Acknowledgments

I would like to take this opportunity to express my gratitude and appreciation for the assistance and support of the following people.

My heartfelt thanks go to George F. Pinder, who served as my advisor throughout the research and compilation of this work. To categorize him as a mere “advisor,” however, does not due justice to the role he played in overseeing some nebulous ideas coalesce into the body of knowledge that is this work. What I shall remember best are the times we spent in his office discussing mathematical ideas about which he often had probing insights that I had barely begun to glimpse. He gave generously of his time and energy and has earned my profound respect and affection.

I also thank Ilene Carpenter, my sponsor at Cray Research/SGI, for providing access to the T3E supercomputer. If the only assistance I received from her was the ability to use the T3E, she would have contributed significantly to this work. However, she went way beyond the call of duty by answering all my questions promptly, accurately, and fully and on occasion stayed awake late at night to aid me at particularly critical junctures. If, in the future, I have the opportunity to assist someone in a manner similar to how she helped me, I will remember her example as an ideal worthy of emulation.

I also am grateful for the assistance of Joseph Guarnaccia who, in the early stages of my research, always dropped what he was doing and gave me his undivided attention, often for considerable periods of time, whenever I knocked at his office door. It would be difficult to find a person more willing to take time away from his own work to answer the questions of a relative novice.

Finally, I thank my wife, Frances Wolfe, for endless encouragement and support, particularly
during the long spells of frustration that are inevitable in the research process. She graciously
endured countless occasions when my mind was more focused on mathematics than on topics of her
preference.
Contents

Acknowledgments ii

List of Tables vii

List of Figures viii

1 Introduction 1

1.1 Collocation 1

1.1.1 Hermite interpolation in one dimension 1

1.1.2 Collocation in one spatial dimension 2

1.1.3 Collocation in two spatial dimensions 5

1.1.4 Red-Black numbering 6

1.2 Conjugate gradient methods 7

1.2.1 Preconditioned Bi-CGSTAB method 8

1.2.2 The block Red-Black Gauss-Seidel preconditioner $P$ 9

1.2.3 Matrix-vector multiplication in the Bi-CGSTAB algorithm 11

2 Eigenvalue Analysis 15

2.1 Preliminaries 15

2.2 Entries of the matrix for Poisson’s equation 17

2.2.1 The eigenvalues of $\hat{J}$ for the case $\xi = \frac{1}{\sqrt{12}}$ 21

2.2.2 The eigenvalues of $\hat{J}$ for the case of general $\xi \in (0, \frac{1}{2})$ 35
2.3 A model parabolic equation ........................................... 36

3 Shifting the Collocation Points to Minimize \( \| \sigma (I - P^{-1}A) \|_2 \) .................................................. 39

3.1 Poisson’s equation .................................................. 40

3.2 Model parabolic equation .......................................... 49

3.3 Flow equation .................................................... 57

3.4 Transport equation ................................................ 57

4 The Eigenproblem of \( C^{-1}A \) ........................................ 60

4.1 One dimensional formulation ...................................... 60

4.2 Two-dimensional formulation .................................... 68

5 Application to Multiphase Equations .............................. 78

5.1 The PDEs of multiphase flow and transport ...................... 78

5.2 Results .......................................................... 83

5.2.1 Experiment 1: LNAPL spill .................................. 84

5.2.2 Experiment 2: DNAPL spill .................................. 85

5.2.3 Experiment 3: DNAPL vapor transport ...................... 86

5.3 Interpretation of results ......................................... 87

5.4 Comparison with GMRES with our preconditioner .............. 87

6 Parallel Implementation ............................................... 93

6.1 The allocation of portions of A, x, and b to the various processors .................................................... 94

6.2 Choice of architecture ........................................... 95

6.3 Parallel implementation of preconditioned Bi-CGSTAB .......... 96

6.3.1 Saxpy ...................................................... 96

6.3.2 Scalar arithmetic ............................................ 97

6.3.3 Vector inner products ....................................... 97

6.3.4 Psolve .................................................... 98

6.3.5 Matrix-vector multiplication ................................ 102
6.4 Implementation on the Cray T3E ................................................... 103

6.4.1 Speedup ................................................................. 104

6.4.2 Scalability and the isoefficiency function ......................... 105

7 Literature Review ......................................................... 112

7.1 Bi-CGSTAB ............................................................ 112

7.2 Collocation .............................................................. 118

7.3 Collocation applied to multiphase problems ....................... 124

8 Summary and Conclusions ............................................... 125

Bibliography ................................................................. 128
List of Tables

3.1 Results for Poisson’s equation with $\xi = \frac{1}{\sqrt{12}}$ and analytical solution (3.1) for various values of $m$ .......................... 52

3.2 Results for Poisson’s equation with $\xi = \frac{1}{\sqrt{12}}$ and analytical solution (3.2) for various values of $m$ .......................... 52

3.3 Number of iterations required for convergence of the model parabolic equation over one time step for various time step sizes .......................... 56

5.1 Results from Experiment 1. $D =$ results for DSLUGM. $B =$ results for Bi-CGSTAB. 84

5.2 Results from Experiment 2. $D =$ results for DSLUGM. $B =$ results for Bi-CGSTAB. 87

5.3 Results from Experiment 3. $D =$ results for DSLUGM. $B =$ results for Bi-CGSTAB. 87

5.4 Results from Experiment 1. $D =$ results for DSLUGM. $B =$ results for GMRES with our preconditioner. .......................... 90

5.5 Results from Experiment 2. $D =$ results for DSLUGM. $B =$ results for GMRES with our preconditioner. .......................... 90

6.1 Speedup results .......................... 105

6.2 Efficiency results .......................... 107

6.3 Computations required for a Bi-CGSTAB iteration .......................... 107

6.4 Example of data correlating to the theoretical isoefficiency function .......................... 111
List of Figures

1.1  Red-Black numbering of equations and unknowns ............................... 12
1.2  Structure of matrix $A$ arising from red-black numbering scheme ............... 13
1.3  Structure of preconditioning matrix $P$ ............................................... 14

3.1  Comparison of numerical results, condition numbers and $\| \sigma (I - P^{-1}A) \|_2$ for Poisson's equation for $m = 10$ ........................................ 41
3.2  Comparison of numerical results, condition numbers and $\| \sigma (I - P^{-1}A) \|_2$ for Poisson's equation for $m = 20$ ........................................ 42
3.3  Optimal value of $\xi$ for various values of $m$ for Poisson's equation .......... 43
3.4  Eigenvalues of flavor 1 for Poisson's equation for various values of $\xi$ for $m = 10$ ............................................................... 44
3.5  Eigenvalues of flavor 2 for Poisson's equation for various values of $\xi$ for $m = 10$ ............................................................... 45
3.6  Eigenvalues of flavor 3 for Poisson's equation for various values of $\xi$ for $m = 10$ ............................................................... 46
3.7  Eigenvalues of flavor 4 for Poisson's equation for various values of $\xi$ for $m = 10$ ............................................................... 47
3.8  Eigenvalues of flavor 5 for Poisson's equation for various values of $\xi$ for $m = 10$ ............................................................... 48
3.9  Eigenvalues of flavor 6 for Poisson's equation for various values of $\xi$ for $m = 10$ ............................................................... 49
3.10 Eigenvalues of flavor 7 for Poisson's equation for various values of $\xi$ for $m = 10$ ............................................................... 50
3.11 Eigenvalues of flavor 8 for Poisson's equation for various values of $\xi$ for $m = 10$ ............................................................... 51
3.12 $\| \sigma (I - (P^*)^{-1}A^*) \|_2$ as a function of $\xi$ for various values of $\tau$ for $m = 20$ ............................................................... 53
3.13 $\| \sigma (I - (P^*)^{-1}A^*) \|_2$ corresponding to optimal $\xi$ as a function of $\tau$ for various values of $m$ ............................................................... 54
3.14 Optimal $\xi$ as a function of $\tau$ for various values of $m$ ............................................................... 55
5.1 “Improvement” solving each linear system in Experiment 1 using Bi-CGSTAB with our preconditioner. The solution of each pressure equation is indicated by an asterisk, (5.4) by a circle, and (5.5) by a ‘plus’ sign. .................................................. 86

5.2 “Improvement” solving each linear system in Experiment 2 using Bi-CGSTAB with our preconditioner. The solution of each pressure equation is indicated by an asterisk, (5.4) by a circle, and (5.5) by a ‘plus’ sign. .................................................. 88

5.3 “Improvement” solving each linear system in Experiment 3 using Bi-CGSTAB with our preconditioner. The solution of each pressure equation is indicated by an asterisk, (5.4) by a circle, (5.5) by a ‘plus’ sign, (5.2) by an ‘x’, and (5.3) by a square. .... 89

5.4 “Improvement” solving each linear system in Experiment 1 using GMRES with our preconditioner. The solution of each pressure equation is indicated by an asterisk, (5.4) by a circle, and (5.5) by a ‘plus’ sign. .................................................. 91

5.5 “Improvement” solving each linear system in Experiment 2 using GMRES with our preconditioner. The solution of each pressure equation is indicated by an asterisk, (5.4) by a circle, and (5.5) by a ‘plus’ sign. .................................................. 92

6.1 Speedup curve for \( m_x = m_y = 128 \). The upper curve represents the unattainable ideal of linear speedup. .............................................................. 106
Chapter 1

Introduction

1.1 Collocation

1.1.1 Hermite interpolation in one dimension

Let \( u(x) \) be a differentiable function defined on the interval \( I = [a, b] \). Partition \( I \) into \( m \) linear finite elements \([x_j, x_{j+1}], \) \( j = 0, 1, 2, \ldots m-1 \), where the \( m+1 \) mesh points (or nodes) are \( a = x_0 < x_1 < x_2 < \cdots < x_m = b \). Let \( h_j = x_j - x_{j-1}, \) \( j = 1, 2, \ldots, m \).

Consider the Hermite cubic polynomials, defined for \( \eta \) in the interval \([-\frac{1}{2}, \frac{1}{2}]\):

\[
f_j(x) = \begin{cases} 
\frac{1}{2} (1 + 2\eta)^2 (1 - \eta), & x_{j-1} \leq x = x_j + (\eta - \frac{1}{2}) h_j \leq x_j \\
\frac{1}{2} (1 - 2\eta)^2 (1 + \eta), & x_j \leq x = x_j + (\eta + \frac{1}{2}) h_{j+1} \leq x_{j+1} \\
0, & \text{otherwise}
\end{cases}
\] (1.1)

and

\[
g_j(x) = \begin{cases} 
\frac{h_j}{2} (2\eta + 1)^2 (2\eta - 1), & x_{j-1} \leq x = x_j + (\eta - \frac{1}{2}) h_j \leq x_j \\
\frac{h_{j+1}}{2} (2\eta - 1)^2 (2\eta + 1), & x_{j} \leq x = x_j + (\eta + \frac{1}{2}) h_{j+1} \leq x_{j+1} \\
0, & \text{otherwise}
\end{cases}
\] (1.2)
\( j = 0, 1, 2, \ldots, m \). The functions \( f_j(x) \) and \( g_j(x) \) are defined such that

\[
f_j(x_i) = \begin{cases} 
1, & i = j \\
0, & \text{otherwise}
\end{cases}
\]

and

\[
dg_j(x_i) = \begin{cases} 
1, & i = j \\
0, & \text{otherwise}
\end{cases}
\]

for \( i, j = 0, 1, 2, \ldots, m \).

If we know the values \( u_j = u(x_j) \) and \( u_j' = \frac{du}{dx}(x_j) \) for \( j = 0, 1, 2, \ldots, m \), then the piecewise cubic polynomial interpolating the values \( u_j \) and \( u_j' \), \( j = 0, 1, 2, \ldots, m \) is

\[
\hat{u}(x) = \sum_{j=0}^{m} [u_j f_j(x) + u_j' g_j(x)]. \quad (1.3)
\]

### 1.1.2 Collocation in one spatial dimension

Consider the second order linear ordinary differential equation (ODE)

\[
\mathcal{L}[u](x) = u'' - p(x) u = H(x) \quad (1.4)
\]

with Dirichlet boundary conditions

\[
u(a) = u_a
\]

\[
u(b) = u_b. \quad (1.5)
\]

If we introduce the interpolating polynomial (1.3) into the differential equation (1.4), we obtain

\[
\mathcal{L}[^{\hat{u}}](x) - H(x) = E(x), \quad (1.6)
\]

where \( E(x) \) is an error function. Our goal is to determine \( u_j \) and \( u_j' \) for \( j = 0, 1, 2, \ldots, m \) while controlling \( E(x) \).
With reference to (1.5), we see that we may let

\begin{align*}
  u_0 &= u_a \\
  u_m &= u_b.
\end{align*}

We therefore have \(2m\) undetermined coefficients (namely \(u_j\) for \(j = 1, 2, \ldots, m - 1\) and \(u'_j\) for \(j = 0, 1, 2, \ldots, m\)). To find their values, we must write \(2m\) equations in these \(2m\) undetermined coefficients. Since there are \(m\) finite elements, an appropriate option is to select two equations per element. These equations are generated by enforcing the condition \(E(x) = 0\) at two distinct “collocation points” in the interior of each finite element. If we let \(c_0 < c_1 < c_2 < \cdots < c_{2m-1}\) be the \(2m\) collocation points, we obtain from (1.6) the “collocation equations”

\[
  \mathcal{L} [\hat{u}] (c_i) = H (c_i), \quad (1.7)
\]

\(i = 0, 1, \ldots, 2m - 1\), which represents a system of \(2m\) linear algebraic equations in the unknowns \(u_j\) for \(j = 1, 2, \ldots, m - 1\) and \(u'_j\) for \(j = 0, 1, 2, \ldots, m\).

Some remarks are in order. First, should we elect to redefine the problem, one or both of the Dirichlet boundary conditions (1.5) could be replaced by the corresponding Neumann boundary conditions

\begin{align*}
  \frac{du}{dx} (a) &= u'_a = u'_0 \\
  \frac{du}{dx} (b) &= u'_b = u'_m.
\end{align*}

In this case, it is clear that although we have a slightly different set of undetermined coefficients, the number of these coefficients remains the same. Thus the collocation equations may be generated as described above.

Secondly and more interestingly, we consider how the collocation points are chosen within each finite element. It is well known [40, 41, 105] (given certain restrictions on \(\frac{\partial^2 u}{\partial x^2}\)) that if the collocation points within each finite element are chosen to correspond with the points of Gaussian quadrature (i.e., \(\eta \pm \frac{1}{\sqrt{2}}\) in (1.1) and (1.2)), then we obtain minimum discretization errors.
\[ u(x_j) - \tilde{u}(x_j) = O(h^4) \text{, where } h = \max \{ h_j : j = 1, 2, \ldots, m \}. \text{ When the Gauss points are chosen as the collocation points, the method is known as “orthogonal collocation” ([88], [105]).} \]

Let us adopt the following convention. Within each finite element (where \( \eta \) varies between \( -\frac{1}{2} \) and \( \frac{1}{2} \)), let us define the location of the two collocation points as \( \eta = \pm \xi \), where \( \xi \in (0, \frac{1}{2}) \). Then orthogonal collocation corresponds to the choice \( \xi = \frac{1}{\sqrt{n}} \).

We conclude this section with a simple example. If the ODE (1.4) to be solved is

\[ u'' = H(x) \]

with Dirichlet boundary conditions (1.5), then the collocation equations (1.7) become

\[ \sum_{j=0}^{m} \left[ f_j''(c_i) u_j + g_j'(c_i) u'_j \right] = H(c_i). \tag{1.8} \]

If we have \( m = 4 \) finite elements, then we may write (1.8) in matrix form as

\[ Ax = b, \tag{1.9} \]

where

\[
A = \begin{bmatrix}
g_0''(c_0) & f_0''(c_0) & g_0''(c_0) \\
g_1''(c_1) & f_1''(c_1) & g_1''(c_1) \\
f_0''(c_2) & g_0''(c_2) & f_0''(c_2) \\
f_1''(c_3) & g_1''(c_3) & f_1''(c_3) \\
f_2''(c_4) & g_2''(c_4) & f_2''(c_4) \\
f_3''(c_5) & g_3''(c_5) & f_3''(c_5) \\
f_4''(c_6) & g_4''(c_6) & f_4''(c_6) \\
f_5''(c_7) & g_5''(c_7) & f_5''(c_7)
\end{bmatrix}
\]
\[ \mathbf{x} = \begin{bmatrix} u_0' & u_1' & u_2' & u_3' & u_4' \end{bmatrix}^T, \]

and

\[ \mathbf{b} = \begin{bmatrix} H(c_0) - f_0''(c_0) u_a \\ H(c_1) - f_0''(c_1) u_a \\ H(c_2) \\ H(c_3) \\ H(c_4) \\ H(c_5) \\ H(c_6) - f_1''(c_6) u_b \\ H(c_7) - f_1''(c_7) u_b \end{bmatrix}. \]

Because \( A \) is a 2 \( \times \) 2 block tridiagonal matrix, efficient code may be written for the solution of (1.9).

### 1.1.3 Collocation in two spatial dimensions

Defining collocation in two dimensions proceeds quite naturally from the one-dimensional formulation given above. Details of the derivation of two-dimensional collocation may be found in [54], [86], [88].

Briefly, we are given a partial differential equation in two spatial dimensions (call them \( x \) and \( y \)) with Dirichlet and/or Neumann boundary conditions defined on a rectangular domain \( D = [a_x, b_x] \times [a_y, b_y] \) whose north and south boundaries are parallel to the \( x \)-axis and whose east and west boundaries are parallel to the \( y \)-axis. If we define a set \( X = \{ a_x = x_0 < x_1 < x_2 < \cdots < x_{m_x} = b_x \} \) for the \( x \)-direction and a set \( Y = \{ a_y = y_0 < y_1 < y_2 < \cdots < y_{m_y} = b_y \} \) for the \( y \)-direction, then the set of points

\[ \{(x_q, y_r): x_q \in X \text{ and } y_r \in Y\} \]

defines the set of nodes for our two dimensional problem. Clearly, this partitions the domain \( D \) into \( m_x m_y \) rectangular finite elements, where \( m_x \) is the number of elements in the \( x \)-direction and \( m_y \) (assumed to be even throughout this work) is the number of elements in the \( y \)-direction. By
analogy to (1.3), the bi-cubic piecewise polynomial interpolating $u_{q,r} = u(x_q, y_r)$, $u_{q,r}^x = \frac{du}{dx}(x_q, y_r)$, $u_{q,r}^y = \frac{du}{dy}(x_q, y_r)$ for $q = 0, 1, \ldots, m_x$ and $r = 0, 1, \ldots, m_y$ is

$$\hat{u}(x,y) = \sum_{q=0}^{m_x} \sum_{r=0}^{m_y} \left[ u_{q,r} f_q(x) f_r(y) + u_{q,r}^x g_q(x) f_r(y) + u_{q,r}^y g_q(x) f_r(y) + u_{q,r}^{xy} g_q(x) g_r(y) \right].$$

(1.11)

Of the $4(m_x+1)(m_y+1)$ coefficients in (1.11), $4(m_x + m_y + 1)$ are known. To see this, consider the case of Dirichlet boundary conditions on all four sides of the domain $D$. Then at the north and south boundary nodes, we know $u$ and, by differentiating, we know also $\frac{du}{dx}$. Analogously, on the east and west boundaries, we know both $u$ and $\frac{du}{dy}$. It is thus seen that we know three coefficients at each of the four corner nodes and two coefficients at each boundary node not at a corner of the domain, for a total of $4(m_x + m_y + 1)$ known coefficients. This leaves $4m_x m_y$ undetermined coefficients, requiring $4m_x m_y$ equations. Since we have $m_x m_y$ finite elements, we select four equations per element. We therefore choose four collocation points in the interior of each finite element where (1.11) solves the PDE exactly.

1.1.4 Red-Black numbering

One of the attractive properties of collocation is the freedom it provides in numbering the equations and unknowns. One choice is given in [54], [86], and [100], which results in the system of linear algebraic equations

$$\bar{A}\bar{x} = \bar{b}.$$  

(1.12)

We introduce a red-black numbering which corresponds to permuting the rows and columns of $\bar{A}$ in a particular way, resulting in a new matrix $A$. This permutation defines a similarity transformation applied to $\bar{A}$, the importance of which is that the eigenvalue analysis applied to $\bar{A}$ in [86] applies also to $A$. Obviously, the permutation must be applied also to $\bar{x}$ and $\bar{b}$, resulting in new vectors $\bar{x}$ and $\bar{b}$. Our red-black numbering scheme, for the case where $m_x = m_y = 4$, is shown in Figure 1.1. The location of each of the four collocation points (equations) in the interior of each finite element is indicated by a red or black number. In Figure 1.1, the red equations are numbered
0 through 31 while the black equations are numbered 32 through 63. The unknowns, located at
nodes (i.e., corners of finite elements), are indicated by green numbers. For example, the unknowns
numbered 38, 39, 40, and 41 correspond to the unknowns \( u_{2,1}^u, u_{2,1}^{xy}, u_{2,1}^{y}, \) and \( u_{2,1}^{xy}, \) respectively.

This numbering produces the system of linear algebraic equations

\[
Ax = b, \quad (1.13)
\]

where the structure of \( A \) is depicted in Figure 1.2. Considering the structure of matrix \( A \) in Figure
1.2, we write \( Ax = b \) as

\[
\begin{bmatrix}
R & U \\
L & B
\end{bmatrix}
\begin{bmatrix}
x_R \\
x_B
\end{bmatrix} =
\begin{bmatrix}
b_R \\
b_B
\end{bmatrix}.
\]

\[
(1.14)
\]

1.2 Conjugate gradient methods

To solve (1.13) when the matrix \( A \) is an arbitrary non-symmetric matrix, the two most widely used
approaches are the GMRES [114] and Bi-CGSTAB [128] methods. For this work, we choose the
Bi-CGSTAB method for two reasons. First, the Bi-CGSTAB method lends itself more readily to
parallel processing than does GMRES. Also, there is evidence [32], [116], [135] that Bi-CGSTAB is
superior for a large class of problems.

As with any conjugate gradient method, the fastest convergence is obtained when \( A \) in (1.13) is
“close” to the identity matrix [57]. We therefore choose a “preconditioning” matrix \( P \) and apply
Bi-CGSTAB to

\[
P^{-1}Ax = P^{-1}b
\]

where the matrix \( P \) is chosen such that \( P \approx A \) and linear systems of the form \( Py = c \) are easy to
solve. For our purposes, we stipulate a third requirement: \( P \) must have a structure amenable to
parallel processing.
1.2.1 Preconditioned Bi-CGSTAB method

The preconditioned Bi-CGSTAB algorithm to solve (1.13) may be written [128]:

\[
\begin{align*}
\text{choose } x_0 \text{ as an initial guess for } x \\
r_0 &= b - Ax_0 \\
\text{choose } \bar{r} \text{ such that } \bar{r}^T r_0 \neq 0 \\
\rho_0 &= \alpha_0 = \omega_0 = 1 \\
\nu_0 &= p_0 = 0 \\
\text{iterate } i = 1, 2, 3, \ldots \\
\rho_i &= \bar{r}^T r_{i-1} \\
\beta_i &= \frac{\rho_i - \alpha_i}{\rho_{i-1} - \omega_i} \\
p_i &= r_{i-1} + \beta_i (p_{i-1} - \omega_i v_{i-1}) \\
\text{solve for } y_i: P y_i = p_i \\
\nu_i &= Ay_i \\
\alpha_i &= \frac{\rho_i}{\bar{r}^T v_i} \\
s_i &= r_{i-1} - \alpha_i v_i \\
\text{solve for } z_i: P z_i = s_i \\
t_i &= Az_i \\
\omega_i &= t_i^T s_i \\
x_i &= x_{i-1} + \alpha_i y_i + \omega_i z_i \\
r_i &= s_i - \omega_i t_i \\
\text{if } \|r_i\|_2 \text{ is small enough, then quit}
\end{align*}
\]

for a prescribed preconditioning matrix \( P \).
1.2.2 The block Red-Black Gauss-Seidel preconditioner $P$

If we were to solve (1.14) by a block Gauss-Seidel method, we would write

$$
\begin{bmatrix}
R & L & B \\
L & B & \\
B & &
\end{bmatrix}
\begin{bmatrix}
x_{R}^{(n+1)} \\
x_{B}^{(n+1)}
\end{bmatrix}
= 
\begin{bmatrix}
-U & \cdot & U \\
- & \cdot & \\
& & 
\end{bmatrix}
\begin{bmatrix}
x_{R}^{(n)} \\
x_{B}^{(n)}
\end{bmatrix}
+ 
\begin{bmatrix}
b_{R} \\
b_{B}
\end{bmatrix},
$$

where the superscript indicates iteration number. We are thus motivated to select our preconditioning matrix $P$ to be

$$P = 
\begin{bmatrix}
R \\
L \\
B
\end{bmatrix}.
$$

(1.16)

Since $P$, in some sense, contains 75% of the information of $A$, $P$ fulfills the requirement $P \approx A$. Also, as described below, solving systems of the form $Py = c$ is easily accomplished and is parallelizable.

With respect to Figure 1.2 and (1.16), we see that $P$ has the structure depicted in Figure 1.3 (for the case $m_x = m_y = 4$). With respect to Figure 1.3, we write

$$Py = c$$

(1.17)

as

$$
\begin{bmatrix}
R & L & B \\
L & B & \\
B & &
\end{bmatrix}
\begin{bmatrix}
y_{R} \\
y_{B}
\end{bmatrix}
= 
\begin{bmatrix}
c_{R} \\
c_{B}
\end{bmatrix}
$$

(1.18)

or, for the specific case $m_y = 4$, as

$$
\begin{bmatrix}
A_{F} & A_{1} & A_{L} \\
C_{F} & B_{0} & A_{0} \\
C_{2} & B_{L} & A_{2}
\end{bmatrix}
\begin{bmatrix}
y_{F} \\
y_{1} \\
y_{L}
\end{bmatrix}
= 
\begin{bmatrix}
c_{F} \\
c_{1} \\
c_{L}
\end{bmatrix}.
$$

(1.19)
To solve (1.17) or, equivalently, (1.18), we first solve

\[ R y_R = c_R \]

for \( y_R \); or, for the specific case \( m_y = 4 \), we solve

\[
\begin{bmatrix}
  A_F & y_F \\
  A_1 & y_1 \\
  A_L & y_L
\end{bmatrix} =
\begin{bmatrix}
  c_F \\
  c_1 \\
  c_L
\end{bmatrix}
\] (1.20)

for \( y_F, y_1, \) and \( y_L \). Because the non-zero entries of \( R \) reside only in its diagonal blocks \( A_F, A_1, A_3, A_5, \ldots, A_{m_y-3}, A_L \), it is clear that solving the subsystems in (1.20) are independent problems and hence can be solved simultaneously in parallel.

Once we have obtained \( y_R \) as described in the preceding paragraph, we may now find \( y_B \) similarly. With respect to (1.18) and (1.19), we write

\[ B y_B = c_B - L y_R \]

or, for the specific case \( m_y = 4 \), we solve

\[
\begin{bmatrix}
  A_0 & y_0 \\
  A_2 & y_2
\end{bmatrix} =
\begin{bmatrix}
  c_0 \\
  c_2
\end{bmatrix} - 
\begin{bmatrix}
  C_F & B_0 \\
  C_2 & B_L
\end{bmatrix}
\begin{bmatrix}
  y_F \\
  y_1 \\
  y_L
\end{bmatrix}
\] (1.21)

Because the non-zero entries of \( B \) reside only in its diagonal blocks \( A_0, A_2, A_4, \ldots, A_{m_y-2} \), it is clear that solving the subsystems in (1.21) are independent problems and hence can be solved simultaneously in parallel.

Not only can subsystems \( A_i y_i = c_i \) be solved simultaneously in parallel, but efficient code can be written for their solution because \( A_i \) is block tridiagonal for all \( i \). When \( i = F \) or \( i = L \), the blocks of \( A_i \) are \( 2 \times 2 \) matrices. When \( i = 0,1,2,\ldots,m_y - 2 \), the blocks are \( 4 \times 4 \) matrices.
1.2.3 Matrix-vector multiplication in the Bi-CGSTAB algorithm

During the iteration process in the Bi-CGSTAB algorithm, we note that every step of the form (1.17)

\[ Py = c \]

is immediately followed by a matrix-vector multiplication step of the form

\[ v = Ay. \]

Because of the manner in which \( P \) was selected, we see that we do not have to perform the complete matrix-vector multiplication \( v = Ay \):

\[
\begin{align*}
  v &= Ay = \begin{bmatrix} R & U \\ L & B \end{bmatrix} \begin{bmatrix} y_R \\ y_B \end{bmatrix} \\
  &= \begin{bmatrix} R \\ L \\ B \end{bmatrix} \begin{bmatrix} y_R \\ y_B \end{bmatrix} + \begin{bmatrix} U \end{bmatrix} \begin{bmatrix} y_R \\ y_B \end{bmatrix} \\
  &= Py + \begin{bmatrix} Uy_B \end{bmatrix} \\
  &= c + \begin{bmatrix} Uy_B \end{bmatrix} \\
  &= \begin{bmatrix} c_R + Uy_B \\ c_B \end{bmatrix}.
\end{align*}
\]

(1.22)

Thus, with our choice of \( P \), the matrix-vector multiplication steps in the Bi-CGSTAB algorithm are performed relatively inexpensively. To be precise, instead of performing the four multiplications \( Ry_R, Uy_B, Ly_L, \) and \( By_B \) and the two additions \( Ry_R + Uy_B \) and \( Ly_L + By_B \), we need perform only one multiplication \( (Uy_B) \) and one addition \( (c_R + Uy_B) \).
Figure 1.1: Red-Black numbering of equations and unknowns
Figure 1.2: Structure of matrix $A$ arising from red-black numbering scheme
Figure 1.3: Structure of preconditioning matrix $P$
Chapter 2

Eigenvalue Analysis

In this section, we assume that our rectangular domain $D$ is the unit square $S = [0, 1] \times [0, 1]$. Furthermore, we assume $m_x = m_y = m$ and that we have a uniform mesh which partitions $S$ into $m^2$ square finite elements, each of dimension $h \times h$. By analogy to the convention adopted in one-dimensional collocation, we say that the four collocation points in each finite element are: $(\xi, \xi)$, $(\xi, -\xi)$, $(-\xi, \xi)$, and $(-\xi, -\xi)$, where, as in the one-dimensional case, greatest accuracy ($O(h^4)$) prevails when $\xi \in (0, \frac{1}{2})$ is $\frac{1}{\sqrt{12}}$ [106]. In the discussion that follows, we use the Gaussian value $\xi = \frac{1}{\sqrt{12}}$, thus enabling us to use results in [86]. We will generalize these results for arbitrary $\xi \in (0, \frac{1}{2})$ in the following section.

2.1 Preliminaries

As reported in [113], the rate at which preconditioned conjugate gradient methods converge “depends on the global eigenvalue distribution [of $P^{-1}A$] more than anything else.” We are therefore motivated to find analytical formulae for the eigenvalues of $P^{-1}A$, buoyed by the knowledge that analytical formulae for the eigenvalues associated with solving (1.13) via a block Jacobi method were determined in [86]. Indeed, we use results reported in [86] in our discussion below as well as using the general approach of [86] as a model for determining the eigenvalues of $P^{-1}A$. We will use the term spectrum of a matrix to refer to the set whose entries are the eigenvalues of the matrix and
denote the spectrum by $\sigma$. We thus seek $\sigma(P^{-1}A)$.

At times, we will want to consider the vector whose entries are those of the set $\sigma(P^{-1}A)$. We will use the same notation, i.e., $\sigma(P^{-1}A)$, for both the vector and the set. We expect that this slight abuse of notation will not be confusing.

Because $P$ is a block $2 \times 2$ matrix, its inverse is computable [38] as

$$P^{-1} = \begin{bmatrix} R^{-1} & \hline -B^{-1}LR^{-1} & B^{-1} \end{bmatrix};$$

therefore

$$P^{-1}A = \begin{bmatrix} I & \hline R^{-1}U & \hline I - B^{-1}LR^{-1}U \end{bmatrix},$$

where $I$ represents the identity matrix of appropriate size. Recall we want $P^{-1}A$ to be “close” to $I$. This is clearly equivalent to

$$I - P^{-1}A = \begin{bmatrix} \hline -R^{-1}U & \hline B^{-1}LR^{-1}U \end{bmatrix}$$

being “close” to the “null” matrix (i.e., the matrix whose entries are all zero). The null matrix has all its eigenvalues equal to zero. Since $I - P^{-1}A$ may be viewed as a block upper-triangular matrix, its spectrum is given by the union of the spectra of those matrices on its diagonal blocks, namely the null matrix and $J' = B^{-1}LR^{-1}U$. We therefore expect the fastest convergence when the eigenvalues of $J'$ are clustered near the origin of the complex plane.

To simplify our eigenvalue analysis, we make two changes to the formulation above. First, noting that the eigenvalues of $J = LR^{-1}UB^{-1}$ and those of $J'$ are identical (because $J$ is obtained from $J'$ from a similarity transformation), we perform our analysis on $J$, which is much easier than using $J'$. Second, we replace $g_j(x)$ in (1.2) by $g'_j(x) = \frac{g_j(x)}{h}$ as done in [86] and [100]. It is clear that this replacement is equivalent to changing the matrix $A$ to $A' = AK$, where $K$ is a diagonal matrix whose diagonal entries are non-positive integer powers of $h$. In this case, we choose the
preconditioning matrix to be \( P^* = PK \). Then \((P^*)^{-1}A^* = K^{-1}P^{-1}AK\), which clearly has the same eigenvalues as \( P^{-1}A \) (again, because the former matrix is obtained from the latter one via a similarity transformation).

### 2.2 Entries of the matrix for Poisson’s equation

As mentioned above in reference to (1.12), our matrix \( A \) was obtained from \( \bar{A} \) via a similarity transformation, under which eigenvalues are invariant. We begin our analysis by examining the entries of \( \bar{A} \) that arise from Poisson’s equation

\[
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = H(x,y)
\]  

(2.1)

with Dirichlet boundary conditions.
An example with $m = 8$ is sufficiently illustrative. In this case,

$$
\begin{bmatrix}
A_2 & A_3 & -A_4 \\
A_4 & A_1 & -A_2 \\
A_1 & A_2 & A_3 & -A_4 \\
A_3 & A_4 & A_1 & -A_2 \\
A_1 & A_2 & A_3 & -A_4 \\
A_3 & A_4 & A_1 & -A_2 \\
A_1 & A_2 & A_3 & -A_4 \\
A_3 & A_4 & A_1 & -A_2 \\
A_1 & A_2 & A_3 & -A_4 \\
A_3 & A_4 & A_1 & -A_2 \\
A_1 & A_2 & A_3 & -A_4 \\
A_3 & A_4 & A_1 & -A_2 \\
A_1 & A_2 & -A_4 \\
A_3 & A_4 & -A_2 \\
\end{bmatrix}
$$

\[ A = \]

$$
\begin{bmatrix}
A_2 & A_3 & -A_4 \\
A_4 & A_1 & -A_2 \\
A_1 & A_2 & A_3 & -A_4 \\
A_3 & A_4 & A_1 & -A_2 \\
A_1 & A_2 & A_3 & -A_4 \\
A_3 & A_4 & A_1 & -A_2 \\
A_1 & A_2 & A_3 & -A_4 \\
A_3 & A_4 & A_1 & -A_2 \\
A_1 & A_2 & A_3 & -A_4 \\
A_3 & A_4 & A_1 & -A_2 \\
A_1 & A_2 & -A_4 \\
A_3 & A_4 & -A_2 \\
\end{bmatrix}
$$
where the submatrices $A_1, A_2, A_3, A_4$ are all $16 \times 16$ (in general, $2m \times 2m$) and have the structure

$$A_i = \begin{pmatrix}
  a_{i,2} & a_{i,3} & -a_{i,4} \\
  a_{i,4} & a_{i,1} & -a_{i,2} \\
  a_{i,1} & a_{i,2} & a_{i,3} & -a_{i,4} \\
  a_{i,4} & a_{i,3} & a_{i,1} & -a_{i,2} \\
  a_{i,1} & a_{i,2} & a_{i,3} & -a_{i,4} \\
  a_{i,4} & a_{i,3} & a_{i,1} & -a_{i,2} \\
  a_{i,1} & a_{i,2} & a_{i,3} & -a_{i,4} \\
  a_{i,4} & a_{i,3} & a_{i,1} & -a_{i,2}
\end{pmatrix}.$$  

For any $\xi \in (0, \frac{1}{\sqrt{12}})$, the entries $a_{i,j}, i, j = 1, 2, 3, 4$, are given below. Note the symmetry $a_{i,j} = a_{j,i}$.

For the Gaussian case $\xi = \frac{1}{\sqrt{12}}$, these entries reduce to those given in [86] and [100].

\begin{align*}
a_{1,1} &= \frac{12\xi(\xi-1)(1+2\xi)^2}{h^2} \\
a_{1,2} &= \frac{(1+2\xi)^2(12\xi^2-8\xi-1)}{2h^2} \\
a_{1,3} &= \frac{12\xi^2(3-4\xi^2)}{h^2} \\
a_{1,4} &= \frac{-1(1+2\xi)(1-2\xi-20\xi^2+24\xi^4)}{2h^2} \\
a_{2,2} &= \frac{(6\xi+1)(1+2\xi)^2(2\xi-1)}{4h^2} \\
a_{2,3} &= \frac{(1-2\xi)(1-2\xi-20\xi^2+24\xi^4)}{2h^2} \\
a_{2,4} &= \frac{(2\xi-1)(2\xi+1)(1-12\xi^2)}{4h^2} \\
a_{3,3} &= \frac{12\xi(2\xi-1)^2(\xi+1)}{h^2} \\
a_{3,4} &= \frac{(1-2\xi)^2(-1+8\xi+12\xi^2)}{2h^2} \\
a_{4,4} &= \frac{(2\xi-1)^2(2\xi+1)(6\xi-1)}{4h^2}
\end{align*}  

(2.2)
We now permute the rows and columns of $\bar{A}$ via a similarity transformation, obtaining

\[
\tilde{A} = \begin{bmatrix}
A_2 \\
A_1 & -A_2 \\
A_1 & A_2 \\
A_1 & -A_2 \\
A_1 & A_2 \\
\vdots & \vdots \\
A_1 & -A_2 \\
A_1 & A_2 \\
A_1 & -A_2 \\
A_1 & A_2 \\
\vdots & \vdots \\
A_1 & -A_2 \\
A_1 & A_2 \\
\vdots & \vdots \\
A_1 & -A_2 \\
A_1 & A_2 \\
\vdots & \vdots \\
A_1 & -A_2 \\
A_1 & A_2 \\
\vdots & \vdots \\
A_1 & -A_2 \\
A_1 & A_2 \\
\vdots & \vdots \\
A_1 & -A_2 \\
A_1 & A_2 \\
\vdots & \vdots \\
A_1 & -A_2 \\
A_1 & A_2 \\
\vdots & \vdots \\
A_1 & -A_2 \\
A_1 & A_2 \\
\vdots & \vdots \\
A_1 & -A_2 \\
A_1 & A_2 \\
\end{bmatrix}
\]

which we abbreviate

\[
\tilde{A} = \begin{bmatrix}
-\bar{A} \\
\bar{A} \\
\bar{A} \\
\bar{A} \\
\bar{A} \\
\vdots \\
\bar{A} \\
\bar{A} \\
\bar{A} \\
\bar{A} \\
\vdots \\
\bar{A} \\
\bar{A} \\
\vdots \\
\bar{A} \\
\bar{A} \\
\vdots \\
\bar{A} \\
\bar{A} \\
\vdots \\
\bar{A} \\
\bar{A} \\
\end{bmatrix}
\]

Let $\hat{J} = \hat{L}\hat{R}^{-1}\hat{U}\hat{B}^{-1}$. The matrix $A$ in (1.13) is obtained from $\tilde{A}$ via another similarity transformation defined by permuting the rows and columns of $\tilde{A}$ such that $\sigma(\tilde{A}) = \sigma(J)$. The eigenvalue analysis given below determines $\sigma(\hat{J})$. 

20
2.2.1 The eigenvalues of $\hat{J}$ for the case $\xi = \frac{1}{\sqrt{12}}$

Because $\hat{R}$ and $\hat{B}$ are both block diagonal, their inverses are easily computed. Noting that

$$\begin{bmatrix} A_1 & -A_2 \\ A_1 & A_2 \end{bmatrix}^{-1} = \frac{1}{2} \begin{bmatrix} A_1^{-1} & A_1^{-1} \\ -A_2^{-1} & A_2^{-1} \end{bmatrix},$$

we see that

$$\hat{R}^{-1} = \frac{1}{2} \begin{bmatrix} 2A_2^{-1} \\ & A_1^{-1} & A_1^{-1} \\ & -A_2^{-1} & A_2^{-1} \\ & & A_1^{-1} & A_1^{-1} \\ & & -A_2^{-1} & A_2^{-1} \\ & & & A_1^{-1} & A_1^{-1} \\ & & & -A_2^{-1} & A_2^{-1} \end{bmatrix} - 2A_2^{-1}$$

and

$$\hat{B}^{-1} = \frac{1}{2} \begin{bmatrix} A_1^{-1} & A_1^{-1} \\ -A_2^{-1} & A_2^{-1} \\ & A_1^{-1} & A_1^{-1} \\ & -A_2^{-1} & A_2^{-1} \\ & & A_1^{-1} & A_1^{-1} \\ & & -A_2^{-1} & A_2^{-1} \end{bmatrix}.$$
\( \hat{J} \) is thus seen to be

\[
\hat{J} = \begin{bmatrix}
S^2 - QS & SQ - Q^2 \\
SQ & S^2 & QS & Q^2 \\
Q^2 & QS & S^2 & SQ \\
SQ & S^2 & QS & Q^2 \\
Q^2 & QS & S^2 & SQ \\
SQ & S^2 & QS & Q^2 \\
Q^2 & QS & S^2 & SQ \\
SQ - Q^2 & S^2 - QS
\end{bmatrix}, \quad (2.3)
\]

where

\[
S = -\frac{1}{2} (A_3 A_1^{-1} + A_4 A_2^{-1}) \quad (2.4)
\]

and

\[
Q = -\frac{1}{2} (A_3 A_1^{-1} - A_4 A_2^{-1}) \quad (2.5)
\]

At this point we employ the strategy of [86], namely to determine \( \sigma(\hat{J}) \) for the case where \( S \) and \( Q \) are real scalars, and then use this result to determine \( \sigma(\hat{J}) \) when \( S \) and \( Q \) are \( 2m \times 2m \) matrices.

To compute the eigenvalues \( \mu \) and eigenvectors \( \mathbf{z} \) of \( \hat{J} \), we set

\[
\left( \hat{J} - \mu I \right) \mathbf{z} = 0, \quad (2.6)
\]
To apply Theorem 8.3 in [56] (as was done successfully in [86]) we must manipulate the first and last rows in (2.7) so that each set of two rows of the resulting matrix is identical. We obtain

\[
\begin{bmatrix}
S^2 - \mu I - QS & SQ - Q^2 \\
SQ & S^2 - \mu I \\
Q^2 & QS & S^2 - \mu I & SQ \\
\vdots & & \vdots & \vdots \\
\end{bmatrix}
= \begin{bmatrix}
z_{1,0} \\
z_{1,1} \\
z_{2,0} \\
z_{2,1} \\
z_{3,0} \\
z_{3,1} \\
z_{4,0} \\
z_{4,1} \\
\end{bmatrix}^T.
\]
where \( b_0 = Q^2 (z_{0,0} + z_{1,1}) + QS (z_{0,1} + z_{1,0}) \) and \( b_1 = Q^2 (z_{4,0} + z_{5,1}) + QS (z_{4,1} + z_{5,0}) \).

To easily apply Theorem 8.3 in [56], it is convenient to have the vector on the right side of (2.8) be identically zero. If we therefore set \( b_0 = b_1 = 0 \), we obtain, for arbitrary even \( m \), the homogeneous matrix difference equations

\[
B_0 z_{k-1} + (B_1 - \mu I) z_k + B_2 z_{k+1} = 0,
\]

\( k = 1, 2, 3, \ldots \frac{m}{2} \), where \( B_0 = \begin{bmatrix} Q^2 & QS \\ 0 & 0 \end{bmatrix}, \ B_1 = \begin{bmatrix} S^2 & SQ \\ SQ & S^2 \end{bmatrix}, \ B_2 = \begin{bmatrix} 0 & 0 \\ QS & Q^2 \end{bmatrix}, \ z_k = \begin{bmatrix} z_{k,0} \\ z_{k,1} \end{bmatrix} \) with boundary conditions

\[
b_0 = Q^2 (z_{0,0} + z_{1,1}) + QS (z_{0,1} + z_{1,0}) = 0
\]

\[
b_1 = Q^2 (z_{4,0} + z_{5,1}) + QS (z_{4,1} + z_{5,0}) = 0.
\]

It is clear that the problem of determining the eigenvalues of \( \hat{J} \) is equivalent to solving the boundary
value problem (2.9), (2.10), which we now proceed to do.

With respect to Theorem 8.3 in [56], we form the matrix polynomial \( \mathcal{L}(\lambda) \) that corresponds to (2.9), namely

\[
\mathcal{L}(\lambda) = B_2 \lambda^2 + (B_1 - \mu I) \lambda + B_0 = \begin{bmatrix}
(S^2 - \mu) \lambda + Q^2 & SQ \lambda + SQ \\
SQ \lambda^2 + SQ \lambda & Q^2 \lambda^2 + (S^2 - \mu) \lambda
\end{bmatrix}
\]  

(2.11)

and compute its determinant

\[
\det(\mathcal{L}(\lambda)) = \lambda \left\{ -Q^2 \mu \lambda^2 + \left[ (S^2 - Q^2)^2 - 2S^2 \mu + \mu^2 \right] \lambda - Q^2 \mu \right\}. 
\]  

(2.12)

Then (from Theorem 8.3 in [56]), the general solution of (2.9) is given by

\[
z_k = X_F J_F^k w,
\]  

(2.13)

where \((X_F, J_F)\) is a Jordan pair (see [56]) of the matrix polynomial \( \mathcal{L}(\lambda) \) in (2.11) and where \(w \in \mathbb{C}^n\), where \(n\) is the degree (in \(\lambda\)) of \(\det(\mathcal{L}(\lambda))\). The matrix \(J_F\) is the Jordan canonical form corresponding to the diagonal matrix whose diagonal entries are the zeros of (2.12).

We consider two separate cases: \(\mu = 0\) and \(\mu \neq 0\).

If \(\mu = 0\), then \(\mathcal{L}(\lambda) = \begin{bmatrix}
S^2 \lambda + Q^2 & SQ \lambda + SQ \\
SQ \lambda^2 + SQ \lambda & Q^2 \lambda^2 + S^2 \lambda
\end{bmatrix}\) and \(\det(\mathcal{L}(\lambda)) = (S^2 - Q^2)^2 \lambda^2\). Thus the only eigenvalue of \(\mathcal{L}(\lambda)\) i.e., zero of \(\det(\mathcal{L}(\lambda))\), is \(\lambda = 0\), which is a double eigenvalue. The Jordan chain (see [56]) associated with the eigenvalue \(\lambda = 0\) is seen to be of length two and it thus forms a canonical set (see [56]). Its components, using the definition in [56], are easily seen to be the columns of

\[
X_F = \begin{bmatrix}
1 & 1 \\
\frac{-Q}{S} & -\frac{s}{Q}
\end{bmatrix}
\]
while the matrix $J_F$ is
\[
J_F = \begin{bmatrix}
0 & 1 \\
0 & 0
\end{bmatrix},
\]
and $w = \begin{bmatrix} w_0 & w_1 \end{bmatrix}^T$. Applying (2.13) with $k = 0, 1, 2, \ldots$ and the definitions of $X_F$ and $J_F$ given above, we see that
\[
\begin{align*}
\mathbf{z}_0 &= \begin{bmatrix}
z_{0,0} \\
z_{0,1} \\
z_{1,0} \\
z_{1,1}
\end{bmatrix} = \begin{bmatrix}
w_0 + w_1 \\
- \frac{Q}{S} w_0 - \frac{S}{Q} w_1 \\
w_1 \\
- \frac{Q}{S} w_1
\end{bmatrix}, \\
\mathbf{z}_1 &= \begin{bmatrix}
z_{0,0} \\
z_{0,1} \\
z_{1,0} \\
z_{1,1}
\end{bmatrix} = \begin{bmatrix}
w_0 + w_1 \\
- \frac{Q}{S} w_0 - \frac{S}{Q} w_1 \\
w_1 \\
- \frac{Q}{S} w_1
\end{bmatrix}, \\
\mathbf{z}_k &= \mathbf{0}, \ k \geq 2.
\end{align*}
\]
Now, recall the boundary condition $b_0 = Q^2 (z_{0,0} + z_{1,1}) + QS (z_{0,1} + z_{1,0}) = 0$ from (2.10). Assuming that $S \neq \pm Q$ (a most reasonable assumption in light of (2.4) and (2.5)) and using the values of $z_{0,0}$, $z_{1,1}$, $z_{0,1}$, and $z_{1,0}$ from (2.14), we conclude that $w_1 = 0$. But then $\mathbf{z}_k = \mathbf{0}$ for all $k \geq 1$, which means that $\mathbf{z}$ in (2.6) is a zero eigenvector, which is impossible. Thus the case $\mu = 0$ is eliminated from consideration.

If $\mu \neq 0$, then $\det (L(\lambda))$ is given by (2.12). Setting $\det (L(\lambda)) = 0$ to determine the eigenvalues $\lambda$ of $L(\lambda)$ gives
\[
\lambda = \lambda_0 = 0 \text{ or } \lambda = \lambda_1 \text{ or } \lambda = \lambda_2
\]
where $\lambda_1$ and $\lambda_2$ are obtained from the quadratic formula. Using the well-known formulae for the sum and product of the roots of a quadratic equation, we obtain
\[
\lambda_1 + \lambda_2 = \frac{(S^2 - Q^2)^2 - 2S^2 \mu + \mu^2}{Q^2 \mu} \quad (2.15)
\]
and
\[
\lambda_1 \lambda_2 = 1. \quad (2.16)
\]
We note that the Jordan chain corresponding to the eigenvalue \( \lambda_0 = 0 \) is
\[
\begin{pmatrix}
1 \\
-\frac{Q}{S}
\end{pmatrix}
\]
or, equivalently,
\[
\begin{pmatrix}
Q \\
-S
\end{pmatrix}.
\]
We now consider two separate cases, namely \( \lambda_1 \neq \lambda_2 \) and \( \lambda_1 = \lambda_2 \).

If \( \lambda_1 \neq \lambda_2 \), then the Jordan chain corresponding to \( \lambda_i \) is seen to be
\[
\begin{pmatrix}
\omega_i \\
1
\end{pmatrix}, \quad i = 1, 2,
\]
where
\[
\omega_i = \frac{-SQ\lambda_i - SQ}{(S^2 - \mu) \lambda_i + Q^2}.
\]  

(2.17)

So we obtain, in this case
\[
X_F = \begin{pmatrix}
Q & \omega_1 & \omega_2 \\
-S & 1 & 1
\end{pmatrix},
\]
\[
J_F = \begin{pmatrix}
0 \\
\lambda_1 \\
\lambda_2
\end{pmatrix},
\]
and \( w = \begin{pmatrix} w_0 & w_1 & w_2 \end{pmatrix}^T \). We now find \( w \) to satisfy the boundary conditions (2.10). Using (2.13) for \( k = 0, 1, \frac{m}{2}, \frac{m}{2} + 1 \) together with (2.10), we obtain
\[
\begin{pmatrix}
Q (\omega_1 + \lambda_1) + S (1 + \omega_1 \lambda_1) \\
Q \left( \omega_1 \frac{m}{2} + \lambda_1 \frac{m}{2} + 1 \right) + S \left( \frac{m}{2} + \omega_1 \lambda_1 \frac{m}{2} + 1 \right)
\end{pmatrix}
\begin{pmatrix}
Q (\omega_2 + \lambda_2) + S (1 + \omega_2 \lambda_2) \\
Q \left( \omega_2 \frac{m}{2} + \lambda_2 \frac{m}{2} + 1 \right) + S \left( \frac{m}{2} + \omega_2 \lambda_2 \frac{m}{2} + 1 \right)
\end{pmatrix}
\begin{pmatrix}
w_1 \\
w_2
\end{pmatrix}
\]
\[
= \begin{pmatrix} 0 \\
0
\end{pmatrix}.
\]  

(2.18)

If the \( 2 \times 2 \) matrix in (2.18) is nonsingular, then \( w_1 = w_2 = 0 \), which implies (from (2.13)) that \( z_k = 0 \) for all \( k \geq 1 \). Thus \( z \) in (2.6) is a zero eigenvector, which is impossible. Therefore, the \( 2 \times 2 \)
matrix in (2.18) is singular so its determinant is zero, which leads to the equation

\[
\left( \lambda_2^\Psi - \lambda_1^\Psi \right) \left[ Q (\omega_1 + \lambda_1) + S (1 + \omega_1 \lambda_1) \right] \left[ Q (\omega_2 + \lambda_2) + S (1 + \omega_2 \lambda_2) \right] = 0. \tag{2.19}
\]

If \( Q (\omega_i + \lambda_i) + S (1 + \omega_i \lambda_i) = 0, \ i = 1, 2 \), then using (2.17) and solving for \( \lambda_i \) yields

\[
\lambda_i = \frac{(S + Q)(S - Q)^2 - S\mu}{Q\mu}, \tag{2.20}
\]

\( i = 1, 2 \). Since \( \lambda_i \) (which is non-zero) is a solution of \( \det (\mathcal{L}(\lambda)) = 0 \), we can conclude from (2.12) that

\[
-Q^2\mu \lambda_i^2 + \left[ (S^2 - Q^2)^2 - 2S^2\mu + \mu^2 \right] \lambda_i - Q^2\mu = 0. \tag{2.21}
\]

Substitution of (2.20) into (2.21) yields a cubic equation in \( \mu \), whose solutions are

\[
\mu = (S - Q)(S + Q) \text{ or } \mu = (S - Q)^2,
\]

where the former solution is of multiplicity two.

If \( \mu = (S - Q)^2 \), then (2.20) reduces to \( \lambda_1 = \lambda_2 = 1 \), which contradicts the assumption \( \lambda_1 \neq \lambda_2 \).

If, on the other hand, \( \mu = (S - Q)(S + Q) \), then (2.20) reduces to \( \lambda_1 = \lambda_2 = -1 \), which also contradicts the assumption \( \lambda_1 \neq \lambda_2 \). Thus, with respect to (2.19), we obtain

\[
\left( \lambda_2^\Psi - \lambda_1^\Psi \right) = 0.
\]

Recalling (2.16) and the assumption \( \lambda_1 \neq \lambda_2 \), we conclude

\[
\lambda_1 = e^{i\theta} \tag{2.22}
\]

and

\[
\lambda_2 = e^{-i\theta} \tag{2.23}
\]
where $\theta = \frac{2k\pi}{m}$, $k = 1, 2, 3, \ldots, \frac{m}{2} - 1$.

If we now add the equation obtained by substituting (2.22) into (2.21) to the equation obtained by substituting (2.23) into (2.21), we obtain a quadratic equation in $\mu$ whose coefficients are all real. Solving this equation for $\mu$ yields

$$
\mu = (Q^2 \cos \theta + S^2) \pm Q \sqrt{2S^2 (1 + \cos \theta) - (Q \sin \theta)^2}, \quad (2.24)
$$

$\theta = \frac{2k\pi}{m}$, $k = 1, 2, 3, \ldots, \frac{m}{2} - 1$.

What we have shown so far is this. If $S$ and $Q$ are scalars defining the $m \times m$ matrix $\hat{J}$ in (2.3), then $m - 2$ of the $m$ eigenvalues of $\hat{J}$ are given by (2.24). The remaining two eigenvalues of $\hat{J}$ arise from the case $\lambda_1 = \lambda_2$ and will be determined below.

If $\lambda_1 = \lambda_2$, then (2.15) becomes

$$
2\lambda_1 Q^2 \mu = (Q^2 - S^2)^2 - 2S^2 \mu + \mu^2. \quad (2.25)
$$

By (2.16), we must have $\lambda_1 = \lambda_2 = 1$ or $\lambda_1 = \lambda_2 = -1$.

If $\lambda_1 = \lambda_2 = 1$, then solving (2.25) for $\mu$ yields

$$
\mu = (S \pm Q)^2.
$$

We now consider these two cases separately.

If $\mu = (S + Q)^2$, then, using the definition of Jordan chain in [56], we obtain

$$
X_F = \begin{bmatrix}
S & 1 & -\frac{1}{2} \\
-\frac{1}{2} & 1 & \frac{Q}{2S} \\
-\frac{1}{2} & \frac{Q}{2S} & 1
\end{bmatrix},
$$

$$
J_F = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{bmatrix}.
$$
and \( \mathbf{w} = \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix}^T \). Using (2.13) with these values and (2.10), we obtain

\[
\begin{bmatrix}
4S & Q + 1 \\
2 & 2m + \frac{1}{2} + \frac{Q}{2S}
\end{bmatrix}
\begin{bmatrix}
w_1 \\
w_2
\end{bmatrix} = \begin{bmatrix} 0 \\
0
\end{bmatrix}.
\]

Unless \( S = \frac{1}{4m-1} \), this \( 2 \times 2 \) matrix is nonsingular, which implies \( w_1 = w_2 = 0 \). Using (2.13), we see that \( z_k = 0 \) for all \( k \geq 1 \) which implies that \( z \) is a zero eigenvector, which is impossible. We thus eliminate the possibility \( \mu = (S + Q)^2 \).

If \( \mu = (S - Q)^2 \), then, again using the definition of Jordan chain from [56], we obtain

\[
X_F = \begin{bmatrix}
S & 1 & -\frac{1}{2} \\
-Q & -1 & \frac{Q}{2S}
\end{bmatrix},
\]

\[
J_F = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{bmatrix},
\]

and \( \mathbf{w} = \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix}^T \). Using (2.13) with these values and (2.10), we find that \( w_2 = 0 \) and that \( w_1 \neq 0 \) is arbitrary. We conclude that \( \lambda = 1 \) provides the eigenvalue \( \mu = (S - Q)^2 \) of \( \tilde{J} \) with corresponding eigenvector \( \begin{bmatrix} 1 & -1 & 1 & -1 & \cdots & 1 & -1 \end{bmatrix}^T \).

If \( \lambda_1 = \lambda_2 = -1 \), then solving (2.25) for \( \mu \) yields the solution of multiplicity two

\[
\mu = (S - Q)(S + Q).
\]

Once more using the definition of Jordan chain in [56], we obtain for this case,

\[
X_F = \begin{bmatrix}
S & 1 & 0 \\
-Q & 0 & 1
\end{bmatrix}.
\]
\[
J_F = \begin{bmatrix}
0 & 0 & 0 \\
0 & -1 & 0 \\
0 & 0 & -1
\end{bmatrix},
\]

and \( w = \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix}^T \). Using (2.13) with these values and (2.10), we find that \( w_1 = w_2 \) is arbitrary but must be non-zero. We conclude that \( \lambda = -1 \) provides the eigenvalue \( \mu = (S - Q)(S + Q) \) of \( \hat{J} \) with corresponding eigenvector \( \begin{bmatrix} -1 & -1 & 1 & 1 & \cdots & -1 & -1 & 1 & 1 \end{bmatrix}^T \).

We have therefore proved

**Lemma 2.1** Let \( \hat{J} \) be the \( m \times m \) matrix defined in (2.3) by the real scalars \( S \) and \( Q \). Then the eigenvalues \( \mu \) of \( \hat{J} \) are given by

\[
\mu = S^2 - Q^2
\]

\[
\mu = (S - Q)^2
\]

\[
\mu = (Q^2 \cos \theta + S^2) \pm Q \sqrt{2S^2 (1 + \cos \theta) - (Q \sin \theta)^2},
\]

where \( \theta = \frac{2k\pi}{m}, k = 1, 2, 3, \ldots, \frac{m}{2} - 1. \)

Before we consider the general case where \( S \) and \( Q \) are matrices, we require another lemma:

**Lemma 2.2** Let \( S \) and \( Q \) be as defined in (2.4) and (2.5), respectively. Then \( SQ = QS \).

**Proof.** Using Lemma 5.2 in [86], we are given the existence of a nonsingular matrix \( X \) and explicit diagonal matrices \( D \) and \( \overline{D} \) such that

\[
X^T (A_4 A_7^{-1}) X^{-T} = D \quad (2.26)
\]

and

\[
X^T (A_3 A_1^{-1}) X^{-T} = \overline{D}. \quad (2.27)
\]

Thus \( A_4 A_7^{-1} \) and \( A_3 A_1^{-1} \) have the same complete set of eigenvectors and must therefore commute.
But this is precisely the condition that is required to show that $SQ = QS$. Q.E.D.

Now that we have established that $S$ and $Q$ commute, we see that we can use the analysis culminating in Lemma 2.1 to determine the eigenvalues of $\hat{J}$ for the case where $S$ and $Q$ are defined as in (2.4) and (2.5). Let us begin by recalling (2.11):

$$\mathcal{L}(\lambda) = \begin{bmatrix} (S^2 - \mu I) \lambda + Q^2 & SQ \lambda + SQ \\ SQ \lambda^2 + SQ \lambda & Q^2 \lambda^2 + (S^2 - \mu I) \lambda \end{bmatrix},$$

where we are making use of the fact that $S$ and $Q$ are commuting matrices. As above, let us now consider

$$0 = \det(\mathcal{L}(\lambda)) = \det \begin{bmatrix} (S^2 - \mu I) \lambda + Q^2 & SQ \lambda + SQ \\ SQ \lambda^2 + SQ \lambda & Q^2 \lambda^2 + (S^2 - \mu I) \lambda \end{bmatrix}.$$  \hspace{1cm} (2.29)

From our work above, we know the solutions $\lambda$ of (2.29), namely $\lambda = 0$, $\lambda = \pm 1$, and $\lambda = e^{\pm i \theta_k}$, $\theta_k = 1, 2, \ldots \frac{m}{2} - 1$. We now exploit this knowledge to compute the eigenvalues $\mu$ of $\hat{J}$.

If we use the values of $\lambda = e^{i \theta_k}$, $\theta_k = \frac{2k\pi}{m}$, $k = 1, 2, \ldots \frac{m}{2} - 1$ in (2.29), we can easily show that

$$0 = \det \begin{bmatrix} S^2 + \frac{1}{\lambda} Q^2 - \mu I & SQ + \frac{1}{\lambda} SQ \\ SQ + \lambda SQ & S^2 + \lambda Q^2 - \mu I \end{bmatrix}. \hspace{1cm} (2.30)$$

Finding the values $\mu$ that satisfy (2.30) is equivalent to finding the eigenvalues $\mu$ of the matrix

$$M_k = \begin{bmatrix} S^2 + \frac{1}{\lambda} Q^2 & SQ + \frac{1}{\lambda} SQ \\ SQ + \lambda SQ & S^2 + \lambda Q^2 \end{bmatrix}.$$ 

To eliminate the complex numbers (i.e., the $\lambda$'s), we perform the similarity transformation $T_k =$
\[ R_k M_k R_k^{-1}, \]

where
\[
R_k = 
\begin{bmatrix}
-i (Q^2 - SQ) \sin \theta_k & i (Q^2 - SQ) \sin \theta_k \\
(Q^2 + SQ) \sin \theta_k & (Q^2 + SQ) \sin \theta_k
\end{bmatrix}.
\]

We thus seek the eigenvalues \( \mu \) of
\[
T_k = 
\begin{bmatrix}
(S - Q)(S - Q \cos \theta_k) & (S - Q) Q \sin \theta_k \\
(S + Q) Q \sin \theta_k & (S + Q) (S + Q \cos \theta_k)
\end{bmatrix}.
\]

\( \theta_k = \frac{2k\pi}{m}, \quad k = 1, 2, \ldots, \frac{m}{2} - 1. \)

The characterization of all the eigenvalues \( \mu \) of \( \tilde{J} \) is found in the following lemma, the proof of which is, except for the obvious notational differences, analogous to that given in Lemma 4.2 in [86]:

**Lemma 2.3** Let \( \tilde{J} \) be the \( 2m^2 \times 2m^2 \) matrix defined in (2.3) by the matrices \( S \) and \( Q \) defined in (2.4) and (2.5). Then \( \sigma(\tilde{J}) \) is given by
\[
\sigma(\tilde{J}) = \bigcup_{k=1}^{\frac{m}{2}-1} \sigma(T_k) \bigcup \sigma(S^2 - Q^2) \bigcup \sigma((S - Q)^2).
\]

Now that we have characterized the spectrum of \( \tilde{J} \) as the union of spectra of other matrices, we now determine these latter spectra, namely for \( T_k, \quad k = 1, 2, \ldots, \frac{m}{2} - 1; \quad S^2 - Q^2; \) and \( (S - Q)^2 \).

We first make some observations and definitions. Let \( A_{31} = A_3 A_1^{-1} \) and let \( A_{42} = A_4 A_2^{-1} \). With respect to (2.4) and (2.5), we see that
\[
S + Q = -A_{31}
\]
and
\[
S - Q = -A_{42}.
\]
Using these definitions, trigonometric identities, and (2.28), we can show
\[
T_k = \begin{bmatrix}
\sin^2 \frac{\theta_k}{2} A_{42} A_{31} + \cos^2 \frac{\theta_k}{2} A_{42}^2 \\
\sin \frac{\theta_k}{2} \left( A_{42}^2 - A_{42} A_{31} \right) \\
\sin^2 \frac{\theta_k}{2} A_{42} A_{31} + \cos^2 \frac{\theta_k}{2} A_{42}^2
\end{bmatrix}
\]

We now use (2.26) and (2.27) to compute the similarity transformation
\[
\begin{bmatrix}
X^T \\
X^T \\
X^T
\end{bmatrix}
T_k
\begin{bmatrix}
X^T \\
X^T \\
X^T
\end{bmatrix}
= \begin{bmatrix}
\sin^2 \frac{\theta_k}{2} D D^T + \cos^2 \frac{\theta_k}{2} D^2 \\
\sin \frac{\theta_k}{2} \left( D^T - D \right) \\
\sin^2 \frac{\theta_k}{2} D D^T + \cos^2 \frac{\theta_k}{2} D^2
\end{bmatrix}
\]

where \(D\) and \(D\) are given explicitly in Lemma 5.2 of [86]. We write \(D = \text{diag}(d_0, d_1, \ldots, d_{2m-1})\) and \(D = \text{diag}(\overline{d}_0, \overline{d}_1, \ldots, \overline{d}_{2m-1})\). If we then permute the rows and columns of (2.32) in an obvious way, we find that \(T_k\) is similar to \(\text{diag}(D_{k,0}, D_{k,1}, \ldots, D_{k,2m-1})\), where
\[
D_{k,i} = \begin{bmatrix}
d_i \left( \overline{d}_i \sin^2 \frac{\theta_k}{2} + d_i \cos^2 \frac{\theta_k}{2} \right) & d_i \sin \frac{\theta_k}{2} \left( \overline{d}_i - d_i \right) \\
\overline{d}_i \sin \frac{\theta_k}{2} \left( \overline{d}_i - d_i \right) & \overline{d}_i \left( \overline{d}_i \sin^2 \frac{\theta_k}{2} + d_i \cos^2 \frac{\theta_k}{2} \right)
\end{bmatrix}
\]

Since each \(D_{k,i}\) is a \(2 \times 2\) matrix, the eigenvalues of each are easily determined:
\[
\sigma(D_{k,i}) = \left\{ \mu : \mu = \frac{z_{k,i}^2 + 2 \overline{d}_i d_i \pm \sqrt{z_{k,i}^2 + 4 \overline{d}_i d_i}}{2} \right\}
\]

where \(z_{k,i} = (\overline{d}_i - d_i) \cos \frac{\theta_k}{2} \), \(\theta_k = \frac{2 \pi}{m} k, \ k = 1, 2, \ldots, \frac{m}{2} - 1, \ i = 0, 1, \ldots, 2m - 1\).

To find \(\sigma(S^2 - Q^2)\), we note that \(S^2 - Q^2 = A_{42} A_{31}\) and \(X^T A_{42} A_{31} X - T = TD\), which is a diagonal matrix. Therefore, \(\sigma(S^2 - Q^2) = \{ \overline{d}_i d_i \}_{i=0}^{2m-1} \).

To find \(\sigma((S - Q)^2)\), we note that \((S - Q)^2 = A_{42}^2\), which is similar to \(D^2\), which is a diagonal matrix. Therefore, \(\sigma((S - Q)^2) = \{ d_i^2 \}_{i=0}^{2m-1} \).

We may now state:

**Lemma 2.4** Let \( \hat{J} \) be the \(2m^2 \times 2m^2\) matrix defined in (2.3) with \(S\) and \(Q\) defined as in (2.4) and
(2.5), respectively. Let $\xi = \frac{1}{\sqrt{12}}$ in (2.2). Then

$$\sigma (\tilde{J}) = \{ \mu : \mu = d_i^2 \} \cup \{ \mu : \mu = \overline{d}_i d_i \} \cup \left\{ \mu : \mu = \frac{z_{k,i}^2 + 2\overline{d}_i d_i \pm \sqrt{z_{k,i}^2 \left( z_{k,i}^2 + 4\overline{d}_i d_i \right)}}{2} \right\},$$

where $z_{k,i} = (\overline{d}_i - d_i) \cos \theta_k$, $\theta_k = \frac{i \pi}{m}$, $k = 1, 2, \ldots \frac{m}{2} - 1$, $i = 0, 1, \ldots, 2m - 1$, and where [86]

$$\{d_i\}_{i=0}^{2m-1} = \left\{ \alpha_0^-, \alpha_1^+, \alpha_1^-, \ldots, \alpha_{m-1}^+, \alpha_{m-1}^-, \alpha_m^- \right\},$$

$$\{\overline{d}_i\}_{i=0}^{2m-1} = \left\{ \beta_0^-, \beta_1^+, \beta_1^-, \ldots, \beta_{m-1}^+, \beta_{m-1}^-, \beta_m^- \right\},$$

$$\alpha_j^\pm = \frac{3\sqrt{3} \pm q_j}{-28 - 16\sqrt{3} + (\sqrt{3} + 1) \cos \varphi_j}, \quad \beta_j^\pm = \frac{(37 + 8 \cos \varphi_j) \pm q_j}{-64 - 36\sqrt{3} + (19 + 9 \sqrt{3} \cos \varphi_j)}, \quad q_j = \sqrt{43 + 40 \cos \varphi_j - 2 \cos^2 \varphi_j},$$

$\varphi_j = \frac{\varpi_j}{m}$, $j = 0, 1, \ldots, m$.

2.2.2 The eigenvalues of $\tilde{J}$ for the case of general $\xi \in (0, \frac{1}{3})$

The result in Lemma 2.4 is for the case $\xi = \frac{1}{\sqrt{12}}$. If we seek an analogous result for arbitrary $\xi \in (0, \frac{1}{3})$, we note that we must consider only three things. The first (and obvious) one is that the entries of matrix $A$ are now given by (2.2) for arbitrary $\xi$ (as opposed to the specific value $\xi = \frac{1}{\sqrt{12}}$). The second is to check whether we obtain the result $w_1 = \overline{w}_1$, where $w_1$ and $\overline{w}_1$ are given in Lemmas 5.1 and 5.2 in [86], which a long and tedious calculation does indeed confirm. The third is to determine how $\alpha_j^\pm$ associates with $\beta_j^\pm$ (see Lemma 5.2 in [86]). In this case, another long and tedious calculation provides that $\alpha_j^+$ associates with $\beta_j^-$ and that $\alpha_j^-$ associates with $\beta_j^+$ for all values of $\xi \in (0, \frac{1}{3})$. We can therefore state, for arbitrary $\xi \in (0, \frac{1}{3})$, the result analogous to Lemma 2.4:

**Theorem 2.5** Let $\tilde{J}$ be the $2m^2 \times 2m^2$ matrix defined in (2.3) with $S$ and $Q$ defined as in (2.4) and (2.5), respectively. Let $\xi$ in (2.2) be an arbitrary number in the interval $(0, \frac{1}{3})$. Then

$$\sigma (\tilde{J}) = \{ \mu : \mu = d_i^2 \} \cup \{ \mu : \mu = \overline{d}_i d_i \} \cup \left\{ \mu : \mu = \frac{z_{k,i}^2 + 2\overline{d}_i d_i \pm \sqrt{z_{k,i}^2 \left( z_{k,i}^2 + 4\overline{d}_i d_i \right)}}{2} \right\};$$
where $z_{k,i} = (d_i - d_k) \cos \theta_k$; $\theta_k = \frac{k\pi}{m}$; $k = 1, 2, \ldots, \frac{m}{2} - 1$; $i = 0, 1, \ldots, 2m - 1$; $c_j = \cos \theta_j$; $j = 0, 1, \ldots, m$;

$$d_i = \left\{ \alpha_0^+, \alpha_1^+, \alpha_1^-, \ldots, \alpha_{m-1}^+, \alpha_{m-1}^- \right\};$$

$$a_i = \left\{ \beta_0^-, \beta_1^-, \beta_1^+, \ldots, \beta_{m-1}^-, \beta_{m-1}^+ \right\};$$

$q_j = 2\xi \sqrt{(16\xi^4 - 24\xi^2 + 21) + c_j (-32\xi^4 + 18) + c_j^2 (16\xi^4 + 24\xi^2 - 3)}$;

\[
\alpha_j^\pm = \frac{96\xi^4 - 92\xi^2 + 5 + c_j (1 - 12\xi^2) (-1 + 8\xi^2) \pm q_j}{-96\xi^4 - 66\xi^3 + 76\xi^2 + 42\xi + 5 + c_j (1 - 2\xi)(1 + 4\xi - 4\xi^2 - 48\xi^3)};
\]

\[
\beta_j^\pm = \frac{128\xi^6 - 192\xi^4 + 60\xi^2 - 1 + c_j (-128\xi^6 + 96\xi^4 - 12\xi^2 + 1) \pm q_j}{(1 + 2\xi)(1 - 2\xi)(1 - 16\xi - 28\xi^2 + 80\xi^3 + 32\xi^4 - 64\xi^5) + c_j (1 - 2\xi)(1 + 2\xi)^2 (1 + 4\xi + 8\xi^2 - 16\xi^3)}.
\]

### 2.3 A model parabolic equation

We now consider solving the parabolic PDE

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - H(x, y, t),$$

(2.33)

with Dirichlet boundary conditions on $\partial S$ and appropriate initial conditions. We approximate the time derivative by

$$\frac{\partial u}{\partial t} = \frac{u^{(n+1)} - u^{(n)}}{\Delta t},$$

(2.34)

where the superscript $(n)$ indicates the value of $u$ after $n$ time steps.

Recalling (1.11), (1.13), and the two-dimensional analogue of (1.8), we see that matrix $A$ (whose entries are given in (2.2)) was formed by evaluating $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ at the collocation points. Correspondingly, we form the matrix $C$ by evaluating $\hat{u}$ at the collocation points. We find that matrix $C$ has exactly the same structure as matrix $A$, where the entries of $C$ are given below. Note the
symmetry \( c_{i,j} = c_{j,i} \).

\[
\begin{align*}
  c_{1,1} &= \frac{(1-\xi)^2(1+2\xi)^4}{4} \\
  c_{1,2} &= \frac{(1-2\xi)(1-\xi)(1+2\xi)^4}{16} \\
  c_{1,3} &= \frac{(1-2\xi)^2(1-\xi)(1+\xi)(1+2\xi)^2}{4} \\
  c_{1,4} &= \frac{(1-2\xi)^2(1-\xi)(1+2\xi)^3}{16} \\
  c_{2,2} &= \frac{(1-2\xi)^2(1+2\xi)^4}{64} \\
  c_{2,3} &= \frac{(1-2\xi)^2(1+\xi)(1+2\xi)^2}{16} \\
  c_{2,4} &= \frac{(1+2\xi)^2(1-2\xi)^3}{64} \\
  c_{3,3} &= \frac{(1-2\xi)^3(1+\xi)^2}{4} \\
  c_{3,4} &= \frac{(1+2\xi)(1+\xi)(1-2\xi)^3}{16} \\
  c_{4,4} &= \frac{(1+2\xi)^2(1-2\xi)^4}{64}
\end{align*}
\] (2.35)

If we now introduce (2.34) and the interpolating polynomial (1.11)\(^1\) into (2.33) and evaluate the right side of (2.33) at the collocation points at time \( \theta t^{(n+1)} + (1-\theta) t^{(n)} \), where \( 0 \leq \theta \leq 1 \), then we obtain the matrix form of the collocation discretization of the parabolic PDE (2.33):

\[
\frac{C x^{(n+1)} - C x^{(n)}}{\Delta t} = \theta \left[ A x^{(n+1)} - b^{(n+1)} \right] + (1-\theta) \left[ A x^{(n)} - b^{(n)} \right].
\] (2.36)

Letting \( \tau = \theta \Delta t \) and \( \tau = (1-\theta) \Delta t \), we may express (2.36) as

\[
(C - \tau A) x^{(n+1)} = (C + \tau A) x^{(n)} - \left( \tau b^{(n+1)} + \tau b^{(n)} \right).
\] (2.37)

In examining (2.37), we see that this equation defines how we may move from time step \( (n) \) to time step \( (n+1) \), because, at time step \( (n) \), all the vectors on the right side of (2.37) contain known values. We may therefore apply to (2.37) the preconditioned Bi-CGSTAB algorithm that we developed for (1.13). That is, at each time step in (2.37), we iterate to convergence using our preconditioned Bi-CGSTAB algorithm.

\(^1\) The interpolating polynomial (1.11) and forcing function now have time dependence, i.e., \( u_{q,r}, u_{q,r}^x, u_{q,r}^y, u_{q,r}^{q_n} \) and \( H \) are now functions also of \( t \).
We now apply the analysis used to determine the eigenvalues associated with Poisson’s equation (2.1) to the eigenvalues associated with the model parabolic equation (2.33). The result is:

**Theorem 2.6** By analogy to Theorem 2.5, the eigenvalues for the preconditioned Bi-CGSTAB method corresponding to solving the model parabolic equation (2.33) are given by the same recipe as in Theorem 2.5, with the exceptions of \( \alpha_j^\pm \) and \( \beta_j^\pm \), which are defined for this case as follows.

We first define \( \nu = \frac{c}{\pi^2} \).

Let \( a_{\alpha_j} = [(5 - 64\xi^2 + 288\xi^4 - 512\xi^6 + 256\xi^8) + c_j (-1 + 16\xi^2 - 96\xi^4 + 256\xi^6 - 256\xi^8)] \)
+ \([(176 - 2240\xi^2 + 7424\xi^4 - 5120\xi^6) + c_j (-16 + 448\xi^2 - 2816\xi^4 + 5120\xi^6)] \nu \)
+ \([(1280 - 23552\xi^2 + 24576\xi^4) + c_j (-256 + 5120\xi^2 - 24576\xi^4)] \nu^2 \).

Let \( b_{\alpha_j} = [(5 + 20\xi - 24\xi^2 - 176\xi^3 - 64\xi^4 + 448\xi^5 + 384\xi^6 - 256\xi^7 - 256\xi^8) + \]
\(+ c_j (-1 - 4\xi + 8\xi^2 + 48\xi^3 - 192\xi^5 - 128\xi^6 + 256\xi^7 + 256\xi^8)] \)
+ \([(176 + 1024\xi + 448\xi^2 - 5120\xi^3 - 5888\xi^4 + 4096\xi^5 + 5120\xi^6) + \]
\(+ c_j (-16 - 128\xi + 64\xi^2 + 1536\xi^3 + 1280\xi^4 - 4096\xi^5 - 5120\xi^6)] \nu \)
\+ \([(1280 + 10752\xi + 19456\xi^2 - 14336\xi^3 - 24576\xi^4) + \]
\(+ c_j (-256 - 1536\xi - 1024\xi^2 + 14336\xi^3 + 24576\xi^4)] \nu^2 \).

Let \( a_{\beta_j} = [(-5 + 69\xi^2 - 352\xi^4 + 800\xi^6 - 768\xi^8 + 256\xi^{10}) + \]
\(+ c_j (1 - 17\xi^2 + 112\xi^4 - 352\xi^6 + 512\xi^8 - 256\xi^{10})] \)
+ \([(96 + 1712\xi^2 - 7872\xi^4 + 11520\xi^6 - 5120\xi^8) + c_j (-272\xi^2 + 2496\xi^4 - 6912\xi^6 + 5120\xi^8)] \nu \)
+ \([(192 + 11520\xi^2 - 3664\xi^4 + 24576\xi^6) + c_j (192 - 2304\xi^2 + 18432\xi^4 - 24576\xi^6)] \nu^2 \).

Let \( b_{\beta_j} = [(-5 - 30\xi - 21\xi^2 + 184\xi^3 + 320\xi^4 - 288\xi^5 - 800\xi^6 + 128\xi^7 + 768\xi^8 - 256\xi^{10}) + \]
\(+ c_j (1 + 6\xi + \xi^2 - 56\xi^3 - 80\xi^4 + 160\xi^5 + 352\xi^6 - 128\xi^7 - 512\xi^8 + 256\xi^{10})] \)
+ \([(96 - 816\xi - 1456\xi^2 + 2688\xi^3 + 7872\xi^4 - 1792\xi^5 - 11520\xi^6 + 5120\xi^8) + \]
\(+ c_j (48\xi + 16\xi^2 - 1152\xi^3 - 2496\xi^4 + 1792\xi^5 + 6912\xi^6 - 5120\xi^8)] \nu \)
+ \([(192 - 3456\xi - 11520\xi^2 + 4608\xi^3 + 3686\xi^4 - 24576\xi^6) + \]
\(+ c_j (192 + 1152\xi + 2304\xi^2 - 4608\xi^3 - 18432\xi^4 + 24576\xi^6)] \nu^2 \).

Then \( \alpha_j^\pm = \frac{a_{\alpha_j} \pm 256^j \nu^2}{b_{\alpha_j}} \) and \( \beta_j^\pm = \frac{a_{\beta_j} \pm 192^j \nu^2}{b_{\beta_j}} \).
Chapter 3

Shifting the Collocation Points to Minimize $\|\sigma (I - P^{-1}A)\|_2$

In this chapter we investigate the possibility of increasing the speed at which the Bi-CGSTAB algorithm converges by using values for $\xi$ other than $\frac{1}{\sqrt{12}}$, acknowledging that any increase in convergence rate will come at the expense of losing the $O(h^4)$ accuracy provided by the Gaussian value $\xi = \frac{1}{\sqrt{12}}$.

Recall the discussion above where we stated that we expect the fastest convergence of the preconditioned Bi-CGSTAB algorithm to occur when the eigenvalues are clustered about the origin of the complex plane. In order to measure the clustering, we consider the vector norm $\|\sigma (I - P^{-1}A)\|_2 = \|\sigma (\tilde{J})\|_2$. In light of the geometric interpretation of the 2-norm, we may say that optimal clustering about the origin of the complex plane is equivalent to minimizing $\|\sigma (\tilde{J})\|_2$ as a function of $\xi$, the parameter which controls collocation point location.

In the literature (e.g. [57] and [113]), the condition number $\kappa_\gamma(P^{-1}A)$ is often used as a measure of how close $P^{-1}A$ is to the identity matrix $I$, with rapid convergence occurring when $P^{-1}A \approx I$.

The condition number is defined

$$
\kappa_\gamma (M) = \|M\|_\gamma \|M^{-1}\|_\gamma,
$$
where $\gamma$ is a given matrix norm. The four most common matrix norms correspond to $\gamma = 1, 2, \infty, F$ (see [57] for the definitions of these norms). Because these matrix norms satisfy the consistency [136] or submultiplicative [57] property, the minimum value that $\kappa_\gamma (M)$ can attain is unity (which occurs when $M = I$). Thus if we were able to achieve $P = A$, then we would have $\| \sigma (\hat{J}) \|_2 = 0$ (the minimum value a vector norm can attain) and $\kappa_\gamma (P^{-1}A) = 1$ (the minimum value a condition number can attain).

We are thus motivated to examine the relationships between the value of $\xi$ that produces the fastest convergence of Bi-CGSTAB, the value of $\xi$ that minimizes $\| \sigma (\hat{J}) \|_2$ and the value of $\xi$ that minimizes each of the four condition numbers.

### 3.1 Poisson’s equation

In Figure 3.1, we summarize the results obtained for Poisson’s equation (2.1) for the case $m = 10$. Convergence is defined by $\| r_i \|_2 < 10^{-8}$ (see (1.15)). For $\xi = 0.1, 0.2, \ldots, 0.49$, we solved Poisson’s equation using Bi-CGSTAB, computed $\| \sigma (\hat{J}) \|_2 = \| \sigma (I - P^{-1}A) \|_2$ using the eigenvalue formulae derived above, and computed the condition numbers using Matlab. With reference to Figure 3.1, we see that the general behavior in all six graphs is similar: i.e., as $\xi$ increases, the curves rapidly decrease to a minimum, then increase gradually. Upon more careful examination, we see that the value of $\xi$ that minimizes each of $\| \sigma (I - P^{-1}A) \|_2$, $\kappa_F (P^{-1}A)$, and $\kappa_\infty (P^{-1}A)$ corresponds well to the value of $\xi$ that minimizes the number of iterations required for convergence of Bi-CGSTAB. The condition numbers using the 1- and 2-norms are less effective in predicting the value of $\xi$ that produces the fastest convergence of Bi-CGSTAB.

In Figure 3.2, we repeat our study for $m = 20$. The results are similar, except that now the condition number using the $\infty$-norm has fallen into disrepute, leaving only $\| \sigma (I - P^{-1}A) \|_2$ and $\kappa_F (P^{-1}A)$ as effective predictors for the value of $\xi$ that produces optimal convergence of Bi-CGSTAB.

It is worth noting that computing $\| \sigma (I - P^{-1}A) \|_2$ is easy, since we have formulas for the eigenvalues of $\hat{J}$. Furthermore, finding the value of $\xi$ that minimizes $\| \sigma (I - P^{-1}A) \|_2$ for any given
value of $m$ is also easy if we use the MINOS optimization software (see [99] for documentation).

Using MINOS, we determined the value of $\xi$ that minimizes $\|\sigma (I - P^{-1}A)\|_2$ for Poisson’s equation for all even $m$ between 2 and 100, inclusive. The results are given in Figure 3.3.

In examining Figure 3.3, we see that the optimal value of $\xi$ is relatively insensitive to problem size $m$ for sufficiently large $m$. This is an important result, for it means that a separate optimization for each value of $m$ is not necessary each time we wish to solve Poisson’s equation as quickly as possible. A qualitative explanation for the insensitivity of optimal $\xi$ with respect to $m$ follows.

With respect to Theorem 2.5, we see that there are eight “flavors” of eigenvalues. Four of them are $(\alpha^+)^2$, $(\alpha^-)^2$, $\alpha^+\beta^-$, and $\alpha^-\beta^+$. The remaining four are of the form

$$\frac{1}{2} \left( z^2 + 2\varepsilon l \pm \sqrt{z^2 (z^2 + 4\varepsilon l)} \right).$$
This represents four additional flavors since \( z \) may be formed by combining \( \alpha^+ \) and \( \beta^- \) or by combining \( \alpha^- \) and \( \beta^+ \) and, for each form of \( z \), we either add or subtract the radical. To summarize:

<table>
<thead>
<tr>
<th>flavor number</th>
<th>characterization</th>
<th>number of eigenvalues per flavor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((\alpha^+)^2)</td>
<td>(m + 1)</td>
</tr>
<tr>
<td>2</td>
<td>((\alpha^-)^2)</td>
<td>(m - 1)</td>
</tr>
<tr>
<td>3</td>
<td>(\alpha^+\beta^-)</td>
<td>(m + 1)</td>
</tr>
<tr>
<td>4</td>
<td>(\alpha^-\beta^+)</td>
<td>(m - 1)</td>
</tr>
<tr>
<td>5</td>
<td>(z ) with (\alpha^+, \beta^-) and (+)</td>
<td>((m + 1) (\frac{m}{2} - 1))</td>
</tr>
<tr>
<td>6</td>
<td>(z ) with (\alpha^+, \beta^-) and (-)</td>
<td>((m + 1) (\frac{m}{2} - 1))</td>
</tr>
<tr>
<td>7</td>
<td>(z ) with (\alpha^-, \beta^+) and (+)</td>
<td>((m - 1) (\frac{m}{2} - 1))</td>
</tr>
<tr>
<td>8</td>
<td>(z ) with (\alpha^-, \beta^+) and (-)</td>
<td>((m - 1) (\frac{m}{2} - 1))</td>
</tr>
</tbody>
</table>
Figures 3.4 through 3.11 show the eigenvalues associated with each of the eight flavors for $\xi = 0.01$, 0.05, 0.10, 0.15, 0.20, and 0.40 and $m = 10$. The fact that we use the specific value $m = 10$ should not obfuscate what occurs in the general case, for the eigenvalues are merely values of continuous functions (i.e., the eight flavors) evaluated at the discrete points defined by $\theta_j$ ($j = 0, 1, \ldots, m$) and $\theta_k$ ($k = 1, 2, \ldots, \frac{m}{2} - 1$) where $\theta_i = \frac{i}{m}$, $i = j$ or $k$ (see Theorem 2.5).

For very small $\xi$, all flavors have a large real part. As $\xi$ increases to 0.15, all eigenvalues associated with all flavors cluster tightly around the origin, with the exception of those of flavor 7, which has a few large real eigenvalues. As $\xi$ increases further, we see that the eigenvalues of flavors 1, 2, 3, 5, and 6 remain tightly clustered about the origin, while eigenvalues of flavor 7 gain more outliers and eigenvalues of flavors 4 and 8 disperse significantly. This behavior is clearly not a function of $m$, but rather of the eight flavors. This explains why the optimal value of $\xi$ (i.e., the value corresponding to greatest clustering) is approximately 0.154, irrespective of problem size $m$. 

Figure 3.3: Optimal value of $\xi$ for various values of $m$ for Poisson's equation
Figure 3.4: Eigenvalues of flavor 1 for Poisson’s equation for various values of $\xi$ for $m = 10$

We also compare the ease of finding the value of $\xi$ that minimizes $\norm{\sigma (I - P^{-1}A)}_2$ compared to that of minimizing a condition number. Assuming sufficiently large $m$, the value of $\xi$ that minimizes $\norm{\sigma (I - P^{-1}A)}_2$ is seen a priori to be about 0.154. On the other hand, computing condition numbers, let alone minimizing them with respect to $\xi$, is an exceedingly difficult task. Indeed, both [57] and [136] report on methods which estimate only the order of magnitude of condition numbers. However, it is evident that if we wish to use condition numbers to determine the optimal value of $\xi$, we require more precise information than mere orders of magnitude. And, as is readily seen from Figures 3.1 and 3.2, the insensitivity of optimal $\xi$ with respect to $m$ that we find for $\norm{\sigma (I - P^{-1}A)}_2$ does not hold for the condition numbers, negating the possibility of a priori optimality.

Finally, we examine the issue of the tradeoff between increasing the speed of convergence of Bi-CGSTAB using the optimal value of $\xi$ and the commensurate loss of accuracy. To address this question, we solved Poisson’s equation with Dirichlet boundary conditions for two cases in which
the analytical solution is known. In the first numerical experiment, the analytical solution is

\[ u(x,y) = 100(y - x)^2 + (1 - x)^2. \] (3.1)

We first set \( m = 40 \) and determine the corresponding optimal value of \( \xi \) from MINOS (in this case,
Figure 3.6: Eigenvalues of flavor 3 for Poisson’s equation for various values of $\xi$ for $m = 10$

$\xi = 0.15488$). Now, let

$$
\mathbf{v} =
\begin{bmatrix}
  u_{0,0} \\
  u_{0,1} \\
  \vdots \\
  u_{0,m_0} \\
  u_{1,0} \\
  u_{1,1} \\
  \vdots \\
  u_{1,m_0} \\
  \vdots \\
  u_{m_x,0} \\
  u_{m_x,1} \\
  \vdots \\
  u_{m_x,m_y}
\end{bmatrix}
$$
Figure 3.7: Eigenvalues of flavor 4 for Poisson’s equation for various values of $\xi$ for $m = 10$

Since the analytical solution is known, we had the Bi-CGSTAB iterations stop when $\|v_{\text{analytical}} - v_{\text{numerical}}\|_{\infty} < 10^{-2}$. For this case, 23 iterations were required for convergence and the average time of three runs was 14.51 seconds.

We compare this result by running the same problem with $\xi = \frac{1}{\sqrt{12}}$ for various problem sizes. The results, where we took the mean of three run times for each value of $m$, is given in Table 3.1. The analysis of Table 3.1 has two components. First we note, for $m = 40$, that using the optimal value of $\xi = 0.15488$ takes 14.51 seconds to reach convergence while running the same problem with the Gaussian value $\xi = \frac{1}{\sqrt{12}}$ requires 19.74 seconds. Thus, within the confines of the attainable accuracy using a non-Gaussian value for $\xi$, we obtain a faster solution using the optimal value of $\xi$ as opposed to using the Gaussian value. This demonstrates that the using the optimal value of $\xi$ results in real time savings for fixed accuracy and fixed problem size.

Let us now analyze the effect of changing the problem size by decreasing $m$ while retaining
the Gaussian value of $\xi$. Because the mesh spacing is increased, we lose some accuracy. However, because the problem is now smaller, convergence is reached in fewer iterations. Table 3.1 shows how much time is required to reach the convergence criterion for various values of $m$. We see that we obtain faster results using $m = 40$ and its optimal value of $\xi$, provided $m \geq 36$. If $m < 36$, then we obtain faster results using the Gaussian value $\xi = \frac{1}{\sqrt{12}}$.

We obtain similar results when the analytical solution of Poisson's equation was

$$u(x, y) = \sin x \sin y.$$  \hfill (3.2)

Again, we first let $m = 40$ and used the corresponding optimal value of $\xi = 0.15488$ from MINOS. We find for this example that 18 iterations and 11.30 seconds were required for convergence. The comparative results for the Gaussian value $\xi = \frac{1}{\sqrt{12}}$ are given in Table 3.2. For this example, we see
that for fixed accuracy, using the optimal value of $\xi$ with $m = 40$ produces faster results than using the Gaussian value of $\xi = \frac{1}{\sqrt{12}}$ if $m \geq 38$. If $m < 38$, then we obtain faster results using $\xi = \frac{1}{\sqrt{12}}$.

### 3.2 Model parabolic equation

Recall the model parabolic equation (2.33) and its collocation discretization (2.37). Consider the matrix on the left side of (2.37), namely

$$A^* = C - \tau A,$$

where $A^*$ assumes the role played by $A$ in the eigenvalue analysis of Poisson’s equation. Let $P^*$ be the block Red-Black Gauss-Seidel preconditioning matrix that corresponds to $A^*$. By comparing (2.2) and (2.35), it is clear that the entries of $A$ are significantly larger than those of $C$, due primarily...
Figure 3.10: Eigenvalues of flavor 7 for Poisson’s equation for various values of $\xi$ for $m = 10$

to the presence of the $h^2$ term in the denominator of the entries of $A$. For convenience, let us rewrite (3.3) as

$$A^* = C - \tau^\diamond A^\diamond,$$

where $\tau^\diamond = \frac{r}{h} = \frac{8 \Delta t}{h^2}$ and $A^\diamond = h^2 A$.

So, for values of $\tau^\diamond$ that are large, $\tau^\diamond A^\diamond$ dominates $C$ and we obtain eigenvalue results very similar to those given in Figures 3.4 through 3.11, including the characterization of the optimal value of $\xi$ as approximately 0.154 for sufficiently large $m$ [22]. If, however, $\tau^\diamond$ is small, then the presence of the matrix $C$ becomes significant. In Figure 3.12, we show how $\left\| \sigma \left( I - (P^*)^{-1} A^* \right) \right\|_2$ varies as a function of $\xi$ for various values of $\tau$ for $m = 20$. For $\tau > 10^{-1}$, the corresponding curve is visually indistinguishable from that of $\tau = 10^{-1}$; for $0 < \tau < 10^{-6}$, the corresponding curve is visually indistinguishable from that of $\tau = 10^{-6}$. What is particularly interesting is the fact that for
sufficiently small \( \tau \), we may obtain \( \| \sigma \left( I - (P^*)^{-1} A^* \right) \|_2 \approx 0 \), provided \( \xi \) is chosen appropriately. Note that this is precisely the condition we desire for optimal clustering of eigenvalues about the origin of the complex plane, which in turn corresponds to fastest convergence of Bi-CGSTAB.

In Figure 3.13, we consider the value of \( \| \sigma \left( I - (P^*)^{-1} A^* \right) \|_2 \) that corresponds to optimal \( \xi \) as a function of \( \tau \) for \( m = 10, 20, 30, \) and \( 40 \). We see that by making \( \tau \) sufficiently small, we may obtain optimal \( \xi \) that makes \( \| \sigma \left( I - (P^*)^{-1} A^* \right) \|_2 \approx 0 \).

Figure 3.14 shows the optimal value of \( \xi \) as a function of \( \tau \) for \( m = 10, 20, 30, \) and \( 40 \). We plainly see that as \( \tau \) decreases that the optimal value of \( \xi \) increases. Indeed, for very small \( \tau \), the optimal value of \( \xi \) is approximately \( \frac{1}{2} \). In particular, should we wish to use the Gaussian value of \( \xi = \frac{1}{\sqrt{12}} \approx 0.29 \) (which provides greatest accuracy), we may select \( \tau \) appropriately.

Thus, at each time step, if we seek to solve the collocation discretization (2.37) as quickly as possible, we may set the value of \( \tau \) to be sufficiently small to achieve \( \| \sigma \left( I - (P^*)^{-1} A^* \right) \|_2 \approx 0 \).
Table 3.1: Results for Poisson’s equation with $\xi = \frac{1}{\sqrt{12}}$ and analytical solution (3.1) for various values of $m$

<table>
<thead>
<tr>
<th>$m$</th>
<th>iterations for convergence</th>
<th>mean run time [secs]</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>16</td>
<td>2.38</td>
</tr>
<tr>
<td>30</td>
<td>25</td>
<td>8.26</td>
</tr>
<tr>
<td>32</td>
<td>26</td>
<td>10.10</td>
</tr>
<tr>
<td>34</td>
<td>28</td>
<td>12.28</td>
</tr>
<tr>
<td>36</td>
<td>31</td>
<td>15.75</td>
</tr>
<tr>
<td>38</td>
<td>31</td>
<td>17.01</td>
</tr>
<tr>
<td>40</td>
<td>32</td>
<td>19.74</td>
</tr>
</tbody>
</table>

Table 3.2: Results for Poisson’s equation with $\xi = \frac{1}{\sqrt{12}}$ and analytical solution (3.2) for various values of $m$

<table>
<thead>
<tr>
<th>$m$</th>
<th>iterations for convergence</th>
<th>mean run time [secs]</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td>1.48</td>
</tr>
<tr>
<td>30</td>
<td>17</td>
<td>5.62</td>
</tr>
<tr>
<td>32</td>
<td>18</td>
<td>6.89</td>
</tr>
<tr>
<td>34</td>
<td>20</td>
<td>8.81</td>
</tr>
<tr>
<td>36</td>
<td>21</td>
<td>10.64</td>
</tr>
<tr>
<td>38</td>
<td>22</td>
<td>12.12</td>
</tr>
<tr>
<td>40</td>
<td>24</td>
<td>14.86</td>
</tr>
</tbody>
</table>

Since $\tau = \theta \Delta t$, small values of $\tau$ force either small time steps or small $\theta$ (which corresponds to a mostly explicit time advance).

Let us now examine the stability of the time-stepping scheme (2.37), which may be written

$$x^{(n+1)} = W x^{(n)} - p^{(n)}, \quad (3.4)$$

where

$$W = (C - \theta \Delta t A)^{-1} (C + [1 - \theta] \Delta t A)$$

and

$$p^{(n)} = (C - \theta \Delta t A)^{-1} \left[ \theta \Delta t b^{(n+1)} + (1 - \theta) \Delta t b^{(n)} \right].$$

For stability, we require all the eigenvalues of $W$ to lie inside (or on) the unit circle in the complex plane. We may write

$$W = (I - \theta \rho G)^{-1} (I + [1 - \theta] \rho G),$$
Figure 3.12: $\| \sigma \left( I - (P^*)^{-1} A^* \right) \|_2$ as a function of $\xi$ for various values of $\tau$ for $m = 20$

where $G = h^2 C^{-1} A = C^{-1} A^\dagger$ (the entries of which depend only on $\xi$) and

$$\rho = \frac{\Delta t}{h^2}.$$ (3.5)

Appealing to well-known linear algebra facts (e.g. [78]), we see that the eigenvalues of $G$ and those of $W$ are related: If $\lambda$ is an eigenvalue of $G$, then $\frac{1+(1-\theta)\rho}{1-\sigma^2 \rho \lambda}$ is an eigenvalue of $W$.

It is possible to show that all eigenvalues of $G$ are real and negative (see Theorem 4.7). If we then let $\lambda = -\omega^2$, where $\omega$ is a real number, then the condition that ensures stability of the time-stepping scheme (3.4) is

$$\left| \frac{1-\omega^2 (1-\theta) \rho}{1+\omega^2 \theta \rho} \right| < 1$$ (3.6)
for all eigenvalues $\lambda = -\omega^2$ of $G$. Algebraic manipulation of (3.6) yields the stability condition

$$\omega^2 \rho (1 - 2\theta) \leq 2, \quad (3.7)$$

which must be true when $\frac{1}{2} \leq \theta \leq 1$. That is, if $\frac{1}{2} \leq \theta \leq 1$, we get unconditional stability (i.e., stability irrespective of the value of $\rho$.) If, on the other hand, $0 \leq \theta < \frac{1}{2}$, stability is guaranteed only if $\rho$ satisfies (3.7) for all appropriate values of $\omega^2$, i.e., for all eigenvalues $\lambda$ of $G$.

We thus see that if we want both unconditional stability and the very rapid convergence that occurs when $\left\| \sigma \left( I - \left( P^* \right)^{-1} A^* \right) \right\|_2 \approx 0$, we must choose very small time steps. For example, with reference to Figure 3.12, we achieve $\left\| \sigma \left( I - \left( P^* \right)^{-1} A^* \right) \right\|_2 \approx 0$ for $m = 20$ when $\xi \approx 0.32$ and $\tau \approx 10^{-4}$. Thus unconditional stability requires that $10^{-4} < \Delta t < 2 \cdot 10^{-4}$.

Let us compare this restriction on $\Delta t$ to that imposed by the fully explicit time-stepping scheme
corresponding to $\theta = 0$. With reference to (3.7) and the fact that all eigenvalues of $G$ are real and negative, it is clear that if (3.7) holds for the largest value of $\omega^2$ (which corresponds to the eigenvalue greatest in absolute value), then (3.7) holds for all relevant values of $\omega^2$ (i.e., for all eigenvalues $\lambda$ of $G$). Thus (3.7) may be written

$$\omega_{\text{max}}^2 \rho (1 - 2\theta) \leq 2. \quad (3.8)$$

In Corollary 4.8, we derive a formula for $\omega_{\text{max}}^2$:

$$\omega_{\text{max}}^2 = \frac{48}{1 - 4\xi^2}. \quad (3.9)$$

Note that $\omega_{\text{max}}^2$ depends only upon $\xi$; in particular, $\omega_{\text{max}}^2$ is independent of problem size $m$.

Using (3.9), we find that $\omega_{\text{max}}^2 \approx 81.3$ corresponds to $\xi = 0.32$. Setting $h = 0.05$ (which corresponds to $m = 20$) and recalling (3.5), we see that (3.8) requires $\Delta t < 6.15 \cdot 10^{-5}$ for the fully
Table 3.3: Number of iterations required for convergence of the model parabolic equation over one time step for various time step sizes

<table>
<thead>
<tr>
<th>$\Delta t$</th>
<th>optimal $\xi$</th>
<th>iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-4}$</td>
<td>0.32020</td>
<td>2</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>0.20409</td>
<td>4</td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>0.16094</td>
<td>12</td>
</tr>
<tr>
<td>$10^{-1}$</td>
<td>0.15742</td>
<td>26</td>
</tr>
</tbody>
</table>

explicit time stepping scheme. Recall that the time step restriction for unconditional stability and very rapid convergence was $10^{-4} < \Delta t < 2 \cdot 10^{-4}$. Thus for this example, the time steps permissible for unconditional stability and very rapid convergence are roughly two or three times the size of time steps imposed by stability requirements for the fully explicit time-stepping problem.

We saw above that if we take very small time steps, then it is possible to achieve very rapid convergence. A natural question to ask is: If we increase the time step size, how will the commensurate increase in iterations required for convergence compare? For example, if we double the time step size and the number of iterations increases by less than a factor of two, then we should take one doubled time step as opposed to two smaller ones. On the other hand, if we double the time step size and the number of iterations increases by more than a factor of two, then it would be less expensive to take two smaller time steps instead of one doubled larger one.

To examine this question, we solved the model parabolic equation (2.33) over one time step for various time step sizes. We used $m = 20$ and $\theta = \frac{1}{2}$. For each value of $\Delta t$, we used the corresponding optimal value of $\xi$ as determined from the MINOS optimization software. The results are given in Table 3.3.

We see from Table 3.3 that taking very small step sizes to obtain extremely fast convergence is not a good strategy. For example, if we took ten time steps each of size $10^{-4}$, we would require 20 iterations. This is much greater than the 4 iterations required by taking a single time step of size $10^{-3}$. 
3.3 Flow equation

The equation that models two-dimensional horizontal flow in an isotropic aquifer is given by [53, 76]:

\[
\frac{\partial}{\partial x} \left( T \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( T \frac{\partial u}{\partial y} \right) - S \frac{\partial u}{\partial t} + Q = 0,
\]

where \( T \) = the transmissivity of the aquifer, \( S \) = the storativity of the aquifer, \( Q \) = a source/sink term, and \( u \) = hydraulic head. If the medium is homogeneous, then \( T \) and \( S \) are constants and we obtain

\[
\frac{\partial u}{\partial t} = \frac{T}{S} \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + \frac{Q}{S}.
\] (3.10)

If we now apply to (3.10) the discretization applied to (2.33) to produce (2.37), then we obtain

\[
\left( C - \frac{T \theta \Delta t}{S h^2} A^{\phi} \right) x^{(n+1)} = \left( C + \frac{T (1 - \theta) \Delta t}{S h^2} A^{\phi} \right) x^{(n)} - \left[ \theta \Delta t k^{(n+1)} + (1 - \theta) \Delta t k^{(n)} \right],
\] (3.11)

which is analogous to (2.37). It is easy to see, with respect to the discussion above for the model parabolic equation, that the dimensionless expression \( T \frac{\theta \Delta t}{S h^2} \) plays the role of \( \tau^{\phi} \), that \( T \frac{\Delta t}{S h^2} \) plays the role of \( \tau \), and that \( \frac{T \Delta t}{S h^2} \) plays the role of \( \rho \). Thus the analysis given for the model parabolic equation may be readily applied to the flow equation (3.10).

3.4 Transport equation

The equation that models two-dimensional contaminant transport in a porous medium is given by [76]:

\[
D \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \left( V_x \frac{\partial u}{\partial x} + V_y \frac{\partial u}{\partial y} \right) - \frac{\partial u}{\partial t} + Q = 0,
\] (3.12)

where \( D \) = hydrodynamic dispersion, \( V_x \) and \( V_y \) are velocities in the \( x \)- and \( y \)-directions, respectively, \( Q \) is a source/sink term, and \( u \) is contaminant concentration. Even if \( D \), \( V_x \), and \( V_y \) are constant, the theory discussed above is not directly applicable to (3.12) because of the presence of the advective terms \( \frac{\partial u}{\partial x} \) and \( \frac{\partial u}{\partial y} \). This is easily seen by examining the one-dimensional case. If we recall the
matrix (1.10) which arose from the collocation discretization of the differential operator \( \frac{\partial^2}{\partial x^2} \), assume a uniform mesh, and evaluate its entries using the formulas (1.1) and (1.2), we obtain the repeated computational molecule in matrix \( A \), which is given symbolically as

\[
\begin{bmatrix}
\alpha & -\beta & \delta & -\gamma \\
\delta & \gamma & \alpha & \beta 
\end{bmatrix}
\]  

(3.13)

The repeated computational molecule of the collocation discretization of the differential operator \( u \) is also given symbolically by (3.13), although, of course, \( \alpha, \beta, \gamma, \) and \( \delta \) represent different quantities in the \( \frac{\partial^2}{\partial x^2} \) and \( u \) cases. However, the repeated computational molecule of the collocation discretization of the differential operator \( \frac{\partial u}{\partial x} \) is given symbolically by

\[
\begin{bmatrix}
-\alpha & \beta & \alpha & -\beta \\
-\alpha & -\beta & \alpha & \beta 
\end{bmatrix}
\]

which is clearly different from (3.13). This explains why the theory derived for Poisson’s equation works also for the model parabolic equation (2.33) but is not directly applicable to (3.12). However, by utilizing a transformation of the dependent variable \( u \), we are able to derive an equivalent form to which the theory applies.

The idea behind the transformation is a generalization of the well-known technique in the theory of ODEs (see, for example, [18]) by which one eliminates the first derivative term in the ODE

\[
y'' + p(x)y' + q(x)y = r(x).
\]

This idea was applied to PDEs in [66], [67], and [68]. Assuming constant \( D \), \( V_x \), and \( V_y \), the transformation is

\[
w(x, y, t) = \frac{u(x, y, t)}{\exp\left[\frac{1}{2D}(V_x x + V_y y)\right]}.
\]
If we define $V = \sqrt{V_x^2 + V_y^2}$ and $\zeta = \frac{V}{TV}$, then the transformed PDE (3.12) becomes

$$D \left( \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} \right) - \zeta \frac{\partial w}{\partial t} = \frac{-Q}{\exp \left[ \frac{1}{2T} (V_x x + V_y y) \right]}$$

(3.14)

Because (3.14) lacks the first derivative terms that (3.12) has, we may apply to (3.14) the theory developed for the model parabolic equation (2.33).

The collocation discretization of (3.14) is seen to be

$$[(1 + \zeta \Delta t) C - (D \Delta t) A] x^{(n+1)} = [(1 + \zeta (1 - \theta) \Delta t) C + (D (1 - \theta) \Delta t) A] x^{(n)} - \left[ \theta \Delta t k^{(n+1)} + (1 - \theta) \Delta t k^{(n)} \right].$$

So the role of $\tau$ in the collocation discretization of model parabolic equation (2.37) is assumed here by the dimensionless expression $\frac{D \Delta t}{1 + \zeta (1 - \theta) \Delta t}$, the role of $\tau$ is assumed by $\frac{D \Delta t}{1 + \zeta (1 - \theta) \Delta t}$, and the role of $\rho$ is assumed by $\frac{D \Delta t}{1 + \zeta (1 - \theta) \Delta t}$. 
Chapter 4

The Eigenproblem of $C^{-1}A$

We discuss here the eigenproblem for the matrix $C^{-1}A$, the results of which are cited in the stability analysis of the previous chapter. We first present analysis for the one-dimensional problem and then use these results to study the two-dimensional case. An analysis for the one-dimensional case is included in [14], but is limited to the Gaussian collocation point location $\xi = \frac{1}{\sqrt{12}}$ and uses an approach quite different from ours.

4.1 One dimensional formulation

In the one-dimensional problem, matrix $A$ represents the differential operator $u''$ while matrix $C$ represents the differential operator $u$. The eigenproblem may be written

$$C^{-1}Ax = \lambda x \quad (4.1)$$

which is equivalent to

$$Ax - \lambda Cx = 0. \quad (4.2)$$

This is merely the collocation discretization of the (continuous) Sturm-Liouville ODE

$$u'' - \lambda u = 0 \quad (4.3)$$
with homogeneous Dirichlet boundary conditions on the interval $[0, 1]$. The solution of (4.3) consists of the eigenfunctions

$$u = \sin j\pi x$$

(4.4)

and the eigenvalues

$$\lambda = -(j\pi)^2,$$

$j = 1, 2, 3, \ldots$

With reference to (1.1), (1.2), (1.10), and (4.2), we see that we may write

$$A = \begin{bmatrix}
-\beta & \alpha & -\gamma \\
\gamma & -\alpha & \beta \\
-\alpha & -\beta & \alpha & -\gamma \\
\alpha & \gamma & -\alpha & \beta \\
& \ddots & \ddots & \ddots & \ddots \\
-\alpha & -\beta & \alpha & -\gamma \\
\alpha & \gamma & -\alpha & \beta \\
& \ddots & \ddots & \ddots & \ddots & \ddots \\
-\alpha & -\beta & -\gamma \\
\alpha & \gamma & \beta
\end{bmatrix},$$

61
where \( \alpha = \frac{12\xi}{k^2} \), \( \beta = \frac{6\xi + 1}{k^2} \), \( \gamma = \frac{6\xi - 1}{k^2} \);

\[
C = \begin{bmatrix}
\chi & \frac{1}{2} - \zeta & -\psi \\
\psi & \frac{1}{2} + \zeta & -\chi \\
\frac{1}{2} + \zeta & \chi & \frac{1}{2} - \zeta & -\psi \\
\frac{1}{2} - \zeta & \psi & \frac{1}{2} + \zeta & -\chi \\
\end{bmatrix},
\]

where \( \chi = \frac{(1+2\xi)^2(1-2\xi)}{8} \), \( \psi = \frac{(1-2\xi)^2(1+2\xi)}{8} \), \( \zeta = \frac{\xi (3-4\xi)^2}{2} \); and

\[ x = c_1 x_1 + c_2 x_2, \quad (4.5) \]

where

\[
x_1 = \begin{bmatrix}
(m - j) \pi \cos [(m - j) \pi x_0] \\
\sin [(m - j) \pi x_1] \\
(m - j) \pi \cos [(m - j) \pi x_1] \\
\sin [(m - j) \pi x_2] \\
\vdots \\
(m - j) \pi \cos [(m - j) \pi x_{m-2}] \\
\sin [(m - j) \pi x_{m-1}] \\
(m - j) \pi \cos [(m - j) \pi x_{m-1}] \\
(m - j) \pi \cos [(m - j) \pi x_{m}]
\end{bmatrix}, \quad \text{and} \quad
x_2 = \begin{bmatrix}
(m + j) \pi \cos [(m + j) \pi x_0] \\
\sin [(m + j) \pi x_1] \\
(m + j) \pi \cos [(m + j) \pi x_1] \\
\sin [(m + j) \pi x_2] \\
\vdots \\
(m + j) \pi \cos [(m + j) \pi x_{m-2}] \\
\sin [(m + j) \pi x_{m-1}] \\
(m + j) \pi \cos [(m + j) \pi x_{m-1}] \\
(m + j) \pi \cos [(m + j) \pi x_{m}]
\end{bmatrix}.
\]

Here \( x_k = \frac{k}{n} = kh, \ k = 0,1, \ldots m \) and \( j = 0,1, \ldots m \). It is clear that \( x_1 \) and \( x_2 \) are merely
Hermite discretizations of the eigenfunctions (4.4) with the homogeneous Dirichlet boundary conditions included. We require entries with cosines because the Hermite polynomials interpolate both eigenfunction values (i.e., the sines) and their derivative values (i.e., the cosines). Since every scalar multiple of an eigenvector is again an eigenvector, we may, without loss of generality, assume that 
\[ c_1 = 1 \] and replace (4.5) with
\[ x = x_1 + c_2 x_2. \]

It is evident that solving the eigenproblem (4.2) is equivalent to solving the system of two equations

\[
\begin{align*}
\left[ -\alpha - \lambda \left( \frac{1}{2} + \zeta \right) \right] v_1 + [-\beta - \lambda \chi] v_2 + \left[ \alpha - \lambda \left( \frac{1}{2} - \zeta \right) \right] v_3 + [-\gamma + \lambda \psi] v_4 &= 0 \quad (4.6) \\
\left[ \alpha - \lambda \left( \frac{1}{2} - \zeta \right) \right] v_1 + [\gamma - \lambda \psi] v_2 + \left[ -\alpha - \lambda \left( \frac{1}{2} + \zeta \right) \right] v_3 + [\beta + \lambda \chi] v_4 &= 0 \quad (4.7)
\end{align*}
\]

in the two unknowns \( \lambda \) and \( c_2 \), where

\[
\begin{align*}
v_1 &= \sin \left[ (m - j) \pi x_k \right] + c_2 \sin \left[ (m + j) \pi x_k \right] \\
v_2 &= (m - j) \pi \cos \left[ (m - j) \pi x_k \right] + c_2 (m + j) \pi \cos \left[ (m + j) \pi x_k \right] \\
v_3 &= \sin \left[ (m - j) \pi x_{k+1} \right] + c_2 \sin \left[ (m + j) \pi x_{k+1} \right] \\
v_4 &= (m - j) \pi \cos \left[ (m - j) \pi x_{k+1} \right] + c_2 (m + j) \pi \cos \left[ (m + j) \pi x_{k+1} \right]
\end{align*}
\]

Since (4.6) and (4.7) hold for all \( k = 0, 1, \ldots, m - 1 \), we simplify matters by considering the case \( k = 0 \). In this case, \( x_k = x_0 = 0, x_{k+1} = x_1 = h \), and, using trigonometric identities, we obtain

\[
\begin{align*}
v_1 &= 0 \\
v_2 &= \pi \left[ m - j + c_2 (m + j) \right] \\
v_3 &= (1 - c_2) \sin j\pi h \\
v_4 &= -c_2 \cos j\pi h
\end{align*}
\]

(4.8)
We now eliminate \( c_2 \) from the system (4.6), (4.7), obtaining a quadratic equation in \( \lambda \):

\[
a \lambda^2 + b \lambda + c = 0, \quad (4.9)
\]

where

\[
a &= \frac{1}{4} (2\xi - 1)(2\xi + 1) (5 - 4\xi^2 + \kappa - 4\xi^2 \kappa) \quad (4.10)
\]
\[
b &= \frac{8}{h^2} (-3 + 4\xi^2 + 4\xi^2 \kappa)
\]
\[
c &= -\frac{48}{h^4} (1 + \kappa).
\]

Here \( \kappa = \cos j\pi h \). The discriminant \( b^2 - 4ac \) is \( \frac{16}{h^4} D^* \), where

\[
D^* = (21 - 24\xi^2 + 16\xi^4) + (-18 + 32\xi^4) \kappa + (-3 + 24\xi^2 + 16\xi^4) \kappa^2.
\]

**Lemma 4.1** The eigenvalues \( \lambda \) that satisfy the quadratic equation (4.9) are all real.

**Proof.** It suffices to show that \( D^*(\xi, \kappa) \geq 0 \) on the domain \( \xi \in (0, \frac{1}{2}) \), \( \kappa \in [-1, 1] \). On the boundary of the domain, we see that \( D^*(\xi, \kappa) \geq 0 \):

\[
D^*(0, \kappa) = 3 (1 - \kappa)(7 + \kappa) \geq 0
\]
\[
D^* \left( \frac{1}{2}, \kappa \right) = 4 (2 - \kappa)^2 > 0
\]
\[
D^*(\xi, -1) = 36 > 0
\]
\[
D^*(\xi, 1) = 64\xi^4 > 0.
\]

We will be done when we show that there are no critical points in the interior of the domain. Indeed, setting \( \frac{\partial D^*}{\partial \xi} = 16\xi (1 + \kappa) (3 + 3\kappa + 4\xi^2 + 4\kappa \xi^2) \) equal to zero implies that \( \kappa = -1 \) (which is on the boundary of the domain) or \( \kappa = \frac{3 - 4\xi^2}{3 + 4\xi^2} \). Setting \( \frac{\partial D^*}{\partial \kappa} = 2 (-9 - 3\kappa + 24\xi^2 + 16\xi^4 + 16\kappa \xi^2) \) equal
to zero implies \( \xi = \frac{9 - 16\xi^4}{3 + 24\xi^2 + 16\xi^4} \). We thus have critical points where \( \xi \) satisfies

\[
\frac{3 - 4\xi^2}{3 + 4\xi^2} = \frac{9 - 16\xi^4}{-3 + 24\xi^2 + 16\xi^4}.
\]

However, it is easy to see that (4.11) reduces to a contradiction. This shows there are no critical points in the interior of our domain, allowing us to conclude that \( D^*(\xi, \kappa) \) is indeed non-negative.

Q.E.D.

**Lemma 4.2** In (4.10), we have \( a < 0, b < 0, \) and \( c \leq 0 \).

**Proof.** To show \( a < 0 \), it suffices to show that \( a^*(\xi, \kappa) = 5 - 4\xi^2 + \kappa - 4\xi^2 \kappa > 0 \). Note that \( a^* \) is linear in \( \kappa \) and that

\[
a^*(\xi, -1) = 4 > 0
\]

\[
a^*(\xi, 1) = 2(3 - 4\xi^2) > 0.
\]

Thus \( a^*(\xi, \kappa) > 0 \).

To show \( b < 0 \), it suffices to show that \( b^*(\xi, \kappa) = -3 + 4\xi^2 + 4\xi^2 \kappa < 0 \). Note that \( b^* \) is linear in \( \kappa \) and that

\[
b^*(\xi, -1) = -3 < 0
\]

\[
b^*(\xi, 1) = -3 + 8\xi^2 < 0.
\]

Thus \( b^*(\xi, \kappa) < 0 \).

That \( c \leq 0 \) is obvious. Q.E.D.

The sum of the roots of (4.9) is \( -\frac{a}{\kappa} \), which must be negative. The product of the roots of (4.9) is \( \frac{c}{\kappa} \), which must be non-positive. It must therefore be the case that both roots of (4.9) are negative or that one is negative and one is zero. Since the roots of (4.9) are the eigenvalues we seek, we
know that all eigenvalues $\lambda$ are non-positive.

If $\lambda = 0$, then the product of the roots of (4.9) is zero, so we have $c = 0$. Thus $\kappa = -1$ (see (4.10)), which, by the definition of $\kappa$, implies that $j = m$. In this case, (4.2) reduces to

$$2m\pi c_2 A \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = 0.$$ 

Since $A$ is non-singular, we conclude that $c_2 = 0$. But this implies that $x$ in (4.1) is a zero eigenvector, which is impossible. So $\lambda = 0$ is not an eigenvalue, proving

**Theorem 4.3** All eigenvalues $\lambda$ of (4.1) are negative.

According to (4.9), the eigenvalues should occur in pairs given by the quadratic formula. However, we just saw that for $j = m$, one solution of (4.9), namely $\lambda = 0$, is not an eigenvalue. If, on the other hand, we have $j = 0$, then (4.8) becomes $v_1 = v_3 = 0$ and $v_2 = -v_4 = \pi m (c_2 + 1)$. In this case, (4.6) and (4.7) are identical, producing one equation in the two unknowns $c_2$ and $\xi$:

$$\frac{c_2 + 1}{4h^3 \pi} (-8 - h^2 \lambda + 4h^2 \lambda \xi^2) = 0.$$ 

If $c_2 = -1$, then $v_2 = v_4 = 0$, which implies the impossibility that $x$ is a zero eigenvector. Thus the case $j = 0$ also provides a single eigenvalue.

We can summarize:
Theorem 4.4 The $2m$ eigenvalues $\lambda$ of (4.1), which are all negative, are determined by solving the quadratic equation (4.9) for $j = 0, 1, \ldots, m$. When $j = 0$ or $j = m$, we produce only one eigenvalue for each of these values of $j$. For $j = 1, 2, \ldots, m - 1$, we produce two eigenvalues for each value of $j$.

Corollary 4.5 Of all the eigenvalues of $C^{-1}A$, the one that is greatest in absolute value corresponds to $j = m$.

Proof. The eigenvalues $\lambda$ are given by the quadratic formula

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

(4.12)

where $a$, $b$, and $c$ are given in (4.10). To prove the corollary, it is clear that it suffices to show that $j = m$ maximizes $a$ (i.e., makes the denominator in (4.12) a negative number of smallest absolute value possible); minimizes $b$ (i.e., makes $-b$ the largest (positive) number possible); and maximizes $b^2 - 4ac$ (i.e., makes $b^2 - 4ac$ the largest (positive) number possible). Recall that $j = m$ corresponds to $\kappa = -1$.

We saw in the proof of Lemma 4.2 that $a^*$ is linear in $\kappa$. Since $\frac{\partial a^*}{\partial \kappa} = 1 - 4\xi^2 > 0$, we see that $a^*$ is minimized at $\kappa = -1$. Thus $a$ is maximized at $\kappa = -1$.

We also saw in the proof of Lemma 4.2 that $b^*$ is linear in $\kappa$. Since $\frac{\partial b^*}{\partial \kappa} = 4\xi^2 > 0$, we see that $b^*$ is minimized at $\kappa = -1$. Thus $b$ is minimized at $\kappa = -1$.

Since $b < 0$ is minimized at $\kappa = -1$, we see that $b^2$ is maximized at $\kappa = -1$. To maximize $\lambda$ in (4.12), we thus want to subtract from $b^2$ the non-negative quantity $4ac$ of minimum magnitude. This is achieved when $c = 0$, which, as we saw above, occurs when $\kappa = -1$. Q.E.D.

Corollary 4.6 For the one-dimensional problem, the eigenvalue of $C^{-1}A$ of greatest magnitude is

$$\lambda = \frac{-24}{b^2(1 - 4\xi^2)}.$$

Proof. It is clear that, with respect to (4.12), we have

$$\lambda_+ < \lambda_- \leq 0.$$
When \( \kappa = -1 \), which corresponds to the eigenvalue of greatest magnitude, we obtain \( \lambda_+ = \frac{-2\kappa}{4(1 - \kappa^2)} \).

\textbf{Q.E.D.}

We make one last comment before tackling the two-dimensional problem. It is clear that once we have obtained the value of an eigenvalue \( \lambda \), we may substitute this value into either (4.6) or (4.7) to find the value of the corresponding parameter \( c_2 \), thus allowing us to compute the eigenvector corresponding to the eigenvalue \( \lambda \).

### 4.2 Two-dimensional formulation

We now discuss the analogous two-dimensional problem. That is, we want to solve the eigenproblem

\[ C^{-1}A\mathbf{x} = \lambda \mathbf{x}. \]  \hfill (4.13)

In this case, matrix \( A \) represents the differential operator \( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \) while matrix \( C \) represents the differential operator \( u \). As in the one-dimensional case, the eigenproblem (4.13) may be written

\[ A\mathbf{x} - \lambda C\mathbf{x} = 0. \]  \hfill (4.14)

This is merely the collocation discretization of the (continuous) PDE

\[ \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \lambda u = 0 \]  \hfill (4.15)

with homogeneous Dirichlet boundary conditions on the boundary of \([0, 1] \times [0, 1]\). The solution of (4.15) consists of the eigenfunctions

\[ u = \sin j\pi x \sin \ell \pi y \]

and the eigenvalues

\[ \lambda = -(j\pi)^2 - (\ell\pi)^2, \]  \hfill (4.16)
Let us interpret the result (4.16) as follows. The eigenvalues given by (4.16) are precisely the sum of the eigenvalues of two one-dimensional problems (4.3).

The corresponding result for the (discrete) two-dimensional eigenproblem (4.14) is completely analogous to that for the continuous two-dimensional eigenproblem (4.15). We first consider to have two sets of identical eigenvalues to the one-dimensional problem: Let \( \Lambda_j = \{ \lambda_1^j, \lambda_2^j, \ldots, \lambda_{2m}^j \} \) be the eigenvalues of the one-dimensional problem (4.2) and let \( \Lambda_\ell = \{ \lambda_1^\ell, \lambda_2^\ell, \ldots, \lambda_{2m}^\ell \} \) be such that 
\[
\lambda_k^j = \lambda_k^\ell \text{ for } k = 1, 2, \ldots, 2m.
\]
Write \( \lambda \) in (4.14) as \( \lambda = \lambda_j + \lambda_\ell \), where \( \lambda_j \in \Lambda_j \) and \( \lambda_\ell \in \Lambda_\ell \).

We now write the four (because, for two-dimensional problems, we have four collocation points per rectangular finite element) equations analogous to (4.6) and (4.7):

\[
[M_{11}, M_{12}, M_{13}, -M_{14}, M_{21}, M_{22}, M_{23}, -M_{24}, M_{31}, M_{32}, M_{33}, -M_{34}, -M_{41}, -M_{42}, -M_{43}, M_{44}] \mathbf{v} = 0
\]
\[
(4.17)
\]

\[
[M_{31}, M_{32}, M_{33}, -M_{34}, M_{41}, M_{42}, M_{43}, -M_{44}, M_{11}, M_{12}, M_{13}, -M_{14}, -M_{21}, -M_{22}, -M_{23}, M_{24}] \mathbf{v} = 0
\]
\[
(4.18)
\]

\[
[M_{51}, M_{52}, M_{53}, -M_{54}, M_{61}, M_{62}, M_{63}, -M_{64}, M_{15}, M_{16}, M_{17}, -M_{18}, -M_{25}, -M_{26}, -M_{27}, M_{28}] \mathbf{v} = 0
\]
\[
(4.19)
\]

\[
[M_{33}, M_{34}, M_{35}, -M_{36}, M_{43}, M_{44}, M_{45}, -M_{46}, M_{53}, M_{54}, M_{55}, -M_{56}, -M_{63}, -M_{64}, -M_{65}, M_{66}] \mathbf{v} = 0
\]
\[
(4.20)
\]
where

\[
\begin{align*}
M_{11} & = a_{1,1} - \lambda c_{1,1} = \left( \frac{1}{2} + \zeta \right) \left[ -\alpha - \lambda_j \left( \frac{1}{2} + \zeta \right) \right] + \left( \frac{1}{2} + \zeta \right) \left[ -\alpha - \lambda \left( \frac{1}{2} + \zeta \right) \right] \\
M_{12} & = a_{1,2} - \lambda c_{1,2} = \left( \frac{1}{2} + \zeta \right) \left[ -\beta - \lambda_j \chi \right] + \chi \left[ -\alpha - \lambda \left( \frac{1}{2} + \zeta \right) \right] \\
M_{13} & = a_{1,3} - \lambda c_{1,3} = \left( \frac{1}{2} + \zeta \right) \left[ \alpha - \lambda_j \left( \frac{1}{2} - \zeta \right) \right] + \left( \frac{1}{2} - \zeta \right) \left[ -\alpha - \lambda \left( \frac{1}{2} + \zeta \right) \right] \\
M_{14} & = a_{1,4} - \lambda c_{1,4} = \left( \frac{1}{2} + \zeta \right) \left[ \gamma - \lambda_j \psi \right] + \psi \left[ -\alpha - \lambda \left( \frac{1}{2} + \zeta \right) \right] \\
M_{21} & = a_{2,1} - \lambda c_{2,1} = \chi \left[ -\alpha - \lambda_j \left( \frac{1}{2} + \zeta \right) \right] + \left( \frac{1}{2} + \zeta \right) \left[ -\beta - \lambda \chi \right] \\
M_{22} & = a_{2,2} - \lambda c_{2,2} = \chi \left[ -\beta - \lambda_j \chi \right] + \chi \left[ -\beta - \lambda \chi \right] \\
M_{23} & = a_{2,3} - \lambda c_{2,3} = \chi \left[ \alpha - \lambda_j \left( \frac{1}{2} - \zeta \right) \right] + \left( \frac{1}{2} - \zeta \right) \left[ -\beta - \lambda \chi \right] \\
M_{24} & = a_{2,4} - \lambda c_{2,4} = \chi \left[ \gamma - \lambda_j \psi \right] + \psi \left[ -\beta - \lambda \chi \right] \\
M_{31} & = a_{3,1} - \lambda c_{3,1} = \left( \frac{1}{2} - \zeta \right) \left[ -\alpha - \lambda_j \left( \frac{1}{2} + \zeta \right) \right] + \left( \frac{1}{2} + \zeta \right) \left[ \alpha - \lambda \left( \frac{1}{2} - \zeta \right) \right] \\
M_{32} & = a_{3,2} - \lambda c_{3,2} = \left( \frac{1}{2} - \zeta \right) \left[ -\beta - \lambda_j \chi \right] + \chi \left[ \alpha - \lambda \left( \frac{1}{2} - \zeta \right) \right] \\
M_{33} & = a_{3,3} - \lambda c_{3,3} = \left( \frac{1}{2} - \zeta \right) \left[ \alpha - \lambda_j \left( \frac{1}{2} - \zeta \right) \right] + \left( \frac{1}{2} - \zeta \right) \left[ \alpha - \lambda \left( \frac{1}{2} - \zeta \right) \right] \\
M_{34} & = a_{3,4} - \lambda c_{3,4} = \left( \frac{1}{2} - \zeta \right) \left[ \gamma - \lambda_j \psi \right] + \psi \left[ \alpha - \lambda \left( \frac{1}{2} - \zeta \right) \right] \\
M_{41} & = a_{4,1} - \lambda c_{4,1} = \psi \left[ -\alpha - \lambda_j \left( \frac{1}{2} + \zeta \right) \right] + \left( \frac{1}{2} + \zeta \right) \left[ \gamma - \lambda \psi \right] \\
M_{42} & = a_{4,2} - \lambda c_{4,2} = \psi \left[ -\beta - \lambda_j \chi \right] + \chi \left[ \gamma - \lambda \psi \right] \\
M_{43} & = a_{4,3} - \lambda c_{4,3} = \psi \left[ \alpha - \lambda_j \left( \frac{1}{2} - \zeta \right) \right] + \left( \frac{1}{2} - \zeta \right) \left[ \gamma - \lambda \psi \right] \\
M_{44} & = a_{4,4} - \lambda c_{4,4} = \psi \left[ \gamma - \lambda_j \psi \right] + \psi \left[ \gamma - \lambda \psi \right]
\end{align*}
\]

and \( \mathbf{v} \) (analogous to (4.8)) contains the linear combination of relevant entries of discretized eigenfunctions of (4.15):

\[
\mathbf{v} = \sum_{k=1}^{8} d_k \mathbf{w}_k,
\]
where

\[
\begin{align*}
\mathbf{w}_1 &= \begin{bmatrix}
0 \\
(m - j) \pi \sin L \\
\sin J \sin L \\
-(m - j) \pi \cos J \sin L \\
0 \\
-(m - j)(m - \ell) \pi^2 \cos L \\
-(m - \ell) \pi \cos L \sin J \\
(m - j)(m - \ell) \pi^2 \cos J \cos L \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
(m - j)(m - \ell) \pi^2 \\
(m - \ell) \pi \sin J \\
-(m - j)(m - \ell) \pi^2 \cos J \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\mathbf{w}_2 &= \begin{bmatrix}
0 \\
-(m - j) \pi \sin L \\
-\sin J \sin L \\
(m - j) \pi \cos J \sin L \\
0 \\
-(m - j)(m + \ell) \pi^2 \cos L \\
-(m + \ell) \pi \cos L \sin J \\
(m - j)(m + \ell) \pi^2 \cos J \cos L \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
(m - j)(m + \ell) \pi^2 \\
(m + \ell) \pi \sin J \\
-(m - j)(m + \ell) \pi^2 \cos J \\
\end{bmatrix}
\end{align*}
\]
\[
\begin{align*}
\mathbf{w}_7 &= \begin{bmatrix}
0 \\
(m + j) \pi \sin L \\
- \sin J \sin L \\
- (m + j) \pi \cos J \sin L \\
- (m + j)(m - \ell) \pi^2 \cos L \\
(m - \ell) \pi \cos L \sin J \\
(m + j)(m - \ell) \pi^2 \cos J \cos L
\end{bmatrix} \\
\mathbf{w}_8 &= \begin{bmatrix}
0 \\
- (m + j) \pi \sin L \\
\sin J \sin L \\
(m + j) \pi \cos J \sin L \\
- (m + j)(m + \ell) \pi^2 \cos L \\
(m + \ell) \pi \cos L \sin J \\
(m + j)(m + \ell) \pi^2 \cos J \cos L
\end{bmatrix}
\end{align*}
\]
\[
\begin{align*}
\mathbf{w}_3 &= \\
&= \begin{bmatrix}
0 \\
(m - \ell) \pi \sin J \\
\sin J \sin L \\
-(m - \ell) \pi \cos L \sin J \\
\end{bmatrix} \\
\mathbf{w}_4 &= \\
&= \begin{bmatrix}
0 \\
-(m - \ell) \pi \sin J \\
-\sin J \sin L \\
(m - \ell) \pi \cos L \sin J \\
\end{bmatrix}
\end{align*}
\]
\[ w_6 = \begin{bmatrix} 0 \\ (m + \ell) \pi \sin J \\ -\sin J \sin L \\ - (m + \ell) \pi \cos L \sin J \end{bmatrix}, \quad \begin{bmatrix} 0 \\ - (m + \ell) \pi \sin J \\ \sin J \sin L \\ (m + \ell) \pi \cos L \sin J \end{bmatrix} \]

\[ w_6 = \begin{bmatrix} 0 \\ - (m + j) (m + \ell) \pi^2 \cos J \\ (m + j) \pi \cos J \sin L \\ (m + j) (m + \ell) \pi^2 \cos J \cos L \end{bmatrix}, \quad \begin{bmatrix} 0 \\ - (m + j) (m + \ell) \pi^2 \cos J \\ (m + j) \pi \cos J \sin L \\ (m + j) (m + \ell) \pi^2 \cos J \cos L \end{bmatrix} \]

Here \( J = j \pi h \) and \( L = \ell \pi h \). We have assumed here (without loss of generality and by analogy to our assumption in the one-dimensional analysis) that \( x_k = y_k = 0 \) and \( x_1 = y_{k+1} = y_1 = h \).

The values \( d_k, k = 1, 2, \ldots, 8 \), that produce the eigenvector \( \mathbf{x} \) in (4.14) (or, equivalently, give the vector \( \mathbf{v} \) that satisfies (4.17), (4.18), (4.19), and (4.20)) are

\[
\begin{align*}
    d_3 &= (m + j)(m + \ell), &
    d_1 &= \frac{\ell}{m} (1 - c_2^j) - c_2^j - d_3 \\
    d_5 &= -(m + j)(m - \ell), &
    d_2 &= \frac{\ell}{m} (1 - c_2^j) - 1 - d_5 \\
    d_4 &= -(m - j)(m + \ell), &
    d_7 &= \frac{\ell}{m} c_2^j (1 - c_2^j) - c_2^j d_4 \\
    d_6 &= (m - j)(m - \ell), &
    d_8 &= \frac{\ell}{m} c_2^j (1 - c_2^j) - c_2^j - d_6 
\end{align*}
\]

Here \( c_2^j \) is the value of \( c_2 \) corresponding to the eigenvalue \( \lambda_j \) of the one-dimensional problem while...
$c^2_k$ is the value of $c_2$ corresponding to the eigenvalue $\lambda_4$ of the one-dimensional problem. With these choices of $d_k$, $k = 1, 2, \ldots 8$, the vector $v$ becomes

$$v = \begin{bmatrix}
0 \\
[ v_2^c ] \left[ (1 - c^2_2) \sin L \right] \\
\left[ (1 - c^2_2) \sin J \right] \left[ (1 - c^2_2) \sin L \right] \\
- v_2^c \cos J \left[ (1 - c^2_2) \sin L \right] \\
0 \\
[ v_3^c ] \left[ v_2^c \cos L \right] \\
\left[ (1 - c^2_2) \sin J \right] \left[ v_2^c \cos L \right] \\
- v_2^c \cos J \left[ v_2^c \cos L \right] \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\left[ v_2^c \right] \left[ - v_2^c \right] \\
\left[ (1 - c^2_2) \sin J \right] \left[ - v_2^c \right] \\
- v_2^c \cos J \left[ - v_2^c \right]
\end{bmatrix}, \quad (4.21)

where

$$v_2^c = \pi \left[ (m - j) + c^2_2 (m + j) \right]$$

$$v_3^c = \pi \left[ (m - \ell) + c^2_2 (m + \ell) \right].$$

(Note the analogy to (4.8) (for the one-dimensional problem).)

It is interesting to observe, given the definition of $v$ in (4.21), that the left sides of equations (4.17), (4.18), (4.19), (4.20), can be represented as linear combinations of

$$z_1^j = [- \beta - \lambda_j \chi] v_2 + \left[ \alpha - \lambda_j \left( \frac{1}{2} - \zeta \right) \right] v_3 + [- \gamma + \lambda_j \psi] v_4 = 0$$
\[ z_2^j = [\gamma - \lambda_j \psi] v_2 + \left[ -\alpha - \lambda_j \left( \frac{1}{2} + \zeta \right) \right] v_3 + [\beta + \lambda_j \chi] v_4 = 0 \]
\[ z_1^\ell = [-\beta - \lambda_\ell \chi] v_2 + \left[ \alpha - \lambda_\ell \left( \frac{1}{2} - \zeta \right) \right] v_3 + [-\gamma + \lambda_\ell \psi] v_4 = 0 \]
\[ z_2^\ell = [\gamma - \lambda_\ell \psi] v_2 + \left[ -\alpha - \lambda_\ell \left( \frac{1}{2} + \zeta \right) \right] v_3 + [\beta + \lambda_\ell \chi] v_4 = 0, \]

which are simply the left sides of (4.6) and (4.7) under the simplifications which result in (4.8) for eigenvalues \( \lambda_j \) and \( \lambda_\ell \). Specifically, we find:

\[
\text{left side of (4.17)} = \left[ \left( \frac{1}{2} + \zeta \right) (1 - c_2^j) \sin L + \chi v_2^j \cos L + \psi v_2^j \right] z_1^j + \left[ \left( \frac{1}{2} - \zeta \right) (1 - c_2^\ell) \sin J + \chi v_2^\ell \cos J + \psi v_2^\ell \right] z_2^\ell \]
\[
\text{left side of (4.18)} = \left[ \left( \frac{1}{2} + \zeta \right) (1 - c_2^j) \sin L + \chi v_2^j \cos L + \psi v_2^j \right] z_2^j + \left[ \left( \frac{1}{2} + \zeta \right) (1 - c_2^\ell) \sin J + \chi v_2^\ell \cos J + \psi v_2^\ell \right] z_2^\ell \]
\[
\text{left side of (4.19)} = \left[ \left( \frac{1}{2} - \zeta \right) (1 - c_2^j) \sin L + \chi v_2^j \cos L + \psi v_2^j \right] z_1^j + \left[ \left( \frac{1}{2} - \zeta \right) (1 - c_2^\ell) \sin J + \chi v_2^\ell \cos J + \psi v_2^\ell \right] z_1^\ell \]
\[
\text{left side of (4.20)} = \left[ \left( \frac{1}{2} - \zeta \right) (1 - c_2^j) \sin L + \chi v_2^j \cos L + \psi v_2^j \right] z_1^j + \left[ \left( \frac{1}{2} + \zeta \right) (1 - c_2^\ell) \sin J + \chi v_2^\ell \cos J + \psi v_2^\ell \right] z_1^\ell \]

Now that the eigenvalues and eigenvectors of the two-dimensional problem are fully described in terms of their one-dimensional counterparts, we may state our results:

**Theorem 4.7** Let \( \Lambda_j = \{ \text{eigenvalues of the one-dimensional problem given in Theorem 4.4} \} \). We know, from Theorem 4.4, that \( \Lambda_j \) contains \( 2m \) elements. Let \( \Lambda_\ell = \Lambda_j \). Then the \( 4m^2 \) eigenvalues \( \lambda \) of (4.14), which are all negative, are given by all possible sums of the form

\[
\lambda = \lambda_j + \lambda_\ell
\]

where \( \lambda_j \in \Lambda_j \) and \( \lambda_\ell \in \Lambda_\ell \).

Using Theorem 4.7 and Corollary 4.6, we obtain

**Corollary 4.8** For the two-dimensional problem, the eigenvalue of \( C^{-1}A \) of greatest magnitude is
\[ \lambda = \frac{-\sqrt{-165}}{\kappa + (1 - \kappa^2)}. \]
Chapter 5

Application to Multiphase Equations

In this chapter we apply our method of solution to the PDEs that model multiphase flow and contaminant transport in porous media. After giving the general governing mass-balance equations, their final form, and strategy of solution, we discuss the comparison of solving the multiphase equations via our method of solution to that of using GMRES/ILU, a popular method for the solution of nonsymmetric systems of linear algebraic equations.

5.1 The PDEs of multiphase flow and transport

We solve the multiphase equations as implemented in the NAPL software of Guarnaccia, Pinder, and Fishman [64]. In this simulator, we consider three fluid phases ($W =$ water; $N =$ NAPL (i.e., a non-aqueous phase liquid); and $G =$ gas), each consisting of (at most) three constituent species ($w =$ water, $n =$ NAPL, and $g =$ gas). Note the convention that capital letters indicate phases while lower case letters denote species.
The equation that gives the mass-balance for species $i$ in phase $\alpha$ in a porous medium is \[ \frac{\partial}{\partial t} (\varepsilon S_\alpha \rho_i^\alpha) + \nabla \cdot (\varepsilon S_\alpha \rho_i^\alpha \mathbf{v}^\alpha) - \nabla \cdot \left( \varepsilon S_\alpha \rho_i^\alpha \mathbf{D}^\alpha \nabla \left( \frac{\rho_i^\alpha}{\rho^\alpha} \right) \right) + \varepsilon S_\alpha \kappa_i^\alpha D^\alpha i = \rho_i^\alpha Q^\alpha + \bar{\rho}_i^\alpha \] (5.1)

where $\varepsilon$ is the porosity of the porous medium; $S_\alpha$ is the saturation of the $\alpha$-phase; $\rho_i^\alpha$ is the mass concentration of species $i$ in the $\alpha$-phase; $\mathbf{v}^\alpha$ is the mass average velocity vector of phase $\alpha$; $\rho^\alpha$ is the $\alpha$-phase mass density; $\mathbf{D}^\alpha$ is the dispersion coefficient of the $\alpha$-phase, a symmetric second-order tensor; $\kappa_i^\alpha$ is the decay coefficient for species $i$ in phase $\alpha$; $Q^\alpha$ is a point source or sink of $\alpha$-phase mass; and $\bar{\rho}_i^\alpha$ is the source or sink of mass of species $i$ in the $\alpha$-phase due to interphase mass exchange. We note that there are nine versions of (5.1) because for each of the three phases $\alpha = W, N, G$, we write three species equations $i = w, n, g$. These nine mass balance equations are all coupled in various ways and possess many non-linearities.

After introducing various simplifying assumptions, combining certain equations, incorporating relevant porous media physics, discretizing with respect to time, and linearizing (see [64] for details), we arrive at the final form of the multiphase equations. There are five of these, written in terms of known parameters and the five primary variables for which the equations are solved: $P^W$ = pressure of the water phase, $S_W$, $S_{Tw} = S_W + S_N$, $\rho_n^W$, and $\rho_n^G$. As we advance from time level $k$ to time level $k + 1$, the five equations are solved as described below. See [64] for additional detail.

We first solve an elliptic-type equation (the pressure equation)

\[ \nabla \cdot (q^T)^{(k+1)} = (Q^W + Q^N + Q^G)^{(k)} - \frac{(E_{n/W}^S)^{(k+1)*}}{\rho^sr} + (\cdot)^{(k+1)*} + (\cdot)^{(k+1)*} \]

for $P^{W(k+1)}$, where $q^T = q^W + q^N + q^G$ is the total fluid flux, $E_{n/W}^S$ defines mass-transfer due to adsorption from the water phase onto the porous medium, $\cdot^\alpha$ defines changes in $\alpha$-phase due to presence of NAPL species, and $\rho^{or}$ is the mass density of pure $\alpha$-phase. The time level $(k + 1)*$ refers to using the updated (i.e., $k + 1$) time level if available; otherwise using the old (i.e., $k$) time level.
The term \( P^{W(k+1)} \) is found in the phase flux expressions

\[
q^W = -\lambda^{(k+1)*}_W \left( \nabla P^{W(k+1)} - \gamma^{W(k+1)*} \nabla z \right)
\]

\[
q^N = -\lambda^{(k+1)*}_N \left( \nabla \left[ P^{W(k+1)} + P^{(k+1)*}_{cNW} \right] - \gamma^{N(k+1)*} \nabla z \right)
\]

\[
q^G = -\lambda^{(k+1)*}_G \left( \nabla \left[ P^{W(k+1)} + P^{(k+1)*}_{cNW} + P^{(k+1)*}_{aG} \right] - \gamma^{G(k+1)*} \nabla z \right)
\]

The parameter \( \lambda \) is the \( \alpha \)-phase mobility scalar while \( \gamma^\alpha \) is the specific weight of the \( \alpha \)-phase. The vector \( \nabla z \) is equal to \( \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \) while \( P_{\alpha\beta} \) is the capillary pressure on the interface between the \( \alpha \)- and \( \beta \)-phases.

After calculating other relevant parameters, we arrive at the two coupled time-discrete species transport equations:

\[
\begin{align*}
\varepsilon S^{(k)} & + \left( \frac{1 - \rho^{W(k+1,m)}}{\rho^{Nw}} \right) \rho^b K_d \left[ \frac{\rho_n^{W(k+1,m+1) - \rho_n^{W(k)}}}{\Delta t} \right] \\
& + \left[ \varepsilon S^{(k)} \frac{\rho^{W(k+1,m)}}{\rho^{W_r}} + \left( 1 - \frac{\rho^{W(k+1,m)}}{\rho^{Nw}} \right) \rho^b K_d \right] \kappa_n^W \rho^{W(k+1,m+1)} \\
& + q^{W(k+1)*} \cdot \nabla \rho_n^{W(k+1,m+1)} - \nabla \cdot \left[ \left( \varepsilon S^WD^W \right)^{(k+1)*} \nabla \rho_n^{W(k+1,m+1)} \right] \\
& = \left( \rho_n^W - \rho_n^{W(k+1,m+1)} \right) Q^{W(k)} + \left( 1 - \frac{\rho^{W(k+1,m)}}{\rho^{Nw}} \right) \\
& \left[ C^{W(k+1)*}_n \left( \rho_n^W - \rho_n^{W(k+1,m+1)} \right) \\
& - C^{G(k+1)*}_{n/W} \left( H \rho_n^{W(k+1,m+1)} - \rho_n^{G(k+1,m)*} \right) \right]
\end{align*}
\]

and

\[
\begin{align*}
\varepsilon S^{(k)} & \frac{\rho^{(k+1,m+1)}}{\rho^{G(k)}} + \left( \frac{\varepsilon S^{(k)} \rho^{G(k+1,m+1)}}{\rho^{Gr}} \right) \kappa_n^{G(k+1,m+1)} \\
& + q^{G(k+1)*} \cdot \nabla \rho_n^{G(k+1,m+1)} - \nabla \cdot \left[ \left( \varepsilon S^GD^G \right)^{(k+1)*} \nabla \rho_n^{G(k+1,m+1)} \right] \\
& = \left( \rho_n^G - \rho_n^{G(k+1,m+1)} \right) Q^{G(k)} + \left( 1 - \frac{\rho^{G(k+1,m)}}{\rho^{Gr}} \right) \\
& \left[ C^{G(k+1)*}_n \left( \rho_n^G - \rho_n^{G(k+1,m+1)} \right) - C^{G(k+1)*}_{n/W} \left( H \rho_n^{W(k+1,m)*} - \rho_n^{G(k+1,m)} \right) \right]
\end{align*}
\]
where $\rho^b$ is bulk density of the soil, $K_d$ is a distribution coefficient, $C^b_n$ is a coefficient that regulates the rate at which equilibrium of NAPL concentration is reached between the NAPL and $\alpha$-phases, $C^r_{n/W}$ is a coefficient that regulates the rate at which equilibrium of NAPL concentration is reached between the water and gas phases, $p^a_n$ is the equilibrium value for NAPL concentration between the NAPL and $\alpha$-phases, $H$ is the Henry’s law coefficient, and $\rho_n^{\alpha(k+1,m)} = \theta \rho_n^{\alpha(k+1,m)} + (1 - \theta) \rho_n^{\alpha(k+1,m-1)}, 0 \leq \theta \leq 2$.

Let us examine the details of solving (5.2), (5.3). The notation $\rho_n^{\alpha(k+1,m+1)}$ means the mass concentration of the NAPL species in the $\alpha$-phase at time step $k + 1$ and iteration number $m + 1$. The expression $\rho_n^{\alpha(k)}$ refers to the mass concentration of the NAPL species in the $\alpha$-phase at time step $k$; i.e., the value to which the iterates $\rho_n^{\alpha(k,m)}$ converged. At time step $k + 1$, we solve (5.2) for $\rho_n^{W(k+1,m+1)}$, the concentration of the NAPL species in the water phase at time step $k + 1$ and iteration number $m + 1$, using values of NAPL species concentration in the water phase at both the previous iteration and previous time step and the value of the NAPL species concentration in the gas phase at time step $k + 1$ and iteration number $m$. Similarly, we solve (5.3) for $\rho_n^{G(k+1,m+1)}$, the concentration of the NAPL species in the gas phase at time step $k + 1$ and iteration number $m + 1$, using values of NAPL species concentration in the gas phase at both the previous iteration and previous time step and the value of the NAPL species concentration in the water phase at time step $k + 1$ and iteration number $m$. This coupled iterative solution technique continues until convergence is reached, i.e., until the changes in concentration from one iteration to the next are sufficiently minuscule.

Once this convergence is attained, i.e., we have values for $\rho_n^{W(k+1)}$ and $\rho_n^{G(k+1)}$, we perform some calculations, eventually arriving at two coupled time-discrete phase saturation equations:

$$\frac{\zeta}{\Delta t} \left( \delta S^{(k+1,m+1)}_W - S^{(k+1,m)}_W \right) + \nabla \cdot \left( J^{W(k+1,m)} \nabla \lambda_N \right) + \nabla \cdot \left( f^{W(k+1,m)} \nabla \lambda_N \right)$$

$$\left[ \nabla P^{(k+1,m)}_{cNW} + \left( \frac{dP_{cNW}}{dS_W} \right)^{(k+1,m)} \nabla \delta S^{(k+1,m+1)}_W + \Delta \gamma^{W(k+1,m+1)}_W \nabla \delta S^{(k+1,m+1)}_W \right]$$

$$+ \nabla \cdot \left( J^{W(k+1,m)} \nabla \lambda_G \right) \left[ \nabla \left( P^{(k+1,m)}_{cNW} + P^{(k+1,m)}_{cG} \right) + \Delta \gamma^{W(k+1,m+1)}_W \nabla \delta S^{(k+1,m+1)}_W \right]$$

(5.4)
where we solve for the iterative increment \( \delta S^{(k+1,m+1)}_W = S^{(k+1,m+1)}_W - S^{(k+1,m)}_W \) and where the fractional flow function is \( f^\alpha = \frac{\lambda_\alpha}{\lambda_W + \lambda_N + \lambda_G}. \) \( \Delta \gamma_{a\beta} = \gamma^a - \gamma^\beta, \) \( \sigma_{a\beta} \) is the interfacial tension between the \( \alpha \)- and \( \beta \)-phases, \( E^a_n \) is a term that describes the mass-exchange of NAPL across the interface between the NAPL and \( \alpha \)-phases, and \( E^G_{n/W} \) is a term that describes the mass-exchange of NAPL across the interface between the water and gas phases.

Once (5.4) and (5.5) are solved, we update:

\[
S^{(k+1,m+1)}_W = S^{(k+1,m)}_W - \delta S^{(k+1,m+1)}_W
\]
\[
S^{(k+1,m+1)}_{Tw} = S^{(k+1,m)}_{Tw} - \delta S^{(k+1,m+1)}_{Tw}
\]

We continue to iterate until \( \delta S^{(k+1,m+1)}_W \) and \( \delta S^{(k+1,m+1)}_{Tw} \) are sufficiently small.

To summarize, at each time step, we solve five equations for the five primary variables \( P^W, S_W, S_{Tw}, \rho^W_n, \) and \( \rho^G_n \). The equation for \( P^W \) is solved directly while the equations for the coupled parameters \( \rho^W_n \) and \( \rho^G_n \) and for the coupled parameters \( S_W \) and \( S_{Tw} \) are solved iteratively. It is
worth noting that the pressure equation is independent of time step size. Also, should the scheme for the solution of $\rho_n^W$ and $\rho_n^G$ (respectively, $S_w$ and $S_{Tw}$) not converge in fewer than a prescribed maximum number of iterations, the time step size is reduced and the iterations begin again with the new time step size. It is therefore possible that many restarts and commensurate reductions in time step size may occur before convergence is actually reached at any particular time step.

The multiphase equations are discretized by the method described above in Section 1.1.3. Each of the primary variables is approximated using a Hermite interpolating polynomial analogous to (1.11). The other spatially varying parameters in the multiphase equations are approximated similarly or employ a bi-linear Lagrange interpolating polynomial (see [64] for details).

5.2 Results

In the NAPL software, each of the five matrix equations (pressure equation, two species transport equations, two phase saturation equations) is solved using the DSLUGM [23] routine, which uses the GMRES [114] method with an incomplete LU preconditioner. To compare the efficacy of the DSLUGM routine and the Bi-CGSTAB method with our preconditioner, we solved the multiphase equations using three example problems discussed in [64]. We augmented the NAPL code as follows. Every time DSLUGM (with restart parameter of 20, see [114] for explanation) solved a system (1.13), we then had Bi-CGSTAB (with our preconditioner) solve the identical system. The simulations were driven by the DSLUGM solution of systems $Ax = b$ with the Bi-CGSTAB solutions computed merely for the purpose of making comparisons. Thus, at each time step, both solvers used identical initial conditions and used the same value for the converged solution of the previous time step. Both methods of solution were timed on a serial computer.

In the discussion in previous chapters, all analysis was done on equations with symmetric differential operators, i.e., no first derivative terms were present. However, there are many first derivative terms present in the multiphase equations and thus first derivatives of the Hermite basis functions are used in the collocation discretization of the multiphase equations.

In the figures that illustrate the results, the solution of each pressure equation is indicated by
Table 5.1: Results from Experiment 1. $D =$ results for DSLUGM. $B =$ results for Bi-GCSTAB.

<table>
<thead>
<tr>
<th></th>
<th>$D$ [secs]</th>
<th>$B$ [secs]</th>
<th>$\frac{D}{B}$</th>
<th>$D \leq B$</th>
<th>$D &gt; B$</th>
<th>equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>part 1</td>
<td>657.1325</td>
<td>644.6898</td>
<td>1.0193</td>
<td>81</td>
<td>164</td>
<td>pressure</td>
</tr>
<tr>
<td></td>
<td>314.6103</td>
<td>156.0698</td>
<td>2.0158</td>
<td>2</td>
<td>344</td>
<td>(5.4)</td>
</tr>
<tr>
<td></td>
<td>402.6298</td>
<td>378.9687</td>
<td>1.0624</td>
<td>41</td>
<td>106</td>
<td>pressure</td>
</tr>
<tr>
<td></td>
<td>365.8161</td>
<td>178.1669</td>
<td>2.0532</td>
<td>0</td>
<td>485</td>
<td>(5.4)</td>
</tr>
<tr>
<td>part 2</td>
<td>325.6655</td>
<td>114.7578</td>
<td>2.8379</td>
<td>0</td>
<td>485</td>
<td>(5.5)</td>
</tr>
<tr>
<td></td>
<td>567.3365</td>
<td>572.4922</td>
<td>0.9910</td>
<td>105</td>
<td>99</td>
<td>pressure</td>
</tr>
<tr>
<td></td>
<td>287.1743</td>
<td>147.1151</td>
<td>1.9520</td>
<td>0</td>
<td>374</td>
<td>(5.4)</td>
</tr>
<tr>
<td>part 3</td>
<td>265.7106</td>
<td>107.3698</td>
<td>2.4747</td>
<td>0</td>
<td>374</td>
<td>(5.5)</td>
</tr>
</tbody>
</table>

an asterisk, (5.4) by a circle, (5.5) by a ‘plus’ sign, (5.2) by an ‘x’, and (5.3) by a square. We note for Experiments 1 and 2 that the NAPL is assumed to be immiscible in both water and air, which obviates the necessity for solving the species transport equations (5.2) and (5.3). Each experiment was run three times and the results given in Tables 5.1, 5.2, 5.3, 5.4, and 5.5 report means of the three simulations. Figures 5.1, 5.2, 5.3, 5.4, and 5.5 depict one of the representative runs for each example.

5.2.1 Experiment 1: LNAPL spill

The physical problem represented here is given in [132] and [133]. The details of its implementation using the NAPL code are given in [64]. The domain is discretized on a $26 \times 18$ grid of finite elements. Briefly, from time $= 0$ to 63000 seconds (part 1), a quasi-steady-state is reached by allowing an initially saturated sand to drain. Since no NAPL has been introduced yet, the only equations solved at each time step are the pressure equation and (5.4). In part 2, from time $= 63000$ to 64120 seconds, an LNAPL (i.e., a NAPL that is lighter than water) source is applied. At each time step we thus solve the pressure equation, (5.4), and (5.5). In part 3, from time $= 64120$ to 66000 seconds, the LNAPL source is discontinued and the system is allowed to redistribute and we thus solve the pressure equation, (5.4), and (5.5) at each time step. The results are summarized in Table 5.1.

The second (respectively, third) column gives the time (in seconds) required for DSLUGM (respectively, Bi-CGSTAB with our preconditioner) to solve all the equations of the specified type for the specified part of the experiment. The fourth column gives the ratio of the second and third
columns. The fifth column gives the total number of times DSLUGM solved a system (1.13) of the specified equation type for the specified part of the experiment more quickly than did Bi-CGSTAB (with our preconditioner). The sixth column gives the total number of times Bi-CGSTAB (with our preconditioner) solved a system (1.13) more quickly than did DSLUGM. The seventh column indicates which equation (i.e., the pressure equation, equation (5.2), equation (5.3), equation (5.4), or equation (5.5)) is being considered.

For each time step in Experiment 1, we compare DSLUGM with Bi-CGSTAB (with our preconditioner) in Figure 5.1. The horizontal axis gives the elapsed simulated time while the vertical axis gives the “improvement” \( \frac{D - B}{D} \) of using Bi-CGSTAB \( B \) (with our preconditioner) instead of DSLUGM \( D \) for each time a system (1.13) is solved. Clearly, if the “improvement” is zero, then Bi-CGSTAB and DSLUGM require equal amounts of time to converge. Positive values of “improvement” represent the superiority of Bi-CGSTAB while negative values represent its inferiority.

### 5.2.2 Experiment 2: DNAPL spill

The physical problem represented here is an artificial aquifer experiment conducted by M. Fishman. The details of its implementation using the \textit{NAPL} code are given in [64]. The domain is discretized on a \( 35 \times 16 \) grid of finite elements. Briefly, from time \( t = 0 \) to \( 5000 \) seconds (part 1), a quasi-steady-state is reached after lowering the phreatic surface of an initially saturated sand. Since no NAPL has been introduced yet, the only equations solved at each time step are the pressure equation and (5.4). In part 2, from time \( t = 5000 \) to \( 5143 \) seconds, a source of PCE, a DNAPL (i.e., a NAPL that is heavier than water), is applied. At each time step we thus solve the pressure equation, (5.4), and (5.5). In part 3, from time \( t = 5143 \) to \( 8693 \) seconds, the PCE source is discontinued and the system is allowed to return to equilibrium and we thus solve the pressure equation, (5.4), and (5.5) at each time step. The results are summarized in Table 5.2.

As we did above for Experiment 1, we compare DSLUGM with Bi-CGSTAB (with our preconditioner) in Figure 5.2 for each time step of Experiment 2.
Figure 5.1: “Improvement” solving each linear system in Experiment 1 using Bi-CGSTAB with our preconditioner. The solution of each pressure equation is indicated by an asterisk, (5.4) by a circle, and (5.5) by a ‘plus’ sign.

5.2.3 Experiment 3: DNAPL vapor transport

This experiment is derived from one found in [89] whose NAPL implementation is discussed in [64]. The domain is discretized on a $20 \times 20$ grid of finite elements. We make herein the change of eliminating the water table condition found in [89]. As the experiment begins, the water saturation profile of a sand column is in static equilibrium. A container filled with sand and residual TCE is placed at the top of the center of the sand column to act as a contaminant vapor source. We ran this experiment for time $= 0$ to 33200 seconds. The results are summarized in Table 5.3 and in Figure 5.3.
Table 5.2: Results from Experiment 2. $D = $ results for DSLUGM. $B = $ results for Bi-GCSTAB.

<table>
<thead>
<tr>
<th></th>
<th>$D$ [secs]</th>
<th>$B$ [secs]</th>
<th>$\frac{D}{B}$</th>
<th>$D \leq B$</th>
<th>$D &gt; B$</th>
<th>equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>part 1</td>
<td>631.6191</td>
<td>661.2326</td>
<td>0.9552</td>
<td>140</td>
<td>37</td>
<td>pressure</td>
</tr>
<tr>
<td></td>
<td>342.7492</td>
<td>192.1339</td>
<td>1.7839</td>
<td>1</td>
<td>276</td>
<td>(5.4)</td>
</tr>
<tr>
<td></td>
<td>419.3771</td>
<td>379.5926</td>
<td>1.1048</td>
<td>32</td>
<td>77</td>
<td>pressure</td>
</tr>
<tr>
<td></td>
<td>552.1290</td>
<td>259.8333</td>
<td>2.1249</td>
<td>0</td>
<td>464</td>
<td>(5.4)</td>
</tr>
<tr>
<td>part 2</td>
<td>484.9935</td>
<td>184.8823</td>
<td>2.6233</td>
<td>0</td>
<td>463</td>
<td>(5.5)</td>
</tr>
<tr>
<td></td>
<td>3448.8271</td>
<td>2312.8549</td>
<td>1.4912</td>
<td>59</td>
<td>674</td>
<td>pressure</td>
</tr>
<tr>
<td></td>
<td>1359.4818</td>
<td>606.2629</td>
<td>2.2424</td>
<td>0</td>
<td>1064</td>
<td>(5.4)</td>
</tr>
<tr>
<td>part 3</td>
<td>1319.4750</td>
<td>528.9365</td>
<td>2.4946</td>
<td>0</td>
<td>1064</td>
<td>(5.5)</td>
</tr>
</tbody>
</table>

Table 5.3: Results from Experiment 3. $D = $ results for DSLUGM. $B = $ results for Bi-GCSTAB.

<table>
<thead>
<tr>
<th></th>
<th>$D$ [secs]</th>
<th>$B$ [secs]</th>
<th>$\frac{D}{B}$</th>
<th>$D \leq B$</th>
<th>$D &gt; B$</th>
<th>equation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>234.3932</td>
<td>121.5503</td>
<td>1.9284</td>
<td>0</td>
<td>96</td>
<td>pressure</td>
</tr>
<tr>
<td></td>
<td>142.9025</td>
<td>56.0014</td>
<td>2.5518</td>
<td>0</td>
<td>262</td>
<td>(5.2)</td>
</tr>
<tr>
<td></td>
<td>185.0255</td>
<td>71.0515</td>
<td>2.6041</td>
<td>0</td>
<td>256</td>
<td>(5.3)</td>
</tr>
<tr>
<td></td>
<td>52.0629</td>
<td>29.4106</td>
<td>1.3210</td>
<td>0</td>
<td>96</td>
<td>(5.4)</td>
</tr>
<tr>
<td></td>
<td>51.8614</td>
<td>37.8091</td>
<td>1.3717</td>
<td>0</td>
<td>96</td>
<td>(5.5)</td>
</tr>
</tbody>
</table>

5.3 Interpretation of results

If we examine Tables 5.1, 5.2, and 5.3, we see that we achieve faster convergence almost always if we use Bi-CGSTAB (with our preconditioner) as compared to DSLUGM. When DSLUGM is superior, it is always for the pressure equation and quicker by negligible amounts (Experiment 1 part 3 and Experiment 2 part 1). Furthermore, Bi-CGSTAB is often more than twice as fast as is DSLUGM.

In examining Figures 5.1, 5.2, and 5.3, we find that “improvement” is often a strong function of both equation type and physical experiment. For example, in all three parts of Experiment 1, the values for “improvement” for equation (5.4) are almost always between 0.4 and 0.6. For Experiment 2, however, a greater range of values for “improvement” for the same equation prevails. Also, the results for Experiment 3 reveal, for example, that “improvement” for equation (5.2) is almost always either around 0.42 or 0.67. Indeed, the value is around 0.42 for the final iteration of (5.2) and around 0.67 for all other iterations of (5.2).

5.4 Comparison with GMRES with our preconditioner

Above we have compared Bi-CGSTAB with our preconditioner to GMRES with an incomplete $LU$ preconditioner, determining that the former method is often much better than the latter. A
reasonable question to ask is: How well would the GMRES method with our preconditioner fare? To investigate this question, we directly compared DSLUGM (i.e., GMRES/ILU) and GMRES with our preconditioner on the NAPL code. That is, we replaced Bi-CGSTAB (with our preconditioner, which we used in the investigations above) with GMRES (with our preconditioner). In all cases, we used a restart parameter of 20 (see [114]). To be consistent with the notation established above, “D” in Figures 5.4 and 5.5 and Tables 5.4 and 5.5 refers to DSLUGM (i.e., GMRES/ILU) while “B” refers to GMRES with our preconditioner.

The results obtained by running Experiment 1 are given in Table 5.4 and Figure 5.4. We see that we obtain modest improvement using GMRES with our preconditioner when solving equations (5.4) and (5.5) in parts 2 and 3. However, it is clear that solving the pressure equation using our preconditioner with GMRES is not competitive with DSLUGM.
The results obtained by running Experiment 2 are given in Table 5.5 and Figure 5.5. The results are very similar to those found in Experiment 1. That is, we obtain modest improvement using GMRES with our preconditioner when solving equations (5.4) and (5.5) in parts 2 and 3. However, it is again clear that solving the pressure equation using our preconditioner with GMRES is not competitive with DSLUGM.

Results from Experiment 3 were unattainable due to the fact that GMRES with our preconditioner failed to converge, causing the program to “crash” after one time step.

From this investigation we conclude that of the three methods considered to solve the multiphase equations, Bi-CGSTAB with our preconditioner is best. Next best is GMRES/ILU and GMRES with our preconditioner is worst.

It is interesting to compare the column for “$D$” in Tables 5.1 and 5.4 (in which Experiment 1
Table 5.4: Results from Experiment 1. $D =$ results for DSLUGM. $B =$ results for GMRES with our preconditioner.

<table>
<thead>
<tr>
<th></th>
<th>$D$ [secs]</th>
<th>$B$ [secs]</th>
<th>$\frac{D}{B}$</th>
<th>$D \leq B$</th>
<th>$D &gt; B$</th>
<th>equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>part 1</td>
<td>1060.1144</td>
<td>2891.2661</td>
<td>0.3667</td>
<td>287</td>
<td>0</td>
<td>(5.4)</td>
</tr>
<tr>
<td></td>
<td>541.0131</td>
<td>954.8163</td>
<td>0.5666</td>
<td>407</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>part 2</td>
<td>572.3725</td>
<td>953.3862</td>
<td>0.6004</td>
<td>144</td>
<td>4</td>
<td>(5.4)</td>
</tr>
<tr>
<td></td>
<td>516.3103</td>
<td>379.8497</td>
<td>1.3592</td>
<td>4</td>
<td>486</td>
<td></td>
</tr>
<tr>
<td></td>
<td>456.4720</td>
<td>388.2302</td>
<td>1.1758</td>
<td>92</td>
<td>398</td>
<td>(5.5)</td>
</tr>
</tbody>
</table>

was performed) and to compare the same column for Tables 5.2 and 5.5 (in which Experiment 2 was performed). We would expect that each set of two “$D$” columns contain close to the same values, but this is not the case. A possible explanation is that the computer running the simulations was engaged in other activities at the same time the $NAPL$ program was running, thus affecting memory allocation and, as a result, the time required to complete a $NAPL$ run.

Table 5.5: Results from Experiment 2. $D =$ results for DSLUGM. $B =$ results for GMRES with our preconditioner.

<table>
<thead>
<tr>
<th></th>
<th>$D$ [secs]</th>
<th>$B$ [secs]</th>
<th>$\frac{D}{B}$</th>
<th>$D \leq B$</th>
<th>$D &gt; B$</th>
<th>equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>part 1</td>
<td>589.9852</td>
<td>1461.8340</td>
<td>0.4036</td>
<td>180</td>
<td>1</td>
<td>(5.4)</td>
</tr>
<tr>
<td></td>
<td>342.6088</td>
<td>399.6560</td>
<td>0.8573</td>
<td>214</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>part 2</td>
<td>244.3559</td>
<td>288.2187</td>
<td>0.8478</td>
<td>52</td>
<td>33</td>
<td>pressure</td>
</tr>
<tr>
<td></td>
<td>373.5165</td>
<td>205.4265</td>
<td>1.8182</td>
<td>8</td>
<td>369</td>
<td>(5.4)</td>
</tr>
<tr>
<td></td>
<td>313.7364</td>
<td>199.2116</td>
<td>1.5749</td>
<td>3</td>
<td>373</td>
<td>(5.5)</td>
</tr>
<tr>
<td>part 3</td>
<td>2496.4634</td>
<td>3626.7754</td>
<td>0.6883</td>
<td>637</td>
<td>9</td>
<td>pressure</td>
</tr>
<tr>
<td></td>
<td>1129.7620</td>
<td>934.6742</td>
<td>1.2087</td>
<td>131</td>
<td>985</td>
<td>(5.4)</td>
</tr>
<tr>
<td></td>
<td>1087.4347</td>
<td>851.9601</td>
<td>1.2764</td>
<td>149</td>
<td>967</td>
<td>(5.5)</td>
</tr>
</tbody>
</table>
Figure 5.4: “Improvement” solving each linear system in Experiment 1 using GMRES with our preconditioner. The solution of each pressure equation is indicated by an asterisk, (5.4) by a circle, and (5.5) by a ‘plus’ sign.
Figure 5.5: “Improvement” solving each linear system in Experiment 2 using GMRES with our preconditioner. The solution of each pressure equation is indicated by an asterisk, (5.4) by a circle, and (5.5) by a ‘plus’ sign.
Chapter 6

Parallel Implementation

In this section, we discuss the implementation of the Bi-CGSTAB method with our preconditioner on the Cray T3E, a parallel processing supercomputer. After describing how each entry of $A$, $x$, and $b$ is assigned to its particular processor, we briefly discuss the issue of which parallel processing architecture should be chosen. We next detail the parallel implementation of each operation of the Bi-CGSTAB algorithm. We then give results showing that for sufficiently large problems, we attain almost linear speedup. Finally, we study the scalability of the algorithm via its isoefficiency function.

For simplicity, we will assume that $m_y$, the number of finite elements in the $y$-direction, is a multiple of $p$, the number of processors.
6.1 The allocation of portions of $A$, $x$, and $b$ to the various processors

Throughout the discussion, assume we have $p = 4$ processors; call them $\tilde{P}_1, \tilde{P}_2, \tilde{P}_3, \tilde{P}_4$. For the case where $m_y = 16$, we have $Ax = b$, where

$$A = \begin{bmatrix}
\tilde{A}_F \\
\tilde{A}_1 \\
\tilde{A}_3 \\
\tilde{A}_5 \\
\tilde{A}_7 \\
\tilde{A}_0 \\
\tilde{A}_{11} \\
\tilde{A}_{13} \\
\tilde{A}_L \\
\end{bmatrix}, \quad
\begin{bmatrix}
\tilde{B}_F \\
\tilde{B}_1 \\
\tilde{B}_3 \\
\tilde{B}_5 \\
\tilde{B}_7 \\
\tilde{B}_0 \\
\tilde{B}_{10} \\
\tilde{B}_{12} \\
\tilde{B}_L \\
\end{bmatrix}, \quad
\begin{bmatrix}
\tilde{C}_F \\
\tilde{C}_1 \\
\tilde{C}_3 \\
\tilde{C}_5 \\
\tilde{C}_7 \\
\tilde{C}_0 \\
\tilde{C}_{11} \\
\tilde{C}_{13} \\
\tilde{C}_L \\
\end{bmatrix}$$

$$x = \begin{bmatrix}
\tilde{x}_F \\
\tilde{x}_1 \\
\tilde{x}_3 \\
\tilde{x}_5 \\
\tilde{x}_7 \\
\tilde{x}_0 \\
\tilde{x}_{11} \\
\tilde{x}_{13} \\
\tilde{x}_L \\
\end{bmatrix}, \quad
\begin{bmatrix}
\tilde{y}_0 \\
\tilde{y}_2 \\
\tilde{y}_4 \\
\tilde{y}_6 \\
\tilde{y}_8 \\
\tilde{y}_{10} \\
\tilde{y}_{12} \\
\tilde{y}_{14} \\
\end{bmatrix}^T,$$

and

$$b = \begin{bmatrix}
\tilde{b}_F \\
\tilde{b}_1 \\
\tilde{b}_3 \\
\tilde{b}_5 \\
\tilde{b}_7 \\
\tilde{b}_0 \\
\tilde{b}_{11} \\
\tilde{b}_{13} \\
\tilde{b}_L \\
\end{bmatrix}, \quad
\begin{bmatrix}
\tilde{y}_0 \\
\tilde{y}_2 \\
\tilde{y}_4 \\
\tilde{y}_6 \\
\tilde{y}_8 \\
\tilde{y}_{10} \\
\tilde{y}_{12} \\
\tilde{y}_{14} \\
\end{bmatrix}^T.$$
The four types of decoration used above (i.e., the tilde, circumflex, bar, and brace) are used to indicate to which processor each portion of $A$, $x$, and $b$ is assigned. For example, the blocks $A_i$, $B_i$, $C_i$, $x_i$, and $b_i$ for $i = 3, 4, 5, 6$ all reside on the same processor, namely $\widehat{P_2}$, distinguished by the decoration of the circumflex. Note that each block in its entirety is assigned to a particular processor and that each processor is responsible for approximately one-fourth (in general, one-pth) of all the information in $A$, $x$, and $b$. In the language of parallel processing, we say that the allocation is balanced.

6.2 Choice of architecture

At this point we remark about the architecture of the parallel processing machine on which we want to implement our method. We should think of each processor as an independent computer with independent memory that is wired to other independent processors so that processors can pass messages to each other. As will be discussed in detail below, with one exception that will be noted, the Bi-CGSTAB algorithm with our preconditioner is particularly well-suited to one of the most simple architectures, namely the ring. Because rings are easily mapped onto other more complicated architectures [82] such as meshes or hypercubes, our algorithm is appropriate for most parallel supercomputers. However, the choice of a parallel computer with a tree architecture would be less than ideal because of the absence of a closed loop in a tree.

In a ring architecture, each processor is directly connected to two adjacent neighbors. Continuing with our example of $p = 4$ processors, the schematic of a ring architecture is:

\[
\begin{align*}
\widehat{P}_1 & \leftrightarrow \widehat{P}_2 \\
\uparrow & \uparrow \\
\widehat{P}_4 & \leftrightarrow \widehat{P}_3
\end{align*}
\]

In this example, processor $\widehat{P}_2$ can communicate directly with only processors $\widehat{P}_1$ and $\widehat{P}_3$. If processor $\widehat{P}_2$ should need to communicate with processor $\widehat{P}_4$, it must send the message through either $\widehat{P}_1$ or
6.3 Parallel implementation of preconditioned Bi-CGSTAB

In this section, we discuss the parallel implementation of all of the steps in the Bi-CGSTAB iteration. They can be divided into five basic operations: saxpy, scalar arithmetic, vector inner product, psolve, and matrix-vector multiplication. We describe each one below.

6.3.1 Saxpy

Saxpy is an acronym for “scalar a x plus y”. That is, given, the scalar \( a \) and vectors \( x \) and \( y \), we compute \( z = ax + y \). We assume that the value of scalar \( a \) resides on all processors and that vectors \( x \) and \( y \) are partitioned among the four processors. If, for example, vectors \( x \) and \( y \) contain 12 elements each and we have \( p = 4 \) processors, then the saxpy operation can be schematically represented as

\[
\begin{array}{c|c}
\tilde{P}_1 & \bar{a}\bar{x}_1 + \bar{y}_1 \to \bar{z}_1 \\
 & \bar{a}\bar{x}_2 + \bar{y}_2 \to \bar{z}_2 \\
 & \bar{a}\bar{x}_3 + \bar{y}_3 \to \bar{z}_3 \\
\tilde{P}_2 & \hat{a}\hat{x}_4 + \hat{y}_4 \to \hat{z}_4 \\
 & \hat{a}\hat{x}_5 + \hat{y}_5 \to \hat{z}_5 \\
 & \hat{a}\hat{x}_6 + \hat{y}_6 \to \hat{z}_6 \\
\tilde{P}_3 & \bar{a}\bar{x}_7 + \bar{y}_7 \to \bar{z}_7 \\
 & \bar{a}\bar{x}_8 + \bar{y}_8 \to \bar{z}_8 \\
 & \bar{a}\bar{x}_9 + \bar{y}_9 \to \bar{z}_9 \\
\tilde{P}_4 & \hat{a}\hat{x}_{10} + \hat{y}_{10} \to \hat{z}_{10} \\
 & \hat{a}\hat{x}_{11} + \hat{y}_{11} \to \hat{z}_{11} \\
 & \hat{a}\hat{x}_{12} + \hat{y}_{12} \to \hat{z}_{12} \\
\end{array}
\]

Two points are worth making. The first is to emphasize that the scalar \( a \) must reside on all of the processors. The second is to notice that the parallel implementation of the saxpy operation
requires only one-\textsuperscript{th} the amount of time compared to its serial implementation. In this example, implemented serially, the single processor would have to perform 12 separate operations of the form $ax + y \rightarrow z$. If we use four processors, each of them needs to perform only three $ax + y \rightarrow z$ operations. Assuming that each processor can perform its three $ax + y \rightarrow z$ operations at the same time that the other processors perform their three $ax + y \rightarrow z$ operations, it is clear that only $\frac{1}{4}$ the amount of time is required for the parallel implementation.

### 6.3.2 Scalar arithmetic

As we noted above for the saxp y operation, it is necessary for the scalar $a$ to reside on all processors. Therefore, each processor has to compute $a$ separately prior to the beginning of each saxp y operation. (In a serial implementation, $a$ needs to be computed only once.) Although we are performing the computation of $a$ on each processor, these redundant computations are occurring simultaneously. So even though the parallel implementation requires more work in some cases (e.g., the redundant computations of $a$), no extra time is being spent as a result of these redundancies.

### 6.3.3 Vector inner products

The inner (or dot) product of two vectors $x$ and $y$ is an operation that requires more inter-processor communication than is ideal for the “nearest neighbor” ring architecture. As above, we assume $x$ and $y$ contain 12 elements each. The first step, which requires no inter-processor communication, is given schematically as

\[ \begin{array}{c|l}
\hat{P}_1 & \widehat{x_1y_1} + \widehat{x_2y_2} + \widehat{x_3y_3} \rightarrow \hat{\sigma}_1 \\
\hat{P}_2 & \widehat{x_4y_4} + \widehat{x_5y_5} + \widehat{x_6y_6} \rightarrow \hat{\sigma}_2 \\
\hat{P}_3 & \widehat{x_7y_7} + \widehat{x_8y_8} + \widehat{x_9y_9} \rightarrow \hat{\sigma}_3 \\
\hat{P}_4 & \widehat{x_{10}y_{10}} + \widehat{x_{11}y_{11}} + \widehat{x_{12}y_{12}} \rightarrow \hat{\sigma}_4 
\end{array} \]

At this point, each processor $\hat{P}_i$ knows the value of its corresponding $\sigma_i$. However, to complete the computation of the inner product, it is necessary for each processor to know the value of $\sum_{i=1}^{4} \sigma_i$. 

97
The process by which this summation occurs, an example of a “reduction” operation, requires each processor to exchange information with processors other than its nearest neighbors.

However, in our code we utilized a “shared memory” routine available on the Cray T3E called “shmem_double_sum_to_all” which performs the necessary “reduction” operation very rapidly. Thus the inter-processor communication required by the computation of vector inner products is done with extreme efficiency.

6.3.4 Psolve

The goal of the “psolve” operation is to solve the preconditioned linear system

$$Py = c,$$

which we write in block form as

$$
\begin{bmatrix}
R & \\
L & B
\end{bmatrix}
\begin{bmatrix}
y_R \\
y_B
\end{bmatrix}
= 
\begin{bmatrix}
c_R \\
c_B
\end{bmatrix}.
$$
For the example \( m_y = 16 \), we have

\[
P = \begin{bmatrix}
\tilde{A}_0 \\
\tilde{A}_1 \\
\tilde{A}_3 \\
\tilde{A}_5 \\
\tilde{A}_7 \\
\tilde{A}_9 \\
\tilde{A}_{11} \\
\tilde{A}_{13} \\
\end{bmatrix}
\]
We first solve $Ry_R = c_R$, which we write as

$$
\begin{pmatrix}
\tilde{A}_F \\
\tilde{A}_1 \\
\tilde{A}_3 \\
\tilde{A}_5 \\
\tilde{A}_7 \\
\tilde{A}_9 \\
\tilde{A}_{11} \\
\tilde{A}_{13} \\
\tilde{A}_L
\end{pmatrix}
\begin{pmatrix}
\tilde{y}_F \\
\tilde{y}_1 \\
\tilde{y}_3 \\
\tilde{y}_5 \\
\tilde{y}_7 \\
\tilde{y}_9 \\
\tilde{y}_{11} \\
\tilde{y}_{13} \\
\tilde{y}_L
\end{pmatrix}
= 
\begin{pmatrix}
\tilde{c}_F \\
\tilde{c}_1 \\
\tilde{c}_3 \\
\tilde{c}_5 \\
\tilde{c}_7 \\
\tilde{c}_9 \\
\tilde{c}_{11} \\
\tilde{c}_{13} \\
\tilde{c}_L
\end{pmatrix}.
$$

Because the non-zero entries of $R$ reside only in its diagonal blocks $A_i$, solving each subsystem

$$A_i y_i = c_i \tag{6.1}$$

is an independent task. The parallelism is clear. Processor $P_2$ solves the subsystems (6.1) for $i = 3, 5$ while processor $P_3$ solves the subsystems (6.1) for $i = 7, 9$, etc. Even though processor $P_1$ needs to solve three subsystems (6.1) for $i = F, 1, L$, the subsystems for $i = F, L$ are only half the size as those whose subscript is a number. Thus the job of solving all the subsystems (6.1) is well-balanced among all the processors. Note also that no inter-processor communication occurs at this step.

Once we have obtained $y_R$, we next find $y_B$ by solving

$$By_B = c_B - Ly_R = c_B',$$

However, there seems to be a mistake. In the given equation, $c_B'$ is not properly defined. It should be $c_B - Ly_R = c_B'$, where $c_B'$ is the updated right-hand side after solving the first set of equations.
which we write

\[
\begin{bmatrix}
\tilde{A}_0 \\
\tilde{A}_2 \\
\tilde{A}_4 \\
\tilde{A}_6 \\
\tilde{A}_8 \\
\tilde{A}_{10} \\
\tilde{A}_{12} \\
\tilde{A}_{14}
\end{bmatrix}
= \begin{bmatrix}
c_0 \\
c_2 \\
c_4 \\
c_6 \\
c_8 \\
c_{10} \\
c_{12} \\
c_{14}
\end{bmatrix}
- \begin{bmatrix}
\tilde{C}_F & \tilde{B}_0 \\
\tilde{C}_1 & \tilde{B}_2 \\
\tilde{C}_3 & \tilde{B}_4 \\
\tilde{C}_5 & \tilde{B}_5 \\
\tilde{C}_7 & \tilde{B}_8 \\
\tilde{C}_9 & \tilde{B}_{10} \\
\tilde{C}_{11} & \tilde{B}_{12} \\
\tilde{C}_{13} & \tilde{B}_L
\end{bmatrix}
\begin{bmatrix}
\tilde{y}_F \\
\tilde{y}_1 \\
\tilde{y}_3 \\
\tilde{y}_5 \\
\tilde{y}_7 \\
\tilde{y}_9 \\
\tilde{y}_{11} \\
\tilde{y}_{13}
\end{bmatrix}
\tag{6.2}
\]

Again, solving any subsystem \(A_i y_i = c_i^*\) is an independent task and these tasks are distributed evenly among all the processors. However, we require some inter-processor communication for the computation of \(L y_R\). To illustrate this, let us focus on the portion of work done in (6.2) by processor \(P_2\):

\[
\begin{bmatrix}
\tilde{A}_4 \\
\tilde{A}_6
\end{bmatrix}
\begin{bmatrix}
\tilde{y}_4 \\
\tilde{y}_6
\end{bmatrix}
= \begin{bmatrix}
c_4 \\
c_6
\end{bmatrix}
- \begin{bmatrix}
\tilde{C}_3 & \tilde{B}_4 \\
\tilde{C}_5 & \tilde{B}_5
\end{bmatrix}
\begin{bmatrix}
\tilde{y}_3 \\
\tilde{y}_5 \\
\tilde{y}_7
\end{bmatrix}
\tag{6.3}
\]

The computation of \(\tilde{y}_4\) is easily accomplished since all relevant blocks (\(\tilde{A}_4, c_4, \tilde{C}_3, \tilde{B}_4, \tilde{y}_3, \text{and } \tilde{y}_5\) )
reside on processor \( \hat{P}_2 \). However, before computing \( \hat{y}_6 \), processor \( \hat{P}_2 \) must receive \( \hat{y}_7 \) from processor \( \hat{P}_3 \). Likewise, processor \( \hat{P}_2 \) must send \( \hat{y}_3 \) to processor \( \hat{P}_1 \). So, as part of the “psolve” operation, each processor must both send and receive a portion of a vector. Note that this inter-processor communication occurs only between nearest neighbors in the ring architecture.

### 6.3.5 Matrix-vector multiplication

As noted above in our discussion of the Bi-CGSTAB algorithm, our preconditioner allows for a very efficient computation of the matrix-vector multiplication \( \mathbf{v} = A \mathbf{y} \) (see (1.22)). We need only perform the multiplication \( U \mathbf{y}_B \). Note, however, that this operation requires some inter-processor communication. Let \( q_R \) be the product \( U \mathbf{y}_B \). We write

\[
q_R = U \mathbf{y}_B
\]

as

\[
\begin{bmatrix}
\hat{q}_F \\
\hat{q}_1 \\
\hat{q}_3 \\
\hat{q}_5 \\
\hat{q}_7 \\
\hat{q}_9 \\
\hat{q}_{11} \\
\hat{q}_{13} \\
\hat{q}_L
\end{bmatrix}
= \begin{bmatrix}
\hat{B}_F \\
\hat{C}_0 & \hat{B}_1 \\
\hat{C}_2 & \hat{B}_3 \\
\hat{C}_4 & \hat{B}_5 \\
\hat{C}_6 & \hat{B}_7 \\
\hat{C}_8 & \hat{B}_9 \\
\hat{C}_{10} & \hat{B}_{11} \\
\hat{C}_{12} & \hat{B}_{13} \\
\hat{C}_L
\end{bmatrix}\begin{bmatrix}
\hat{y}_0 \\
\hat{y}_2 \\
\hat{y}_4 \\
\hat{y}_6 \\
\hat{y}_8 \\
\hat{y}_{10} \\
\hat{y}_{12} \\
\hat{y}_{14}
\end{bmatrix}
\]
Again, let us focus on the portion of this task done by processor $P_2$, which at the conclusion of this operation, needs to know the values of $\tilde{q}_6$ and $\tilde{q}_8$:

\[
\begin{bmatrix}
\tilde{q}_6 \\
\tilde{q}_8 \\
\tilde{q}_7
\end{bmatrix}
= \begin{bmatrix}
\tilde{C}_2 & \tilde{B}_3 \\
\tilde{C}_4 & \tilde{B}_6 \\
\tilde{C}_6 & B_7
\end{bmatrix}
\begin{bmatrix}
\tilde{y}_2 \\
\tilde{y}_4 \\
\tilde{y}_6 \\
\tilde{y}_8
\end{bmatrix}
\]

First, processor $P_2$ performs the multiplication $\tilde{C}_6\tilde{y}_6$ and sends the result to processor $P_3$, where $\tilde{C}_6\tilde{y}_6$ is used in the computation of $\tilde{q}_7$. Then processor $P_2$ determines $\tilde{q}_6 = \tilde{C}_4\tilde{y}_4 + \tilde{B}_6\tilde{y}_6$, the computation of which requires no interprocessor communication, since all relevant blocks reside on processor $P_2$. Finally, to compute $\tilde{q}_8$, processor $P_2$ performs the multiplication $\tilde{B}_3\tilde{y}_4$ and adds it to the result of the multiplication $\tilde{C}_2\tilde{y}_2$, which it receives from processor $P_1$. So we see that for the operation of the matrix-vector multiplication, each processor must both send and receive a portion of a vector and that this inter-processor communication occurs only between nearest neighbors in the ring architecture.

### 6.4 Implementation on the Cray T3E

We implemented the Bi-CGSTAB algorithm on a Cray T3E, a supercomputer that can take full advantage of the parallelism inherent in Bi-CGSTAB and in our preconditioner. We explore two measures of the efficacy of our parallel algorithm: the speedup and scalability. (We should note that the times reported below include only the “main loop” portion of the Bi-CGSTAB algorithm which is, of course, where the bulk of the computations occur.)
6.4.1 Speedup

Let $T_S =$ serial run time $= \text{time required to solve a problem using a single processor}$. Let $T_P =$ parallel run time $= \text{time required to solve a problem using parallel processing}$. We define speedup as

$$S = \text{speedup} = \frac{T_S}{T_P}$$

Ideally, we would like $S = p$, a phenomenon known as linear speedup. However, in practice, we observe only sub-linear speedup due to parallel overhead (i.e., manifestations of parallel algorithms that are absent in serial algorithms), the main sources of which are [82]:

- inter-processor communication,
- load imbalance, and
- extra computation.

We saw in our discussion above that inter-processor communication occurs in three separate operations of the Bi-CGSTAB iteration: “psolve”, matrix-vector multiplication, and in the reduction portion of vector inner product computation. Fortuitously, the inter-processor communication is the only significant source contributing to sub-linear speedup. The load is balanced as well as possible (i.e., processors do not sit idle waiting for messages from other processors before they can resume performing useful computations) and although extra computation does occur (e.g., each processor needs to compute the value of the scalar “a” for the saxpy operation), these computations always occur simultaneously across all the processors.

When testing our code on the Cray T3E, we selected a particular PDE with Dirichlet boundary conditions and allowed the number of finite elements in the $x$-direction ($m_x$) and the number of finite elements in the $y$-direction ($m_y$) to (independently) assume the values 16, 32, 64, and 128. We ran the code using $p = 1, 2, 4, 8, 16, \text{and } 32$ processors. For each parameter set $\{m_x, m_y, p\}$, we ran the code five times and then took the mean of these five times as the representative time used to compile the results given in Tables 6.1 and 6.2.

As can be seen in Table 6.1, we achieve only modest speedup for relatively small problem sizes.
We achieve the best results for the largest problem, i.e., when \( m_x = m_y = 128 \), which corresponds to solving a linear system of 65536 equations in 65536 unknowns. For this example, i.e., when \( m_x = m_y = 128 \), we depict the results graphically in Figure 6.1, where we see that our speedup curve stays remarkably close to the ideal linear speedup curve. This phenomenon of *almost linear speedup* shows how effective parallel processing can be for solving large linear systems such as ours.

### 6.4.2 Scalability and the isoeficiency function

The *efficiency* \( E \) of a parallel system measures the fraction of time for which a processor is usefully employed [82]. It is defined as:

\[
E = \frac{S}{p}
\]

Just as the unattainable ideal for speedup is \( S = p \), the ideal efficiency is \( E = 1 \). Our results for efficiency, computable from Table 6.1, are given in Table 6.2.

A natural question to ask is: Can the efficiency of a parallel system remain fixed if we change both the number of processors and the problem size? If the answer is yes, then the parallel system is said to be *scalable*. Here the problem size is governed by the parameters \( m_x \) and \( m_y \).

To measure the scalability of parallel systems, the isoeficiency function is often used. If we let

<table>
<thead>
<tr>
<th>Table 6.1: Speedup results</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m_x )</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>128</td>
</tr>
<tr>
<td>128</td>
</tr>
<tr>
<td>128</td>
</tr>
<tr>
<td>128</td>
</tr>
</tbody>
</table>


105
Figure 6.1: Speedup curve for $m_x = m_y = 128$. The upper curve represents the unattainable ideal of linear speedup.

Let $W$ be the problem size (which can also represent $T_s$, the serial run time) and let

$$T_o = pT_p - W$$

be the parallel overhead function (i.e., the difference between “parallel problem size” and serial problem size), then various algebraic manipulations (see [82] for details) lead to the equation

$$W = KT_o(W, p),$$

where $K = \frac{p}{1 - E}$. If we can solve (6.5) for $W$ as a function of $p$, we call the result $W(p)$ the isoefficiency function of the parallel system. The isoefficiency function tells us how fast we must increase $W$ as a function of $p$ to maintain a fixed efficiency. We will derive the isoefficiency function
### Table 6.2: Efficiency results

<table>
<thead>
<tr>
<th>$m_x$</th>
<th>$m_y$</th>
<th>$p = 1$</th>
<th>$p = 2$</th>
<th>$p = 4$</th>
<th>$p = 8$</th>
<th>$p = 16$</th>
<th>$p = 32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>16</td>
<td>1.0000</td>
<td>0.6053</td>
<td>0.3455</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>1.0000</td>
<td>0.6940</td>
<td>0.4579</td>
<td>0.2652</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>1.0000</td>
<td>0.8186</td>
<td>0.5675</td>
<td>0.3935</td>
<td>0.2341</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>128</td>
<td>1.0000</td>
<td>0.8891</td>
<td>0.7069</td>
<td>0.5473</td>
<td>0.3481</td>
<td>0.2012</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>1.0000</td>
<td>0.7289</td>
<td>0.4743</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>1.0000</td>
<td>0.7900</td>
<td>0.5905</td>
<td>0.3873</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>64</td>
<td>1.0000</td>
<td>0.8204</td>
<td>0.6898</td>
<td>0.5332</td>
<td>0.3488</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>128</td>
<td>1.0000</td>
<td>0.9366</td>
<td>0.8411</td>
<td>0.7011</td>
<td>0.5265</td>
<td>0.3543</td>
</tr>
<tr>
<td>64</td>
<td>16</td>
<td>1.0000</td>
<td>0.7914</td>
<td>0.5767</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>1.0000</td>
<td>0.8821</td>
<td>0.7311</td>
<td>0.5368</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>1.0000</td>
<td>0.9354</td>
<td>0.8518</td>
<td>0.7041</td>
<td>0.5147</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>1.0000</td>
<td>0.9529</td>
<td>0.8447</td>
<td>0.8235</td>
<td>0.6356</td>
<td>0.5052</td>
</tr>
<tr>
<td>128</td>
<td>16</td>
<td>1.0000</td>
<td>0.8753</td>
<td>0.7174</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>32</td>
<td>1.0000</td>
<td>0.9390</td>
<td>0.8526</td>
<td>0.7058</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>64</td>
<td>1.0000</td>
<td>0.9692</td>
<td>0.9131</td>
<td>0.8371</td>
<td>0.6938</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>1.0000</td>
<td>0.9858</td>
<td>0.9709</td>
<td>0.9203</td>
<td>0.8005</td>
<td>0.6879</td>
</tr>
</tbody>
</table>

### Table 6.3: Computations required for a Bi-CGSTAB iteration

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_i$</td>
<td>$F^T r_{i-1}$</td>
<td>1 inner product</td>
</tr>
<tr>
<td>$\beta_i$</td>
<td>$\rho_{i-1} = \omega_{i-1} v_{i-1}$</td>
<td>3 computations</td>
</tr>
<tr>
<td>$p_i$</td>
<td>$r_{i-1} + \beta_i (p_{i-1} - \omega_{i-1} v_{i-1})$</td>
<td>2 saxpy</td>
</tr>
<tr>
<td></td>
<td>solve for $y_i$: $Py_i = p_i$</td>
<td>psolve</td>
</tr>
<tr>
<td>$v_i$</td>
<td>$A y_i$</td>
<td>matrix-vector multiplication</td>
</tr>
<tr>
<td>$\alpha_i$</td>
<td>$\rho_i / v_i$</td>
<td>inner product</td>
</tr>
<tr>
<td>$s_i$</td>
<td>$r_{i-1} - \alpha_i v_i$</td>
<td>saxpy</td>
</tr>
<tr>
<td></td>
<td>solve for $z_i$: $Pz_i = s_i$</td>
<td>psolve</td>
</tr>
<tr>
<td>$t_i$</td>
<td>$A z_i$</td>
<td>matrix-vector multiplication</td>
</tr>
<tr>
<td>$\omega_i$</td>
<td>$\omega_i = t_i^T s_i / t_i^T t_i$</td>
<td>2 inner products</td>
</tr>
<tr>
<td>$x_i$</td>
<td>$x_{i-1} + \alpha_i y_i + \omega_i z_i$</td>
<td>2 saxpy</td>
</tr>
<tr>
<td>$r_i$</td>
<td>$s_i - \omega_i t_i$</td>
<td>saxpy</td>
</tr>
<tr>
<td></td>
<td>if $|r_i|_2$ is small enough, then quit</td>
<td>inner product</td>
</tr>
</tbody>
</table>

for the implementation of the Bi-CGSTAB iteration with our preconditioner on the Cray T3E and show that it correlates well to the data given in Table 6.2.

Let us recall the Bi-CGSTAB iteration (see Table 6.3). It is evident that at each iteration we must perform 5 individual computations, 6 saxpy operations, 5 inner products, 2 “psolves”, and 2 matrix-vector multiplications. We summarize below the computation counts for each operation.

We must first introduce an important parameter. Because matrix $A$ and all the vectors used in the Bi-CGSTAB algorithm are partitioned among the various processors, we require a parameter to describe this. We define

$$b = \frac{m_y}{p},$$
which is easily seen to be the number of finite elements in the $y$-direction assigned to each processor.

The importance of $b$ will become apparent in our discussion below. Let us also define $t_c$ to be the time required for a processor of the Cray T3E to perform one arithmetic operation.

**Saxpy**

Because the vectors $x$ and $y$ in the saxpy operation each contain $4m_x m_y$ entries, each processor is responsible for $4m_x b$ entries from each vector. Two computations per entry of the resultant vector $z = ax + y$ are required: the multiplication $ax_i$ and the addition $ax_i + y_i$. It is therefore apparent that during the saxpy operation, each processor performs $8m_x b = \frac{8}{p} m_x m_y$ operations, requiring time

$$t_{saxpy} = \frac{8}{p} m_x m_y t_c.$$

**Inner product**

There are two steps in the parallel inner product of $x^T y$: the computation and the reduction. As above, each vector contains $4m_x b$ entries and corresponding to each of these entries is a multiplication $x_i y_i$ and then an addition where the product $x_i y_i$ is added to a variable that eventually equals the sum $\sum_{i=1}^{b} x_i y_i$. Therefore, the computational portion of the parallel inner product is the same as that for saxpy: $\frac{8}{p} m_x m_y t_c$. If we let $t_r$ be the time required to perform the reduction necessary to complete the inner product computation, then we see that during an inner product operation, each processor consumes time

$$t_{inner \ product} = \frac{8}{p} m_x m_y t_c + t_r.$$

**Psolve**

The “psolve” operation requires both computation and communication. The computation time is seen to be $\frac{1}{p}(312m_x m_y - 136m_y)$. For the communication, it is customary to define two more parameters. Let $t_s$ be the “startup time”, i.e., the time consumed by a processor to prepare a message to be sent to another processor [82]. Let $t_w$ be the “per-word transfer time”, i.e., the time required for one “word” of data to traverse the link between a sending processor and a receiving
processor [82]. We will assume that one “word” is one matrix or vector entry. As we saw above in the description of the “psolve” operation, each processor must send and receive data to/from a neighboring processor, requiring time $t_s + 4m_w t_w$. Thus each “psolve” operation takes time

$$t_{\text{psolve}} = t_s + 4m_w t_w + \frac{1}{p}(312m_s m_y - 136m_y).$$

Matrix-vector multiplication

The operation of matrix-vector multiplication also requires both computation and communication. The computational part takes time $\frac{1}{p}(32m_s m_y - 16m_y)$ while the communication time is $t_s + 2m_w t_w$. Thus the matrix-vector multiplication requires time

$$t_{\text{matvecmult}} = t_s + 2m_w t_w + \frac{1}{p}(32m_s m_y - 16m_y)$$

Isoefficiency analysis

To make the problem tractable, let $m = m_s = m_y$ and now let $n = 4m^2$. Note that $n$ is the number of equations and unknowns in the matrix equation $Ax = b$. Then the quantities above become

$$t_{\text{saxpy}} = 2t_c \frac{n}{p} = O\left(\frac{n}{p}\right)$$

$$t_{\text{inner product}} = t_r + 2t_c \frac{n}{p} = t_r + O\left(\frac{n}{p}\right)$$

$$t_{\text{psolve}} = t_s + 2\sqrt{m_w} + O\left(\frac{n}{p}\right)$$

$$t_{\text{matvecmult}} = t_s + \sqrt{m_w} + O\left(\frac{n}{p}\right).$$

Then, combining these expressions, we obtain

$$T_P = t_c' \left(\frac{n}{p}\right) + 5t_r + 2\left(t_s + 2\sqrt{m_w}\right) + 2\left(t_s + \sqrt{m_w}\right),$$

(6.6)
where \( t'_c \) is the constant associated with the order operator \( \mathcal{O} \left( \frac{n}{p} \right) \). Since we use a fast reduction algorithm available on the Cray T3E, we take \( t_r \ll 1 \), and then (6.6) becomes

\[
T_P = t'_c \left( \frac{n}{p} \right) + 4t_s + 6\sqrt{n}t_w. \tag{6.7}
\]

If we have only one processor, then there is no communication cost and (6.7) reduces to

\[
W = T_1 = \mathcal{O}(n). \tag{6.8}
\]

We now introduce (6.4) into (6.5), obtaining

\[
W = K (pT_P - W),
\]

or equivalently

\[
(K + 1) W = pT_P. \tag{6.9}
\]

Substitution of (6.8) and (6.7) into (6.9) yields

\[
W = \mathcal{O}(n) = nt'_c + 4pt_s + 6p\sqrt{nt_w}. \tag{6.10}
\]

We now consider each term on the right side of (6.10) independently. Using only \( nt'_c \), we see that (6.10) becomes \( \mathcal{O}(n) = \mathcal{O}(n) \), which while certainly true, does not provide useful information. Using only \( 4pt_s \), we see that (6.10) becomes \( \mathcal{O}(n) = \mathcal{O}(p) \). Using only \( 6p\sqrt{nt_w} \), we have

\[
\mathcal{O}(n) = 6p\sqrt{nt_w}
\]

from which we obtain

\[
\mathcal{O} \left( n^{\frac{2}{3}} \right) = \mathcal{O}(p)
\]
Table 6.4: Example of data correlating to the theoretical isoefficiency function

<table>
<thead>
<tr>
<th>$m_x$</th>
<th>$m_y$</th>
<th>$p$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>64</td>
<td>4</td>
<td>0.3675</td>
</tr>
<tr>
<td>16</td>
<td>128</td>
<td>8</td>
<td>0.5473</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>4</td>
<td>0.5905</td>
</tr>
<tr>
<td>32</td>
<td>64</td>
<td>8</td>
<td>0.5532</td>
</tr>
<tr>
<td>32</td>
<td>128</td>
<td>16</td>
<td>0.5265</td>
</tr>
<tr>
<td>64</td>
<td>16</td>
<td>4</td>
<td>0.5767</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>8</td>
<td>0.5368</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>16</td>
<td>0.5147</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>32</td>
<td>0.5052</td>
</tr>
</tbody>
</table>

and thus

$$O(n) = O(p^2).$$

Selecting the largest of these results, we write

$$W = O(n) = O(p^2).$$

(6.11)

This is the isoefficiency function for our parallel system. Recalling that $n = O(m^2)$, we interpret (6.11) in the following manner. If we double the number of finite elements in one coordinate direction, we must also double the number of processors in order to maintain a fixed efficiency.

We see that this theoretical estimate is mirrored in Table 6.2. For example, consider the entries in Table 6.4, which are taken directly from Table 6.2. For all the entries in Table 6.4, we have

$$\frac{m_x m_y}{p} = \text{constant},$$

and the corresponding values of the efficiency $E$ are all approximately equal.
Chapter 7

Literature Review

In this section we summarize the scientific literature pertinent to the areas covered in this work. We will focus on three topical areas: the Bi-CGSTAB method, collocation, and application of collocation to solving subsurface multiphase flow and transport problems.

7.1 Bi-CGSTAB

All of the methods considered in our discussion below of conjugate gradient methods to solve the \( n \times n \) system of linear algebraic equations

\[
Ax = b
\]  

are so-called “Krylov subspace” methods. A \textit{Krylov subspace} is defined

\[
K(B,s,k) = \text{span} \{ s, Bs, \ldots, B^{k-1}s \}
\]
for a given matrix $B$, vector $s$, and positive integer $k$. In Krylov subspace methods, iterates $x_1, x_2, \ldots$ are generated from an initial guess $x_0$ such that

$$x_j \in x_0 + \mathcal{K}(B, s, j),$$

where $B$ is a function of $A$ and $s$ is a function of $A$, $x_0$, and $b$. For each $j$, the vector $x_j$ usually minimizes some vector norm over $x_0 + \mathcal{K}(B, s, j)$ or produces a residual $r_j = b - Ax_j$ that is orthogonal to another Krylov subspace [57].

The first application of a “conjugate gradient” method to solve (7.1) is apparently due to Hestenes and Stiefel [71] in 1952. In their work, the matrix $A$ is restricted to be symmetric and positive definite (SPD). Also, their conjugate gradient method was viewed as a direct (as opposed to iterative) method, as it gives the solution $x$ in at most $n$ steps (assuming exact arithmetic). However, because of roundoff error, the method often fails to produce $x$ in $n$ steps, and thus the “conjugate gradient” method was not considered to be competitive with other well-known direct methods, such as Gaussian elimination.

The “conjugate gradient” method enjoyed a revival beginning in 1971 with papers by Reid [108], [109] in which he recognized that the “conjugate gradient” method could be considered to be iterative, i.e., converging to $x$ in, perhaps, many fewer than $n$ steps. Reid’s papers still required $A$ to be SPD. However, once the “conjugate gradient” method was appreciated as a truly iterative method, it has since been a major topic of research.

The next major development in conjugate gradient history was the realization that conjugate gradient convergence could be vastly accelerated by what became known as “preconditioning”. In a nutshell, we solve

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

instead of (7.1), where $P$ is a “preconditioning matrix” or “preconditioner”. Clearly, (7.1) and (7.2) have the same solution $x$. The basic idea is that if $P \approx A$, then $P^{-1}A \approx I$, the identity matrix. Thus the left side of (7.2) is approximately $x$, and thus few iterations should be necessary
to produce the true value of $x$ from its initial guess. The earliest papers that discuss preconditioning include Axelsson (1974) [6] and Concus et al. (1976) [36]. Once the concept of acceleration due to preconditioning was firmly established, preconditioned conjugate gradient methods have been among the most popular for solving systems (7.1).

Up to this point, conjugate gradient methods still required that the matrix $A$ be SPD. Considerable work then commenced to generalize the conjugate gradient methods to be applicable to non-symmetric and indefinite linear systems. A fairly straightforward generalization of the “classical” conjugate gradient method that is applicable to non-SPD linear systems is the Bi-Conjugate Gradient (Bi-CG) method, introduced by Fletcher [51] in 1975. While sharing several properties with the “classical” conjugate gradient method, most notably producing the exact solution in at most $n$ steps in exact arithmetic, it nonetheless is unattractive due its tendency to converge to $x$ in a rather unsmooth manner, even with the use of an effective preconditioner. Also, in Bi-CG, it is necessary to perform computations using $A^T$, resulting in a sequence of vectors $\{f_0, f_1, \ldots\}$ whose convergence to $0$ is not exploited. Finally, the possibility exists for Bi-CG to “break down” due to values of zero in the denominator of certain fractions generated by the method. By contrast, this cannot occur in the “classical” conjugate gradient method, due to the fact that $A$ (in that case) is positive definite.

A much more attractive method was introduced by Saad and Schultz [114] in 1986. Their GMRES (generalized minimum residual) method has several attractive properties shared with the “classical” conjugate gradient method: the iterates $x_m$ minimize $\|r\|_2 = \|b - Ax\|_2$ and “break down” cannot occur. Because the method converges monotonically and is robust, GMRES has become very popular and has been studied extensively. One of the drawbacks of the method is that if many iterations are required for convergence, the Krylov subspace information must be periodically “thrown away” and then restarted due to memory and computational considerations (see [114] for a complete discussion). Another drawback is that GMRES is not as readily parallelizable as the other methods discussed here.

The method we have chosen to focus on in this work, the Bi-CGSTAB (Bi-Conjugate Gradient
— Stabilized) method of van der Vorst [128], was introduced in 1992. The motivation of this work was to obtain a smoothly converging variant of Bi-CG that also did not suffer from the “irregular convergence behavior” [128] of the Conjugate Gradients-Squared (CGS) method of Sonneveld (1980) [124], a method that is “often competitive with . . . GMRES” [128].

Although Bi-CGSTAB is theoretically susceptible to “break down” due to values of zero in the denominator of certain fractions generated by the method, we have never seen an instance of this occurring throughout all the numerical experiments we conducted when researching this work. Furthermore, the method does not require reference to $A^T$ as did Bi-CG and is much more easily parallelizable than GMRES. While the method does possess a minimization property (see [128] for details) which results in relatively smooth convergence, the convergence is not monotone.

Bi-CGSTAB has been extensively studied, applied, and generalized since it made its appearance on the linear solver scene. It has proven itself to be a popular method for solving a wide-ranging variety of problems, particularly in fluid dynamics. In a series of four papers (1996) [137], (1996) [138], (1997) [139], and (1997) [140], Wille uses Bi-CGSTAB exclusively to solve the linear systems arising from Navier-Stokes equations. Other papers in which Bi-CGSTAB is the method of choice for solving Navier-Stokes linear systems are Dahl and Wille (1992) [39], Clift and Forsyth (1994) [35], Stella and Buccignani (1996) [125], and Forsyth and Jiang (1997) [52]. Also, Jiang and Forsyth (1995) [77] use Bi-CGSTAB to solve linear systems arising from transonic Euler equations, as do Dijkstra et al. (1995) [43] to study non-parallel steady flows and their linear stability. Brachtendorf et al. (1995) [19] and (1995) [20] use Bi-CGSTAB in their simulator modelling steady-state electronic circuits as do Chen and Yang (1998) [31] to solve the two-dimensional neutron and photon transport equation. Diaz and Shenoi (1995) [42] use Bi-CGSTAB to solve the well equations used in oil reservoir simulation.

A number of authors have made comparisons between Bi-CGSTAB and other methods. In all cases, Bi-CGSTAB more than holds its own. Chin and Forsyth (1993) [32] compare GMRES and Bi-CGSTAB on the linear systems arising from incompressible Navier-Stokes problems. They report that Bi-CGSTAB is “faster (sometimes by factors of two or three) and more robust than GMRES for
two-dimensional Navier-Stokes problems.” Hriberšek et al. (1993) [75] compared three conjugate-gradient-type methods (Bi-CG, CGS, and Bi-CGSTAB) and Gauss elimination on linear systems arising from the “boundary-domain integral method” applied to equations that model viscous flow. They report that the conjugate gradient methods perform much better than Gauss elimination and that among the conjugate gradient methods, “CGS and Bi-CGSTAB are to be preferred because they reduce the error in the fastest and smoothest way.” Chuang and Chieng (1995) [34] compare Bi-CGSTAB to three other methods for solving compressible Navier-Stokes linear systems, concluding that Bi-CGSTAB is the best of the methods they study. Peters (1993) [102] compares seven nonsymmetric linear solvers in the context of solving the advection-dispersion equation and reports that Bi-CGSTAB is competitive with all other methods studied. Schmid (1995) [116] compares Bi-CGSTAB, CGS, GMRES, and the “Broyden type secant method GB” to linear systems arising from the simulation of semiconductor processing. He reports that Bi-CGSTAB and CGS are most effective in terms of accuracy, CPU time, and number of iterations. Vulk (1993) [135] compares his variant of GMRES, called GMRESR, with three other methods, including Bi-CGSTAB, in the context of incompressible Navier-Stokes equations. For most problems considered, Bi-CGSTAB is at least competitive with GMRES and GMRESR. Joo and Downar (1995) [79] compare Bi-CGSTAB with two other methods in the context of a problem in nuclear reactor simulation. They report that the cyclic Chebyshev semi-iterative method is superior for most of their problems. However, they do state that “the rate of convergence of Bi-CGSTAB was generally invariant with changes in the coefficient matrix and exhibited superior performance for ‘near critical’ problems.” They also note that “of the three methods examined, Bi-CGSTAB most easily lends itself to parallel processing”. Van der Vorst, the creator of Bi-CGSTAB, has also authored two survey papers (1995) [129] and (1996) [130] in which he discusses interrelationships between the commonly used conjugate gradient methods for nonsymmetric linear systems, including Bi-CGSTAB.

While the papers mentioned above focus on application of Bi-CGSTAB (and other methods), work has also been done on the analysis of the method. Brezinski et al. (1995) [21], Cao (1997) [25], and Chan and Szeto (1996) [30] discuss strategies to anticipate and thus avoid the potential
“breakdown” of Bi-CGSTAB. Sleijpen and van der Vorst (1996) [121] report that the computed residuals in Bi-CGSTAB and its variants may be inaccurate and discuss remediation strategies.

A number of “offshoots” of Bi-CGSTAB have been developed. Gutknecht (1993) [65], who proposes a variant of Bi-CGSTAB, called BiCGSTAB2, which is designed specifically to accommodate matrices with complex eigenvalues. In a different generalization, Sleijpen and Fokkema (1993) [119] introduce the Bi-CGSTAB(ℓ) method, also for matrices with complex eigenvalues, demonstrating significant improvement over Gutknecht’s BiCGSTAB2 method. Bi-CGSTAB(ℓ) is studied further in Sleijpen et al. (1994) [122]. Sleijpen and van der Vorst (1995) [120] discuss CGS, Bi-CGSTAB, and Bi-CGSTAB(ℓ) in the context of their superiority to Bi-CG and in the context of look-ahead strategies to determine appropriate values for ℓ in Bi-CGSTAB(ℓ). Chan et al. (1994) [29] apply a so-called “quasi-minimal residual” (QMR) method to Bi-CGSTAB, creating the QMRCGSTAB method, which results in obtaining a method that converges more smoothly than does Bi-CGSTAB.

Other authors who have made contributions to the theory of Bi-CGSTAB include Zhang (1997) [144], who proposes a unified theory that encompasses several variants of Bi-CG, namely CGS, Bi-CGSTAB, and Bi-CGSTAB2. Montero and Suarez (1995) [94] discuss a so-called “left-right” preconditioning technique which they report is superior to other preconditioners they tested.

Much work has been done concerning the parallel implementation of Bi-CGSTAB. Scarbnick et al. (1992) [115] discuss MP-PCGPAK2, a software package that implements parallel versions of preconditioned conjugate gradient methods, including Bi-CGSTAB, for MIMD (multiple instruction, multiple data) machines. Another parallel software package, PARASPAR, discussed by Zlatev and Wasniewski (1994) [145], focuses on methods for sparse linear systems, including Bi-CGSTAB, other conjugate gradient methods, and direct methods. Doi and Hoshi (1992) [44] study the vector implementation of conjugate gradient methods with modified incomplete LU (MILU) preconditioner to solve linear systems arising from the three-dimensional finite difference 7-point form. Their experiments, carried out on a SX-3/14 supercomputer, indicate that large-number multicolor MILU/Bi-CGSTAB converges more quickly than than small-number multicolor MILU/Bi-CGSTAB and the “hyperplane” MILU/Bi-CGSTAB methods. Carlenzoli and Gervasio (1992) [27] study the vector-
ization properties of both direct methods and conjugate gradient methods, including Bi-CGSTAB. Muramatsu et al. discuss the parallelization of the linear systems arising from the incompressible Navier-Stokes equations in two papers (1994) [97] and (1995) [98]. They compare several linear solvers, including a multigrid version of preconditioned Bi-CGSTAB, which they report is as much as four times as fast as its competitors. Bi-CGSTAB (with block incomplete $LU$ preconditioner) is used exclusively to parallelize the solution of linear systems arising from incompressible Navier-Stokes equations in the papers by Bucchignani and Stella (1996) [24] and by Kurreck and Wittig (1997) [83]. Shadid and Tuminaro (1994) [117] compare several Krylov subspace methods, including Bi-CGSTAB, in the context of their parallel implementation in solving PDEs on large-scale MIMD machines. Kanschatt (1995) [80] uses Bi-CGSTAB to parallelize the solution of the radiative transfer equation. Van der Vorst (1996) [131] gives an overview of the issues involved in parallelizing conjugate gradient methods, including Bi-CGSTAB. Kulkarni et al. (1996) [81] introduce what they call a “dishonest” preconditioner in their paper concerning the parallelization of various conjugate gradient methods, including Bi-CGSTAB. Amadio and Mazzia (1996) [5] discuss the parallel implementation of preconditioned Bi-CGSTAB in the context solving two-step boundary value methods for initial value problems. Pena et al. (1997) [101] introduce a new method for the parallel implementation of Bi-CGSTAB for the three-dimensional steady-state simulation for semiconductor devices.

7.2 Collocation

Although we have used the term “collocation” in a specific way throughout this work, a more general formulation is possible. Suppose we wish to numerically solve an ODE

$$\mathcal{L}[u](x) = H(x).$$
The “method of collocation” proceeds by defining a function \( \hat{u}(x) \) to approximate \( u(x) \) and enforcing

\[
\mathcal{L}[\hat{u}](c_i) = H(c_i)
\]

for distinct “collocation points” \( c_i \). In our work, we have always chosen \( \hat{u} \) to be piecewise cubic (in the one-dimensional case) or piecewise bi-cubic (in the two-dimensional case). Given a piecewise defined approximating function \( \hat{u} \), a natural choice for collocation point location (in terms of accuracy) is to coincide with the points of Gaussian quadrature, or, equivalently, at the roots of the Legendre polynomial of appropriate degree. Some authors studying collocation have made choices similar to ours while others have not. In the discussion below, we will refer to the selection of Hermite cubic (or bi-cubic) basis functions and collocation points at the Gauss points as “orthogonal collocation”, a choice that is echoed in the literature.

In his 1983 Ph.D. dissertation [28], Celia gives a history of collocation methods. We will mention the highlights of his discussion below and then focus on a post-1983 review of pertinent literature.

In 1934, Slater [118], while studying energy bands in metals, was apparently the first to use the general concepts of collocation. The idea of using roots of polynomials as choices for collocation points was introduced by Lanczos (1938) [87]. He used Chebyshev polynomials instead of the now standard Legendre polynomials.

Applications of collocation became prominent, particularly in the chemical engineering literature, starting with the 1967 paper of Villadsen and Stewart [134]. They considered how to choose the collocation points, suggesting using the roots of orthogonal polynomials.

Up to this point, the approximating function \( \hat{u} \) had been selected to have a single definition over the entire domain in question. One of the first appearances of a \textit{piecewise} definition of \( \hat{u} \) appears in Russell and Shampine (1972) [112]. This was an important development, as it now allowed for easier analysis of collocation and for flexibility in choosing finite element size and location.

The question of collocation point location within each finite element was addressed in 1973 by deBoor and Swartz [41] (see also [40]). They proved that selecting the collocation points to coincide with the roots of Legendre polynomials (i.e., the points of Gaussian quadrature) results in
maximum accuracy (assuming certain smoothness conditions) for one-dimensional problems. The two-dimensional analogue for the case of piecewise bi-cubic $\hat{u}$ was given by Prenter and Russell [106] in 1976, proving (again assuming certain smoothness conditions) that the Gaussian quadrature points (in two dimensions) lead to maximum accuracy.

Much other pre-1983 work was done both in analysis and applications and the interested reader is referred to [28] for an overview of this body of literature.

The numerical solution of convection-diffusion equations with a dominant convective term often exhibits oscillations in the vicinity of a sharp front. Researchers have used various collocation methods in an effort to ameliorate this problem. Allen (1983) [1] studied the effect of evaluating the convective term at collocation points “upstream” from their traditional location at the Gauss points. This produced the effect of eliminating the oscillations while “smearing” the sharp front due to introduction of a dissipative error term arising from the upstream weighting. Other attempts to solve this problem include Mohsen and Pinder [93] (1984), who replace the standard Hermite cubic polynomial basis functions with quintic polynomials, again resolving the oscillations but “smearing” the sharp front. This work, which focused only on one-dimensional problems, was extended in 1990 by Bhuiyan et al. [7] in their study of two-dimensional problems which produced similar results. Soliman (1992) [123] takes the point of view that there is a boundary layer in the vicinity of the sharp front and applies a singular perturbation method to “stretch” the boundary layer. He collocates separately outside and inside the boundary layer, then “matches” the two solutions at an inner position. Funaro (1993) [55] takes the approach of collocating on a “staggered grid”. That is, the location of the collocation points is determined by the ratio of local advection and diffusion coefficients. As a result, the location of the collocation points do not form a predetermined geometric pattern. Another approach was studied by Piccirilli [103] (1994), in which upstream weighting of collocation points was combined with the use of “splines under tension” replacing the Hermite cubic basis functions. This approach eliminated both the oscillations and smearing effect but only if collocation points were placed (almost) as far upstream as possible within each finite element.

Prior to 1983, all the analytical work done on collocation discretizations of PDEs centered on
accuracy estimates. Since that date, work commenced on the analysis of iterative methods to solve the linear systems arising from collocation discretization. The first such work is by Papatheodorou in 1983 [100] in which he applies a “block AOR” (accelerated over-relaxation) method to the orthogonal collocation discretizations of Poisson’s equation. He reports that the numbering of equations and unknowns plays a vital role in determining whether or not the method is convergent. The analytical results in this work were extended in 1995 by Lai et al. [86], in which block versions of the standard iterative methods (Jacobi, Gauss-Seidel, and SOR) are shown to converge for the particular numbering system introduced in [100] for orthogonal collocation applied to Poisson’s equation. Sun (1996) [126] also considers block versions of the standard iterations for the orthogonal collocation discretization of Poisson’s equation. He performs numerous algebraic manipulations, breaking down the two-dimensional problem into a series of one-dimensional problems. His method, unfortunately, can not handle the case of non-constant coefficients and he seems to be ignorant of the work of Lai et al. [86], which is very closely related to his own. Bialecki and Dillery (1993) [10] use Schwarz alternating methods to solve the orthogonal collocation discretization of Poisson’s equation and perform Fourier analysis to obtain explicit formulas governing convergence rates. Fernandes and Fairweather (1993) [50] analyze the convergence rates of Celia’s alternating direction orthogonal collocation methods [28] and also apply these methods of solution to hyperbolic problems. Curiously, Cooper and Prenter (1991) [37] also give a convergence theory of alternating direction orthogonal collocation in their paper. Since neither [50] nor [37] cite the work of the other, we believe that they are not aware of each other’s work. Bialecki and Fernandes (1993) [15] and Fernandes (1997) [49] give convergence and stability analyses for a so-called “Laplace-modified” time discretization of the alternating direction orthogonal collocation methods applied to parabolic problems.

Work has also been done on collocation methods where the basis functions are not Hermite cubics and/or the collocation points do not coincide with the Gauss points. Houstis and Vavalis (1984) [73] solve the two-dimensional Poisson’s equation using Hermite cubic basis functions by collocating the Poisson equation and a variant at the nodes (as opposed to the interior of finite elements). They report that the discretization error is second and fourth order and discuss its formulation on
vector computers. This work was extended by Houstis et al. (1988) [74] by considering a more general differential operator than the Poisson operator considered in [73]. Hadjidimos et al. (1993) [69] use an approach in several dimensions in which the differential operator is approximated along lines in each dimension independently, resulting in a simpler discretization that lacks the cross-derivative terms present in orthogonal collocation. Also, as a result, their method of solution is easily parallelizable.

Some papers studying the direct (non-iterative) solution of orthogonal collocation discretizations include Dyksen et al. (1984) [43], who compare Hermite collocation and Galerkin methods and conclude, in the context of their implementation, that collocation is superior in terms of speed. They solve the linear systems via band Gauss elimination or Cholesky factorization. In a follow-up paper, Dyksen and Rice (1986) [46] discuss the importance of using a scaling strategy in Gauss elimination so that the roundoff error does not overwhelm the solution. Fairweather (1986) [47] discusses variants of Gauss elimination and other direct methods to solve one-dimensional orthogonal collocation problems. Bialecki et al. (1992) [14] give a fast $O(N^2 \log N)$, on an $N \times N$ uniform grid) direct method based on matrix decomposition algorithms using fast Fourier transforms. This method of solution is further applied in Bialecki (1994) [8] and Bialecki and Fairweather (1995) [13]. The fast Fourier transform approach was used also in Bialecki and Remington (1995) [16] where singular collocation discretizations arising from pure Neumann or periodic pure periodic boundary conditions were solved in a least squares sense. Bialecki (1995) [9] also considers solving orthogonal collocation discretizations by cyclic reduction methods.

Work on implementation of collocation methods on irregular (non-rectangular) domains includes Mo and Rice (1989) [96], who use orthogonal collocation. They give experimental evidence that the method has $O(h^4)$ accuracy, but no formal analysis is given. Their work was extended by Lai et al. (1994) [85], in which three-dimensional problems and various strategies of ordering the equations and unknowns are discussed.

Applications of collocation methods to partial integrodifferential equation using Laplace transforms include Yan and Fairweather (1992) [142], in which collocation occurs at the Gaussian points.
with basis function of general degree and Fairweather (1994) [48], who uses orthogonal collocation on equations arising in linear viscoelasticity problems.

A software package for solving orthogonal collocation discretizations of elliptic two-dimensional PDEs was developed by Houstis et al. (1985) [72]. This work gives documentation for the software and examples illustrating its use. Another package is ABDPACK by Majaess et al. (1992) [91], [92], which focuses on one-dimensional collocation problems with monomial basis functions. Alt and Vignes [4] use the CADNA library to show the correlation between analytical error estimates of collocation methods and numerical experiments.

The parallelization of the solution of collocation discretizations of PDEs has been studied. Christara et al. (1988) [33] use quadratic splines in place of the cubic Hermite. They develop and analyze methods of solution for the resulting linear systems and implement their methods on hypercube parallel machines. Matrix decomposition strategies to introduce parallelism are studied in Bialecki and Fairweather (1993) [12], in which they compare and contrast finite difference, finite element, and collocation discretizations. Introducing parallelism via domain decomposition to collocation methods is discussed in Zampieri (1991) [143] in the context of solution of linear elasticity. She also considers finite element preconditioners to solve the linear systems via conjugate gradient methods.

Another domain decomposition algorithm is described by Herrera et al. [70] (1994) and applied to subsurface flow and transport problems in Guarnaccia et al. (1994) [38]. In these papers, orthogonal collocation is used. Lai et al. (1996) [84] also use domain decomposition to introduce parallelism for collocation discretizations of elliptic PDEs.

Other applications of collocation include Lin (1992) [90] who applies orthogonal collocation to non-linear diffusion and adsorption equations. Robinson and Fairweather (1993) [110] apply orthogonal collocation to complex-valued PDEs in the field of acoustics and the same authors (1994) [111] apply orthogonal collocation to the one-dimensional Schrödinger-type equations. Orthogonal collocation is applied to a Schrödinger equation also in Morrison et al. (1996) [95].

Finite element preconditioners for conjugate-gradient-type methods applied to collocation discretizations are discussed in Quarteroni and Zampieri (1991) [107]. Sun et al. (1996) [127] study
finite difference and incomplete \( LU \) preconditioning for conjugate-gradient-type methods for orthogonal collocation. So-called “multilevel” preconditioners applied to orthogonal collocation are discussed in Bialecki and Dryja (1997) [11].

For brief overviews of collocation, the following textbooks are suggested: [17], [76], [88], and [104]. A more thorough treatment is found in [105].

## 7.3 Collocation applied to multiphase problems

Although modelling multiphase flow and contaminant transport in porous media has been well studied (see [64] for extensive bibliography), the applications of collocation discretizations to such problems are relatively few in number. The first is apparently Allen and Pinder (1983) [2], in which the evaluation of the convective terms is done at “upstream” points, resulting in oscillation free solutions which are unattainable otherwise. Convergence analysis is given in Allen and Pinder (1985) [3].

Guarnaccia and Pinder (1992) [60] use orthogonal collocation to model two-phase flow with DNAPL and water, incorporating a terms in the equations that represents interphase mass exchange. They also point out that different time scales involved in the contamination as a slug of NAPL migrates through a subsurface system leaving behind immobile residual NAPL compared to that in which the immobile residual NAPL dissolves in and then is transported by the water phase. These concepts are further discussed in Guarnaccia et al. (1992) [59]. A parallel implementation, using the alternating direction methodology referenced above, is given in Guarnaccia and Pinder (1992) [61] in which the fluid phases are assumed to be immiscible. A different parallel implementation, using domain decomposition and allowing interphase mass transport is given in [62]. Yet another parallel implementation, this one using a “conjugate-gradient domain-decomposition” solver, appears in Guarnaccia and Pinder (1994) [63]. Three phase fluid flow and transport is considered in [64], where the collocation equations are solved via the GMRES algorithm discussed above.
Chapter 8

Summary and Conclusions

The goal of the research presented in this thesis is to rapidly solve the systems of linear algebraic equations \( Ax = b \) arising from the Hermite collocation discretization of partial differential equations (PDEs) in two spatial variables. We elect to use the Bi-CGSTAB method of van der Vorst (1992) to achieve this objective. As with any conjugate gradient method, rapid convergence is obtained by the selection of a suitable “preconditioning matrix.” Part of the motivation of selecting the Bi-CGSTAB method is that it is very easily utilized in a parallel processing mode (assuming the selection of an appropriate preconditioner). Given a sufficiently powerful parallel machine, a good parallel algorithm can reduce execution time significantly.

In Chapter 1, the Hermite collocation discretization of ordinary and partial differential equations is introduced. We then discuss the Red-Black numbering of equations and unknowns in the collocation discretization that provides the parallelism. We then define our preconditioning matrix \( P \) and note that its definition provides a particularly efficient way to perform matrix-vector multiplications of the form \( Ay \) that are found in the Bi-CGSTAB algorithm.

One way to anticipate the efficacy of a preconditioned conjugate gradient method is to examine the eigenvalues of the matrix \( P^{-1}A \). In Chapter 2 we derive analytical formulae for the eigenvalues arising from the discretization of Poisson’s equation (with Dirichlet boundary conditions) and of a model parabolic equation (with Dirichlet boundary conditions and suitable initial conditions). These
eigenvalue formulae are all functions of the parameter $\xi$ which governs the position of the collocation points within each rectangular finite element of the PDE's domain.

Because of the dependence of the eigenvalues on $\xi$, we are motivated to learn how varying $\xi$ can "shift" the location of the eigenvalues and thus affect the rate of convergence of Bi-CGSTAB. This study is carried out in Chapter 3. We find that appropriate selection of $\xi$ can indeed "cluster" the eigenvalues and result in Bi-CGSTAB converging in a minimal number of iterations. The MINOS optimization software is used to determine the optimal value of $\xi$ for a number of model problems, including the PDEs that model single phase flow and contaminant transport in porous media.

Chapter 4 is devoted to a derivation of the eigenvalues (and associated eigenvectors) that control the stability of the solution of the collocation discretization of the parabolic PDEs studied in Chapter 3.

In Chapter 5, we apply our preconditioned Bi-CGSTAB method to the PDEs that model multiphase flow and contaminant transport in porous media. Compared to the GMRES method with ILU preconditioner, Bi-CGSTAB with our preconditioner is at worst competitive and often twice as fast. These comparisons are all made on serial (i.e., uniprocessor) computers.

The parallel implementation of Bi-CGSTAB with our preconditioner on a Cray T3E supercomputer is the focus of Chapter 6. We find for sufficiently large problems that we obtain almost linear speedup. In addition, we study the scalability of the algorithm via its isoefficiency function and demonstrate that the theoretical results and numerical experiments correspond well.

This dissertation contributes to the body of knowledge focused on the solution of two-dimensional PDEs discretized by Hermite collocation in several significant ways. The first is the introduction of the Red-Black numbering system which allows for easy parallelization of the solution of $Ax = b$ via the preconditioned Bi-CGSTAB method, which results in almost-linear speedup for sufficiently large problems when implemented on a parallel supercomputer.

Another major contribution is the definition of our preconditioning matrix $P$. Because $P$ is obtained from $A$ in a very straightforward manner, analytical formulas for the eigenvalues of $P^{-1}A$ are able to be derived for a number of symmetric model problems.
Once we have these eigenvalue formulas, we investigate shifting collocation point location in an effort to accelerate convergence. Shifting collocation points “upstream” has been used in the past to ameliorate undesirable oscillations arising from the Hermite collocation discretization applied to convection-dominated convection-diffusion equations. In our work, the collocation points retain their symmetry with respect to finite element geometry, an approach which is apparently unique. Furthermore, using the value of the parameter $\xi$ (which governs collocation point location) that gives fastest convergence produces convergence in 30% fewer iterations compared to using the Gaussian value $\frac{1}{\sqrt{2}}$.

The last major contribution is the analytic computation of the eigenvalues (and associated eigenvectors) which govern the stability of the time-stepping scheme used to solve the time-dependent two-dimensional model PDEs discussed in Chapter 3.

To summarize, the Bi-CGSTAB method with our preconditioner is an efficient way to solve the systems of linear algebraic equations that arise from the Hermite collocation discretization of PDEs. We have demonstrated this by comparing our method with others. We have further enhanced performance by “shifting” the collocation points in an optimal way and by using parallel supercomputers.

Possibilities for future work include:

- working with other preconditioners that may produce superior results.
- implementation with basis functions other than the Hermite cubic polynomials. In particular, we want to consider “splines under tension”, which evidence suggests may ameliorate the problem of numerical oscillations associated with solving convection-dominated convection-diffusion equations.
- generalizing the methods presented here for three-dimensional PDEs.
Bibliography


[92] F. Majaess, P. Keast, G. Fairweather, and K. R. Bennett, “Algorithm 704: ABDPACK and ABBPACK — FORTRAN Programs for the Solution of Almost Block Diagonal Linear Sys-
tems Arising in Spline Collocation at Gaussian Points with Monomial Basis Functions,” ACM


Calculation for $H^+$,” Journal of Physics B (Atomic, Molecular, and Optical Physics), Vol. 29,
No. 12, pp. 2375-2391, 1996.

of Collocation on General Domains,” Numerical Methods for Partial Differential Equations,

cation to CFD,” Proceedings of the 1994 International Symposium on Parallel Architectures,


[100] T. S. Papatheodorou, “Block AOR Iteration for Nonsymmetric Matrices,” Mathematics of

Semiconductor Device Equations on Multiprocessors,” Integrated Computer-Aided Engineering,

138


