrow_loc = ie[nz] - ib[nz] + 1;
        /* Solve the system A*X=B, overwriting B with X. */
        if ( !(bloc = doublecomplexMalloc(nrow_loc)) )
            ABORT("Malloc fails for bloc.") ;
        for (i = i1; i <= i2; ++i) bloc[i-i1] = bin[i];
        ldb = nrow_loc;
        nrhs = 1;
        zCreate_Dense_Matrix(&B, nrow_loc, nrhs, bloc, ldb, DN, _Z, GE);
        (*transpose == 1) ? *trans = 'T' : *trans = 'N';
```

```
            //*trans = 'T';
            zgstrs (trans, &L[zndx[nz]], &U[zndx[nz]], perm_r[zndx[nz]],
                    perm_c[zndx[nz]], &B, info);
            for (i = i1; i <= i2; ++i) bout[i] = bloc[i-i1];
            /*printf("zgstrs() returns INFO= %d\n", *info);*/
            Destroy_SuperMatrix_Store(&B);
            free(bloc);
        }
    }

    /* Restore to 1-based indexing */
    for (i = 0; i < *nnz; ++i) ++ja[i];
    for (i = 0; i <= *n; ++i) ++ia[i];
    for (nz = 0; nz < *nzone; ++nz) ++ib[nz];
    for (nz = 0; nz < *nzone; ++nz) ++ie[nz];
    for (nz = 0; nz < *nzone; ++nz) ++jb[nz];
    for (nz = 0; nz < *nzone; ++nz) ++je[nz];
    for (nz = 0; nz < *nzone; ++nz) ++zndx[nz];
}

}
```

A.3 C Sparse Matrix Factorization Routines

```
/*
 * -- SuperLU routine (version 2.0) --
 * Univ. of California Berkeley, Xerox Palo Alto Research Center,
 * and Lawrence Berkeley National Lab.
 * November 15, 1997
 * new routine created by Robert Tramel
 * June 8, 2000.
 *
 */
#include "zsp_defs.h"
#include "util.h"

void
```

```
factor(SuperMatrix *A, int *perm_c, int *perm_r, SuperMatrix *L,
      SuperMatrix *U,  int *info )
{
/*
 * Purpose
 * =======
 *
 * FACTOR factors a A, using the LU factorization from ZGSTRF
 * It performs the following steps:
 *
 *   1. If A is stored column-wise (A->Stype = NC):
 *
 *      1.1. Permute the columns of A, forming A*Pc, where Pc
 *           is a permutation matrix. For more details of this step,
 *           see sp_preorder.c.
 *
 *      1.2. Factor A as Pr*A*Pc=L*U with the permutation Pr determined
 *           by Gaussian elimination with partial pivoting.
 *           L is unit lower triangular with offdiagonal entries
 *           bounded by 1 in magnitude, and U is upper triangular.
 *
 *   2. If A is stored row-wise (A->Stype = NR), apply the
 *      above algorithm to the transpose of A:
 *
 *      2.1. Permute columns of transpose(A) (rows of A),
 *           forming transpose(A)*Pc, where Pc is a permutation matrix.
 *           For more details of this step, see sp_preorder.c.
 *
 *      2.2. Factor A as Pr*transpose(A)*Pc=L*U with the permutation Pr
 *           determined by Gaussian elimination with partial pivoting.
 *           L is unit lower triangular with offdiagonal entries
 *           bounded by 1 in magnitude, and U is upper triangular.
 *
 *
 *   See supermatrix.h for the definition of 'SuperMatrix' structure.
 *
 * Arguments
 * =========
 *
 * A        (input) SuperMatrix*
 *          Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
 *          of linear equations is A->nrow. Currently, the type of A can be:
```

```
*          Stype = NC or NR; Dtype = _Z; Mtype = GE. In the future, more
*          general A will be handled.
*
* perm_c   (input/output) int*
*          If A->Stype = NC, column permutation vector of size A->ncol
*          which defines the permutation matrix Pc; perm_c[i] = j means
*          column i of A is in position j in A*Pc.
*          On exit, perm_c may be overwritten by the product of the input
*          perm_c and a permutation that postorders the elimination tree
*          of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
*          is already in postorder.
*
*          If A->Stype = NR, column permutation vector of size A->nrow
*          which describes permutation of columns of transpose(A)
*          (rows of A) as described above.
*
* perm_r   (output) int*
*          If A->Stype = NC, row permutation vector of size A->nrow,
*          which defines the permutation matrix Pr, and is determined
*          by partial pivoting.  perm_r[i] = j means row i of A is in
*          position j in Pr*A.
*
*          If A->Stype = NR, permutation vector of size A->ncol, which
*          determines permutation of rows of transpose(A)
*          (columns of A) as described above.
*
* L        (output) SuperMatrix*
*          The factor L from the factorization
*              Pr*A*Pc=L*U              (if A->Stype = NC) or
*              Pr*transpose(A)*Pc=L*U   (if A->Stype = NR).
*          Uses compressed row subscripts storage for supernodes, i.e.,
*          L has types: Stype = SC, Dtype = _Z, Mtype = TRLU.
*
* U        (output) SuperMatrix*
*          The factor U from the factorization
*              Pr*A*Pc=L*U              (if A->Stype = NC) or
*              Pr*transpose(A)*Pc=L*U   (if A->Stype = NR).
*          Uses column-wise storage scheme, i.e., U has types:
*          Stype = NC, Dtype = _Z, Mtype = TRU.
*
*
* info     (output) int*
```

```
*         = 0: successful exit
*         > 0: if info = i, and i is
*              <= A->ncol: U(i,i) is exactly zero. The factorization has
*                 been completed, but the factor U is exactly singular,
*                 so the solution could not be computed.
*              > A->ncol: number of bytes allocated when memory allocation
*                 failure occurred, plus A->ncol.
*
*/
    double   t1;        /* Temporary time */
    char     refact[1];
    SuperMatrix *AA=NULL; /* A in NC format used by the factorization routine.*/
    SuperMatrix AC; /* Matrix postmultiplied by Pc */
    int      lwork = 0, *etree, i;

    /* Set default values for some parameters */
    double   diag_pivot_thresh = 1.0;
    double   drop_tol = 0;
    int      panel_size;     /* panel size */
    int      relax;          /* no of columns in a relaxed snodes */
    double   *utime;
    extern SuperLUStat_t SuperLUStat;

    /* Test the input parameters ... */
    *info = 0;
    /*Bstore = B->Store;*/
    if ( A->nrow != A->ncol || A->nrow < 0 ||
         (A->Stype != NC && A->Stype != NR) ||
         A->Dtype != _Z || A->Mtype != GE )
        *info = -1;
    if ( *info != 0 ) {
        i = -(*info);
        xerbla_("factor", &i);
        return;
    }

    *refact = 'N';
    panel_size = sp_ienv(1);
    relax = sp_ienv(2);

    StatInit(panel_size, relax);
    utime = SuperLUStat.utime;
```

```
    /* Convert A to NC format when necessary. */
    if ( A->Stype == NR ) {
        NRformat *Astore = A->Store;
        AA = (SuperMatrix *) SUPERLU_MALLOC( sizeof(SuperMatrix) );
        zCreate_CompCol_Matrix(AA, A->ncol, A->nrow, Astore->nnz,
                               Astore->nzval, Astore->colind, Astore->rowptr,
                               NC, A->Dtype, A->Mtype);
    } else if ( A->Stype == NC ) AA = A;

    etree = intMalloc(A->ncol);

    t1 = SuperLU_timer_();
    sp_preorder(refact, AA, perm_c, etree, &AC);
    utime[ETREE] = SuperLU_timer_() - t1;

    /*printf("Factor PA = LU ... relax %d\tw %d\tmaxsuper %d\trowblk %d\n",
          relax, panel_size, sp_ienv(3), sp_ienv(4));*/
    t1 = SuperLU_timer_();
    /* Compute the LU factorization of A. */
    zgstrf(refact, &AC, diag_pivot_thresh, drop_tol, relax, panel_size,
           etree, NULL, lwork, perm_r, perm_c, L, U, info);
    utime[FACT] = SuperLU_timer_() - t1;

    t1 = SuperLU_timer_();
    utime[SOLVE] = SuperLU_timer_() - t1;

    SUPERLU_FREE (etree);
    Destroy_CompCol_Permuted(&AC);
    if ( A->Stype == NR ) {
        Destroy_SuperMatrix_Store(AA);
        SUPERLU_FREE(AA);
    }

    /*PrintStat( &SuperLUStat );
    StatFree();*/

}
```

Vita

Robert Wallace Tramel was born on July 26, 1963 in McMinnville Tn. He attended Tennessee Technological University where he graduated with a B. S. degree cum laude in physics in 1986. In 1991 he graduated from the University of Tennessee, Space Institute with a degree in applied mathematics. In 1991 Robert took a job with Calspan Corporation at the Arnold Engineering Development Center working in the field of computational fluid dynamics. In 1997 Robert joined Micro Craft Inc. in Tullahoma Tn where he worked on the development of numerical algorithms for the computation of time accurate flow-fields of rotor craft configurations on massively parallel computers. In January 2000 Robert joined CFD Research corporation in Huntsville Al where he works in the area of computational electromagnetics. In his spare time time Robert likes to cook, listen to all kinds of music, read, and drink the finest malt beverages in the world. He is fanatical about Tennessee football to the point where long term psychoanalysis appears eminent. Robert has recently been informed that he may be a winner in the latest publishers clearing house sweep stakes.