Linear and Nonlinear Deconvolution Models

by

Julia Ann Olkin

Technical Report 86-10, April 1986

---

1A Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, Rice University.
LINEAR AND NONLINEAR DECONVOLUTION PROBLEMS

JULIA ANN OLKIN

A Thesis Submitted
In Partial Fulfillment Of The
Requirements For The Degree

DOCTOR OF PHILOSOPHY

Approved, Thesis Committee:

William W. Symes
Professor of Mathematical Sciences
Chairman

John E. Dennis Jr.,
Professor of Mathematical Sciences

Alan R. Levander
Assistant Professor of Geophysics

Houston, Texas
May, 1980

ABSTRACT

This dissertation considers computational methods for solving linear and
nonlinear least squares problems arising from deconvolution applications. For
the linear problems we propose a new preconditioner to speed up the conjugate
gradient algorithm. This preconditioner is based on Cybenko's QR factorization
of a circulant matrix. Several cases are presented in which our method reduces
the amount of computation.

The preconditioner is applied to the linear subproblems which arise from
the linearization of the nonlinear problems. We investigate several algorithms
which take advantage of the inherent Toeplitz structure. The scale degeneracy
present in our nonlinear problems is also remedied.
ACKNOWLEDGEMENTS

There are many people at Rice who enhanced my graduate work -- from the students who studied with me and bounced ideas back and forth, to the talented professors who taught me. It would be difficult to mention everyone, but I would like to highlight a few people.

I owe a great deal of thanks to my advisor, Bill Symes, for his faith in me, his patience, and his encouragement in my work. I would also like to thank my committee members John Dennis, Jr., and Alan Levander. I am indebted to Michael Pearlmutter, who spent numerous hours helping me with computer problems, and to Bill LeFeuvre, keeper of the laser printer. Maria Rossa Celis and Dan Woods gave me advice, and hope that someday my dissertation would be finished. Without the emotional and financial support from my parents, Ingram and Anita Olkin, this degree would not have been possible. I greatly appreciate all their love, and also the mathematical advice from my father. Most of all, I would like to thank my husband, Juan Meza. His unceasing humor, affection, and love made this degree easier to achieve.

This work was supported by ONR grant N-000-14-85-K0728 and NSF grant DCR81-16779.

TABLE OF CONTENTS

Chapter 1  Introduction ......................................................... 1

Chapter 2  Applications and Mathematical Background ......................... 4
  2.1 Preliminaries .......................................................... 4
  2.2 A Mathematical Model of Linear Deconvolution ........................ 6
  2.3 A Mathematical Model of Nonlinear Deconvolution ..................... 11

Chapter 3  Linear Deconvolution Problems ................................... 21
  3.1 Fast Solution Methods ................................................ 22
  3.2 Inverse Filters ....................................................... 28
  3.3 Iterative Methods ..................................................... 31
  3.4 The Conjugate Gradient Method ..................................... 33
  3.5 Results in the Linear Case .......................................... 48

Chapter 4  Nonlinear Deconvolution Problems ................................ 62
  4.1 The Quasi-Newton Framework ........................................ 63
  4.2 Inexact Newton Methods .............................................. 67
  4.3 Our Particular Application ......................................... 68
  4.4 The Semilinear Least Squares Problem ................................ 71
  4.5 Nonuniqueness ......................................................... 79

Chapter 5  Results in the Nonlinear Case .................................... 84
  5.1 A Description of the Tests .......................................... 85
  5.2 The Full Problem - INQNT ............................................ 86
  5.3 The Alternation Algorithm ......................................... 90
  5.4 The Reduced Problem - VPINTR .................................... 92
  5.5 A Brief Summary ..................................................... 97

Chapter 6  Concluding Remarks ................................................. 99

Bibliography .................................................................. 101
Chapter 3  Linear And Nonlinear Deconvolution Problems
3.1a Artificial Matrices: Tolerance = .0001  ........................................ 48
3.1b Artificial Matrices: Tolerance = 1.e-9 ........................................ 49
3.2 Artificial Toeplitz Matrices with Preconditioner ........................... 50
3.3 Artificial Matrices: Condition Numbers ........................................ 50
3.4 Green's Functions From Michaels .................................................. 51
3.5 Green's Functions: Tolerance = .0001 ......................................... 52
3.6 Green's Functions: Tolerance = 1.e-9 ......................................... 53
3.7 Green's Functions: Condition Numbers ........................................ 54

Chapter 5 Results in the Nonlinear Case
5.1 The INQNR Algorithm: Small Residual ........................................ 88
5.2 INQNR: Percentage Savings On Small Residual Problem ................ 89
5.3 The INQNR Algorithm: Zero Residual ......................................... 89
5.4 INQNR: Percentage Savings On Zero Residual Problem ................ 90
5.5 The Alternation Algorithm: Small Residual .................................. 91
5.6 Alternation: Percentage Savings On Small Residual Problem .......... 91
5.7 The Alternation Algorithm: Zero Residual ................................... 92
5.8 Alternation: Percentage Savings On Zero Residual Problem .......... 92
5.9 The VPINTR Algorithm: Small Residual ...................................... 96
5.10 VPINTR: Percentage Savings On Small Residual Problem ............ 96
5.11 The VPINTR Algorithm: Zero Residual ...................................... 97
5.12 VPINTR: Percentage Savings On Zero Residual Problem ............ 97

Chapter 2  Applications and Mathematical Background
3.1 Input Wavelet ................................................................. 16
3.2 Green's Function: Receiver Near Source ..................................... 17
3.3 Convolution Result #1 ......................................................... 18
3.4 Green's Function: Receiver Far From Source ............................. 19
3.5 Convolution Result #2 ......................................................... 20

Chapter 3  Linear And Nonlinear Deconvolution Problems
3.1 Actual Solution .............................................................. 58
3.2 Solution Using Cooley and Tukey’s Method ............................... 59
3.3 Hybrid Version of Horizontal Displacement ............................... 60
3.4 Hybrid Version of Vertical Displacement .................................. 61
CHAPTER 1

Introduction

A large portion of research in numerous disciplines in the sciences is focused on the collection and analysis of time series. Experimental sciences depend on instruments that measure and record the magnitude of a quantity that varies as a function of a sequence-ordering variable. This independent variable is frequently taken to be time, so that the result of the measurements is a series of observations varying over time. Each isolated measurement is often corrupted by undesirable contributions that should be confined to neighboring measurements. When these contributions combine in a particular linear fashion the distortion is mathematically described by convolution.

This study concentrates on deconvolution, namely, the attempt to remove all the distortion caused by the measurement system or devices. The volume of research devoted to deconvolution is tremendous and it would be impossible to touch on all the contributives and applications. Robinson and Treitel [1980], Illinois and Halcey [1981], Jansson [1984], and Michaels [1984] comprise a very short list of references representing a variety of the applications of deconvolution.

We begin with a few preliminaries and a presentation of several applications in Chapter 2. We formulate mathematically the problem we set out to solve in both the linear and nonlinear cases.

In computing the solutions to linear deconvolution problems our method of choice is the preconditioned conjugate gradient method of Hestenes and Stiefel [1962]. We develop a new preconditioner, presented in Chapter 3, which cuts down considerably on the number of iterations and work involved. We explore when this preconditioner should work, and include numerical results confirming our predictions.

In Chapter 4 we investigate several computational methods for solving nonlinear deconvolution problems. In particular, we consider a bilinear problem stemming from an acoustic emission experiment (Michaels, [1984]), which is a prototype of problems that monitor structures. The bilinear problem has a scale degeneracy. We present two remedies and prove that they remove the degeneracy problem.

In Chapter 5 we use three of the algorithms presented for the nonlinear problems, and run numerical tests. In every iteration of all three algorithms we must solve a linear system similar to the linear system which we solve in the linear deconvolution problems. Hence we can use our preconditioner and investigate how much work it can save us. The purpose is not to compare the algorithms with each other but, rather, to see how to take advantage of the structure as much as possible to speed up each algorithm.
A summary of our contributions are in Chapter 6, as well as a mention of future work to do.

CHAPTER 2

Applications and Mathematical Background

Deconvolution is a technique used in a wide cross section of fields, such as physics, biology, engineering, astronomy, statistics, optics, and seismology. There is a natural way to classify our study of deconvolution. First we consider the applications of linear deconvolution, where the output from a system can be represented as a linear combination of the input to the system. Section 2.2 provides some examples of applications and discusses the mathematical model.

In Section 2.3 we turn to the applications and mathematics on nonlinear deconvolution. Once again, several examples are given.

We preface this study of deconvolution by Section 2.1, which introduces some mathematical notation and definitions used throughout this study.

2.1. Preliminaries

We adhere to the following notational conventions. Capital letters refer to matrices, small letters to vectors. The symbol "0" may be either the number 0, a vector of all zeros, or a matrix of all zeros; its form will be clear from the context.
We attempt to be consistent with traditional uses of the names of certain symbols, but this is not always possible. For example, \( x \) and \( y \) typically represent unknowns; \( v \) and \( w \) are generally generic vectors. For matrices, \( D \) is diagonal and \( I \) is the identity. A residual is generally denoted by \( R \) or \( r \).

All norms in this paper are the \( l_2 \), or Euclidean, norm. For a vector \( \mathbf{v} \in \mathbb{R}^n \),

\[
\| \mathbf{v} \|_2 = \left( \sum_{i=1}^{n} v_i^2 \right)^{1/2};
\]

for a matrix \( A \),

\[
\| A \|_2 = \sqrt{\lambda_{\max}(A^T A)},
\]

where \( \lambda_{\max}(A^T A) \) is the largest eigenvalue of the positive semidefinite matrix \( A^T A \).

It is extremely useful to discuss a method's rate of convergence as a way of comparing its performance. Let \( \{ x_k \} \) converge to \( x^* \in \mathbb{R}^n \). If there exists a constant \( c \) \( \in (0, 1) \) and an integer \( K \geq 0 \) such that for all \( k \geq K \),

\[
\| x_{k+1} - x^* \| \leq c \| x_k - x^* \|,
\]

then \( x_k \) is said to converge \( q \)-linearly to \( x^* \), where \( q \) signifies quotient. If the constant \( c \) depends on \( k \) in such a way that the sequence \( \{ c_k \} \) converges to 0, and

\[
\| x_{k+1} - x^* \| \leq c_k \| x_k - x^* \|,
\]

then \( x_k \) is said to converge \( q \)-superlinearly to \( x^* \). If there exist constants \( p > 1 \) and \( c > 0 \) such that

\[
\| x_{k+1} - x^* \| \leq c \| x_k - x^* \|^p,
\]

then \( x_k \) is said to converge to \( x^* \) with \( q \)-order at least \( p \). If \( p = 2 \) the convergence is said to be \( q \)-quadratic. A weaker form of convergence is \( r \)-order, where \( r \) signifies root. The errors \( \| x_k - x^* \| \) of a sequence converging with \( r \)-order \( p \) are bounded above by a sequence which converges with \( q \)-order \( p \).

### 2.2. A Mathematical Model of Linear Deconvolution

Linear input-output systems can be described by the following diagram, in which \( x(j) \) denotes the input to a system at time \( j \), and \( s(k) \) the output at time \( k \):

\[
\begin{align*}
\mathbf{x}(j) & \rightarrow \begin{bmatrix} b(k-j) \\ \text{measurement} \\ \text{system} \end{bmatrix} \rightarrow \mathbf{s}(k).
\end{align*}
\]

(2.2.1)

If the output \( s(k) \) is independent of \( j \), the time when the input occurred, then the system is time-invariant.

The function \( b(\cdot) \) is the system's transfer function. In some applications the transfer function is unknown (Mendel [1983]). One well-known application of an unknown transfer function occurs in oil exploration. In this application the most widely used geophysical technique is reflection seismology, in which a subsurface image is generated from surface measurements. A vibrational source of seismic energy, such as explosives, is located at the earth's surface near an array of sensors. These explosives generate waves that are reflected and transmitted at interfaces due to the impedance mismatches between different
geological layers. These reflected waves are then transmitted back to the surface, where they are recorded by the sensors. The procedure is repeated at many source and sensor locations, producing an image of the subsurface reflectors. In this example the transfer function is the physical composition, size, location, or behavior of the subsurface layers.

In other applications the transfer function is known and the input \( z \) is unknown. An example where this occurs is in communication systems, where a message is transmitted over a channel which distorts it. Using deconvolution that distortion is removed so that the original input can be obtained.

Mathematically the diagram in (2.2.1) can be represented as an integral equation by

\[
s(k) = \int_{-\infty}^{\infty} b(k-r)z(r) \, dr. \tag{2.2.2}
\]

If we denote the convolution operator as \( * \), the integral equation can be represented by

\[
s = b * x.
\]

A discrete version of (2.2.2) yields

\[
s(k) = (\Delta r) \sum_{j=-\infty}^{\infty} b(k-j\Delta r)z(j\Delta r). \tag{2.2.3}
\]

By scaling the discretization so that \( \Delta r = 1 \), equation (2.2.3) becomes

\[
s(k) = \sum_{j=-\infty}^{\infty} b(k-j)z(j). \tag{2.2.4}
\]

We begin our analysis with the assumption that the input is zero for large negative time, and subsequently introduce it into the system. In this case \( z \) is causal, that is, for some \( t_0 \), \( z(k) = 0 \) for \( k < t_0 \). We further assume that \( b \) is causal and assign \( t_0 \) equal to 0, so that the limits of summation in (2.2.4) become

\[
s(k) = \sum_{j=0}^{\infty} b(k-j)z(j). \tag{2.2.5}
\]

Because we cannot record the output over infinite time, nor can we run the experiment indefinitely, we make the additional assumption that \( s \) and \( z \) have finite length. Then (2.2.5) becomes

\[
s(k) = \sum_{j=0}^{m} b(k-j)z(j). \tag{2.2.6}
\]

Further, if \( s \) has length \( n+1 \), (2.2.6) becomes a finite set of equations:

\[
s(0) = b(0)x(0),
\]

\[
s(1) = b(1)x(0) + b(0)x(1),
\]

\[
\vdots
\]

\[
s(n) = b(n)x(0) + \cdots + b(n-m)x(m).
\]

These equations can be expressed more succinctly in matrix form as

\[
Pz = s,
\]

where \( P = \{ b(i-j) \}_{i,j=1}^{n} \). For notational ease we use subscripts for the series, that is \( b_k \equiv b(k) \), \( x_k \equiv x(k) \), \( s_k \equiv s(k) \).
\[
B x = \begin{bmatrix}
  b_0 & 0 & \cdots & 0 \\
  b_1 & b_0 & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  b_{n-1} & b_{n-2} & \cdots & b_{n-m}
\end{bmatrix}
\begin{bmatrix}
  x_0 \\
  x_1 \\
  \vdots \\
  x_m
\end{bmatrix} = \begin{bmatrix}
  z_0 \\
  s_1 \\
  \vdots \\
  s_k
\end{bmatrix}.
\tag{2.2.7}
\]

Note that the elements along each diagonal of \( B \) are identical, so that \( B \) is a Toeplitz matrix. If
\[
b_n = b_{n-1} = \cdots = b_{n-m} = 0,
\]
then the matrix \( B \) contains a lower triangle of zeros symmetric to the upper triangle of zeros, in which case \( B \) is a circulant matrix. In this case each column is identical to the previous column, except that its elements are shifted down by one, and the last element wraps around to the first position. If \( b \) were not causal then \( B x = s \) would appear as
\[
\begin{bmatrix}
  b_0 & b_1 & \cdots & b_{n-m} \\
  b_1 & b_0 & \cdots & b_{n-1} \\
  \vdots & \vdots & \ddots & \vdots \\
  b_{n-1} & b_{n-2} & \cdots & b_0
\end{bmatrix}
\begin{bmatrix}
  x_0 \\
  x_1 \\
  \vdots \\
  x_m
\end{bmatrix} = \begin{bmatrix}
  s_0 \\
  s_1 \\
  \vdots \\
  s_k
\end{bmatrix}.
\tag{2.2.8}
\]

which is a more general version of the Toeplitz matrix in (2.2.7). If, in addition, \( b \) were periodic with period \( n+1 \), then \( b_k = b_{k+n+1} \), and (2.2.8) becomes
\[
\begin{bmatrix}
  b_0 & b_n & \cdots & b_{n-m+1} \\
  b_1 & b_0 & \cdots & b_{n-1} \\
  \vdots & \vdots & \ddots & \vdots \\
  b_{n-1} & b_{n-2} & \cdots & b_{n-m}
\end{bmatrix}
\begin{bmatrix}
  x_0 \\
  x_1 \\
  \vdots \\
  x_m
\end{bmatrix} = \begin{bmatrix}
  s_0 \\
  s_1 \\
  \vdots \\
  s_k
\end{bmatrix}.
\tag{2.2.8}
\]

In this case we have a circulant matrix.

Again consider the matrix equation \( B x = s \), where \( B \in \mathbb{R}^{(n+1) \times (m+1)} \) is known, \( x \in \mathbb{R}^{(m+1)} \) is unknown, and \( s \in \mathbb{R}^{(n+1)} \) is known. It frequently happens that the vector \( s \) is not a linear combination of the columns of \( B \), that is, there does not exist a vector \( x \) such that equality in (2.2.8) holds. In such a case, the best we can achieve is to find an \( x \) such that the error between \( B x \) and \( s \) is as small as possible. Precisely, we would like to minimize the squared Euclidean norm of the error

\[
\min_x \| B x - s \|_2^2.
\]

When \( B \) is linear this minimization problem is referred to as a linear least squares problem. Geometrically, \( B x \) is the space spanned by the columns of \( B \). We need to find an \( x \) such that the distance between the point \( s \) and the point \( B x \) in this column space, \( C(B) \), is as small as possible. The minimum distance is achieved by projecting \( s \) onto \( B x \). Hence the error vector
\[
e = B x - s.
\]
is orthogonal to \( C(B) \). Any vector \( y \) in \( C(B) \) has the form \( B y \). Since \( B y \) and \( e \) are orthogonal we have
\[(By)^T e = y^T B^T (B^T B^{-1} e) = y^T (B^T B^{-1} - B^T b) e. \] (2.2.9)

Since (2.2.9) holds for any \( y \in \mathbb{R}^{m+1} \) we must set \( B^T B^{-1} - B^T b \) equal to zero.

To summarize, the solution to the least square problem \( Bx = s \) is given by the solution to the normal equations

\[ B^T B x = B^T s. \]

If \( B \) has full column rank then the unique solution is

\[ x = (B^T B)^{-1} B^T s. \]

Chapter 3 presents several ways to solve the normal equations.

### 2.3. A Mathematical Model of Nonlinear Deconvolution

It would be very convenient if natural phenomena were completely linear. Unfortunately that is far from true. Nonlinearities arise everywhere. One such example of an application of nonlinear deconvolution is the problem of calibrating transducers. Even though a group of transducers may all be produced by the same company, they must still be calibrated in order to be used successfully in laboratory fluid tank experiments. We impose an imaginary rectangular grid on the transducer. If \( s_{ij} \) denotes the \( i-th \) signal at transducer \( j \), \( p_{ik} \) is the boundary condition for the wave equation going from the \( k-th \) gridblock on the \( j-th \) transducer, and \( G_{d,ik} \) is the Green's function that transmits the signal from the \( l-th \) element on the \( i-th \) transducer to the \( k-th \) element on the \( j-th \) transducer, then

\[ s_{ij} = \alpha_{ik} G_{d,ik} p_{ik}. \]

which gives rise to a system of quadratic equations.

The particular nonlinear problem we address here is discussed in Michaels [1982, 1984]. She studies acoustic emission problems, in which a source emits elastic waves through a medium (e.g., a crack or deformation emits waves through a material or structure) and the waves are recorded by transducers or other sensors of particle motion. An application of acoustic emission is in the maintenance of gas pipe lines. The scientist maintains a dynamic history of the materials, watching for stress points and flaws, and determines when to take the pipe out of service.

Consider, again, a linear input-output system where the transfer system is an elastic plate, the input is a force vector \( f \) acting at a point \( P \), and the output is the transducer voltage at a receiver \( R \), which we denote by \( u(t) \). The response \( u(t) \) depends linearly on the orientation of the force vector, so that

\[ u(t) = G f, \] (2.3.1)

where \( G \) is the appropriate Green's function. A time invariant system implies that

\[ u(t) = \int G(t-s) f(s) ds, \]

that is, the response is not dependent on when the force is applied. If we assume that the direction of \( f \) is independent of time \( t \) then

\[ f(t) = \sum_{i=1}^{M} x_i \epsilon_i, \] (2.3.2)

where \( \epsilon_i \) are the standard basis vectors in \( \mathbb{R}^M \). Substituting (2.3.2) into (2.3.1)
yields

\[ u(t) = G(\alpha(t) \sum_{i=1}^{M} x_i \epsilon_i ) = \sum_{i=1}^{M} x_i G(\alpha) \epsilon_i. \]  \hfill (2.3.3)

The response at \( R \) to a force at \( P \) in the \( i \)-th direction with a time history \( \alpha \) is given by \( G(\alpha) \epsilon_i \). We can decompose this as

\[ G(\alpha) \epsilon_i = \int_{0}^{1} G(t-s) \alpha(s) ds. \]  \hfill (2.3.4)

We discretize the integral in (2.3.4), and substitute the discretization into (2.3.3), to yield

\[ u(t) = \sum_{m=1}^{M} \sum_{k=1}^{n} x_m G_m(t-k+1) \alpha(k). \]  \hfill (2.3.5)

Making a small notational change from \( t \) to \( n \) yields our nonlinear equation in final form,

\[ u(n) = \sum_{m=1}^{M} \sum_{k=1}^{n} x_m G_m(n-k+1) \alpha(k). \]  \hfill (2.3.6)

where \( \alpha(k) \) is defined only for \( k=1, \ldots, K \). When \( n > K \) (2.3.6) becomes

\[ u(n) = \sum_{m=1}^{M} \sum_{k=1}^{K} x_m G_m(n-k+1) \alpha(k). \]

The set \( u(n), n=1, \ldots, N \) represents the computed values of \( u \), but we actually measure \( \hat{u}(n), n=1, \ldots, N \). Thus, we would like to choose the \( x \)'s and \( \alpha \)'s to minimize the error between \( u(n) \) and \( \hat{u}(n) \). Define the residual vector \( R \) as the vector of errors

\[ R(y) = \begin{bmatrix} u^1(y) - \hat{u}^1(y) \\ \vdots \\ u^N(y) - \hat{u}^N(y) \end{bmatrix}. \]

Here we have made a subtle change of notation in that \( y \) is the vector combining the unknowns \( x \) and \( \alpha \):

\[ y = (x_1, \ldots, x_M, \alpha(1), \ldots, \alpha(K)). \]

Since \( u \) is a function of \( y \) and of \( n \), we use a superscript notation

\[ u^n(y) \equiv u(n) \]

when referring to \( u \). In order to minimize the vector \( R \), we map \( R \in \mathbb{R}^N \) onto the real line by minimizing the squared Euclidean norm of \( R \):

\[ \min_{y \in \mathbb{R}^{M \times K}} \| R(y) \|_2^2. \]  \hfill (2.3.7)

Methods for solving (2.3.7) are discussed in Chapter 4.

In the elastic plate experiment, if the receiver is close to the source then the Green's function has a few widely separated peaks. Suppose we convolve a wavelet, as in Figure 2.1, with such a Green's function, as in Figure 2.2. The output, shown in Figure 2.3, consists of several fairly well defined copies of the input, scaled and shifted accordingly. On examination one could make a decent guess as to the shape of the input series. On the other hand, if the receiver is far from the source, relative to the plate thickness, then the Green's function contains many peaks, is dense, and may not die out in a short time. An example of such a Green's function is given in Figure 2.4, and its convolution with the same input wavelet is given in Figure 2.5. Intuitively the problem of
deducing the input from this output is a much harder problem. The series in Figure 2.3 is so well conditioned that we cannot do much to improve the methods for deconvolution. As for using the series in Figure 2.5, we can significantly speed up the computational methods, and this is explored further in Chapter 5.
CHAPTER 3

Linear Deconvolution Problems

The main purpose of linear deconvolution is to recover the source signal, which has been blurred, spread, or smoothed by the recording instruments. In Chapter 2 we observe that linear deconvolution can be written as a linear least squares (LLS) model

\[ Bx = s, \]  

(LLS)

where the rectangular matrix \( B \) and the vector \( s \) are known, and \( x \) is the source signal. There is generally no solution \( x \) such that equality holds in the linear least squares model. For reasons discussed in Chapter 2, one attempts to minimize the \( L_2 \) norm of the residual \( s - Bx \) instead. From the theory of least squares the \( x \) that minimizes \( \| s - Bx \|_2 \) is the solution to the normal equations

\[ B^T B x = B^T s. \]  

(NE)

If \( B \) is a circulant matrix then it follows that \( B^T B \) is a Toeplitz matrix. Several fast methods for solving the normal equations in this case are presented in Section 3.1.

If we view \( b \) as a signal which performs a specified action on the input signal \( x \), then we can try to apply an inverse filter to \( b \), thereby "undoing" its operation. Section 3.2 describes an inverse filter.

If \( B \) is Toeplitz then \( B^T B \) has no structure, that is, there is no apparent pattern to the elements of \( B^T B \), in which case fast solution methods are no longer applicable. Iterative methods provide possible alternatives, and several iterative methods are discussed in Section 3.3.

Section 3.4 presents one particular iterative method - the Conjugate Gradient method. This method is particularly appealing because the number of iterations to achieve convergence can be significantly decreased by a judicious choice of the preconditioner.

We have developed a preconditioner whose usefulness is confirmed by the test results given in Section 3.5.

3.1. Fast Solution Methods

We refer to a particular group of methods as fast solution methods because these comprise the fastest ways of solving problem LLS when \( B \) is circulant. For comparison purposes, let \( B \) have dimension \( (n \times p) \), so that \( B^T B \) has dimension \( (p \times p) \). In this section we use the letter \( C \) instead of \( B \) to emphasize the fact that it is circulant.

The earliest algorithm for this model is due to Levinson [1947], who iteratively solved larger and larger Toeplitz systems until he acquired the solution to the full problem. We can form the normal equations (NE) and apply Levinson's algorithm to the Toeplitz system \( C^T C \). Robinson and Treitel [1980]
alter this approach somewhat to develop their Toeplitz recursion procedure.

In brief, one obtains the solution to an \((m+1) \times (m+1)\) Toeplitz system from the solution to an \(m \times m\) Toeplitz system, together with the auxiliary constants:

\[ a_{m0}, \ldots, a_{mm}, \text{ for } m = 1, \ldots, p, \text{ and } \alpha, \beta, \gamma. \]

When \(m = p\) the solution to the full system \((NE)\) is obtained. The auxiliary \(a_j\)'s are updated via a recursive \(z\) transform relationship. Let

\[ A_m(z) = a_{m0} + a_{m1}z + \cdots + a_{mm}z^m. \]

Then

\[ A_{m+1}(z) = A_m(z) - \frac{\beta_m}{\alpha_m} z^{m+1} A_m \left( \frac{1}{z} \right), \]

and the new \(a_j\)'s are computed by matching coefficients on both sides of this equation. Similarly, let

\[ X_m(z) = x_{m0} + x_{m1}z + \cdots + x_{mm}z^m. \]

Then, matching coefficients in

\[ X_{m+1}(z) = X_m(z) + \frac{\gamma_m \beta_{m+1}}{\alpha_{m+1}} z^{m+1} A_{m+1} \left( \frac{1}{z} \right) \]

gives an update to the solution vector. Finally, one updates \(\alpha, \beta, \gamma\) and replaces \(m\) by \(m+1\). The time required to solve \((NE)\) is proportional to \(p \log p\) and the storage requirements are proportional to \(p^2\).

Cooley and Tukey [1965] devised the fast Fourier transform (FFT) of a length \(p\) vector. This technique calculates the discrete Fourier transform (DFT)

In time proportional to \(p \log p\). If we are solving

\[ C_{eq} x = s \]

where \(C_{eq} \in \mathbb{R}^{p \times p}\) is square and circulant, then the DFT diagonalizes \(C_{eq}\). That is, we can represent \(C_{eq}\) in diagonal form

\[ C_{eq} = F A F^{-1}, \]

where \(A\) is a diagonal matrix whose elements are the eigenvalues of \(C\). Then (3.1.1) can be rewritten as

\[ A F^{-1} x = F^{-1} s, \]

where \(F^{-1} s\) and \(F^{-1} x\) are actually the DFT's of the vectors \(s\) and \(x\), respectively. Thus, \(F^{-1} x\) can be readily calculated from (3.1.2), and then inverse Fourier transformed to recover \(x\). If \(C_{ref} \in \mathbb{R}^{m \times m}\) is circulant but rectangular, and if we have consistent data, then the matrix equation

\[ C_{ref} x = s \]

can be written equivalently as

\[ C_{eq} y = s, \]

where \(C_{eq}\) is formed from \(C_{ref}\) by continuing the circulant process. The solution \(y\) to (3.1.4) equals the solution \(x\) to (3.1.3), appended with enough zeros to make an \(n\)-vector.

If the data is inconsistent then this method does not work well, but people use it anyway. Andrews and Hunt [1977], among others, use this approach in image restoration.
Cybenko [1983] constructs a factorization of an \( n \times p \) matrix \( A \) into an upper triangular \( p \times p \) matrix \( T \) and an \( n \times p \) matrix \( Q \) with orthogonal columns:

\[
AT = Q. \tag{3.1.5}
\]

His method consists of sweeping across contiguous columns of \( A \), recursively solving least squares problems from previous solutions. The idea is to piece together the residuals and solutions of small least squares problems to easily calculate the residuals and solutions to larger problems. A particular set of these residuals and solutions form the matrices \( Q \) and \( T \), respectively. The following theorem provides the main tool for the algorithm.

THEOREM 3.1 Let \( A \) be an \( n \times k \) real matrix and \( b \) and \( c \) be vectors in \( \mathbb{R}^n \), such that the augmented matrix \( (b,A,c) \) has full rank \( k + 2 \). Let \( x \) and \( w \) be vectors in \( \mathbb{R}^k \) such that the \( n \)-vectors \( r \) and \( s \) given by

\[
r = Ax + b \quad \text{and} \quad s = Aw + c
\]

are the residuals in solutions to the least-squares problems

\[
\text{minimize} \| Ax + b \|_2, \quad \text{minimize} \| Aw + c \|_2,
\]

respectively. We define

\[
\alpha = \frac{r^T s}{r^T r}, \quad \beta = \frac{s^T r}{s^T s},
\]

so that

\[
z = \begin{bmatrix} \alpha \\ w + \alpha x \end{bmatrix}
\]

solves the least-squares problem

\[
\text{minimize} \| (b, A) z + c \|_2,
\]

while

\[
v = \begin{bmatrix} x + \beta w \\ \beta \end{bmatrix}
\]

solves the least-squares problem

\[
\text{minimize} \| (A, c) v + b \|_2.
\]

The residuals for the above two least-squares solutions are

\[
s + \alpha r \quad \text{and} \quad r + \beta s,
\]

respectively.

Using (3.1.5) we obtain the equalities:

\[
\min_s ||Ax - y||_2 = \min_s ||QT^{-1}x - y||_2 = \min_s ||T^{-1}z - Q^Ty||_2,
\]

from which the solution \( x \) is obtained by multiplying \( y \) by \( Q^T \) and then by \( T \). The primary savings in efficiency occurs when \( A \) is a circulant matrix \( C \). Then this factorization takes time proportional to \( np \).

3.2. Inverse Filters

The input \( x(t) \), which we are trying to recover, is actually contaminated by noise. Obviously we would like to recover as pure an input as possible. In our previous discussions of linear deconvolution we minimized the effects of the noise by minimizing the residual between the output and the input. The idea presented here is to pass the output from the system through an inverse filter, so designed as to minimize the effect of the input noise.
Suppose that the input is linearly contaminated by noise, so that we have

\[ g(t) = x(t) + n(t). \]

Suppose also that \( g(t) \) is given. We would like to recover \( x(t) \) and, if possible, some future values \( x(t+\alpha) \) of \( x(t) \). Schematically we represent this process by the diagram

- **input**
- \( g(t) = x(t) + n(t) \)
- **linear filter**
- \( b(k) \)
- **output**
- \( s(t) \)

\[ Y(\omega) = \left( \begin{array}{c}
Y(\omega) = \\
\text{inverse filter}
\end{array} \right) \]

We will design \( Y(\omega) \) to minimize the mean-square error, on the average over time, between the actual output \( a(t) \) and the desired output \( x(t+\alpha) \). Note that if \( \alpha \) is zero then the filter attempts to just remove the noise. Consequently we choose \( Y(\omega) \) to minimize

\[ \frac{1}{T} \int_{-T}^{T} [a(t) - x(t+\alpha)]^2 dt \]

for large values of \( T \). Wiener [1940] developed the solution to this minimization problem, and we discuss his main results. For more details the reader is referred to Wiener's monograph [1940] or to Goldman [1953].

Let \( \varepsilon(t) \) denote the error

\[ \varepsilon(t) = x(t+\alpha) - a(t), \]

and define \( \varepsilon_T(t) \) by

\[ \varepsilon_T(t) = \begin{cases} 
\varepsilon(t), & t \in [-T,T] \\
0, & \text{otherwise} 
\end{cases}. \]

with similar definitions for \( a_T(t), x_T(t) \), and \( n_T(t) \). Denote the Fourier transforms of \( \varepsilon_T(t), a_T(t), x_T(t), x_T(t+\alpha) \) and \( n_T(t) \) by \( E_T(\omega), A_T(\omega), X_T(\omega), \)

\( X_{T,\alpha}(\omega) \), and \( N_T(\omega) \), respectively. \( W_\varepsilon \) is the power spectrum of \( \varepsilon_T(t) \), and is given by

\[ W_\varepsilon(\omega) = \lim_{T \to \infty} \frac{1}{2T} |E_T(\omega)|^2. \]

Similarly, define \( W_a \) and \( W_x \) as the power spectrum of \( n \) and \( x \), respectively.

Then the average value of \( \varepsilon^2(t) \) over time, denoted by \( \overline{\varepsilon^2(t)} \) is

\[ \overline{\varepsilon^2(t)} = \frac{1}{2T} \int_{-T}^{T} \varepsilon^2(t) dt = \frac{1}{2T} \int_{-\infty}^{\infty} |E_T(\omega)|^2 d\omega \]

\[ = \frac{1}{2T} \int_{-\infty}^{\infty} |E_{T,\alpha}(\gamma)|^2 d\gamma = \int_{-\infty}^{\infty} W_\varepsilon(\gamma) d\gamma, \] \hspace{1cm} (3.2.1)

where \( \omega = 2\pi \gamma \). The input \( g \) over the range \([-T,T]\) is

\[ g_T(t) = x_T(t) + n_T(t), \] \hspace{1cm} (3.2.2)

and the error \( \varepsilon \) is

\[ \varepsilon_T(t) = x_T(t+\alpha) - a_T(t). \] \hspace{1cm} (3.2.3)

From our diagram we have that

\[ a_T(t) = y_T(t) * g_T(t). \] \hspace{1cm} (3.2.4)

Using equation (3.2.2) in (3.2.4) yields

\[ a_T(t) = y_T(t) * (x_T(t+\alpha) + n_T(t)). \]

It is well known that the Fourier transform of a convolution is the product of the individual Fourier transforms, so that
\[ A_T(\omega) = Y(\omega)X_T(\omega) + Y(\omega)N_T(\omega) \]  
(3.2.5)

and
\[ X_{T,a}(\omega) = X_T(\omega)e^{i\omega a}. \]  
(3.2.6)

We Fourier transform both sides of equation (3.2.3) and substitute in (3.2.6) and (3.2.6) to yield
\[ E_T(\omega) = X_{T,a}(\omega) - A_T(\omega) \]
\[ = X_T(\omega)e^{i\omega a} - Y(\omega)\]  
(3.2.7)

Suppose that \( x(t) \) and \( n(t) \) are incoherent, that is, knowing the values in one of the series does not provide information about the values in the other series. The Fourier transforms \( X_T(\omega) \) and \( N_T(\omega) \) are also incoherent, which means that
\[ |X_T(\omega)N_T(\omega)|^2 = \int_{-T}^{T} |(X_T(\omega)N_T(\omega))^2|d\omega = 0. \]  
(3.2.8)

We now take the limit of \( E_T(\omega) \) in equation (3.2.7), and then use equation (3.2.8) to eliminate the cross term. Then
\[ W_s(\omega) = \lim_{t \to \infty} \frac{1}{2T} |E_T(\omega)|^2 \]
\[ = |Y(\omega) - e^{i\omega a}|^2 W_s(\omega) + |Y(\omega)|^2 W_s(\omega). \]  
(3.2.9)

Substituting equation (3.2.9) into equation (3.2.1) yields
\[ \bar{e}(t)^2 = \int_{-\infty}^{\infty} W_s(\omega)d\omega \]
\[ = \int_{-\infty}^{\infty} \left( |Y(\omega) - e^{i\omega a}|^2 W_s(\omega) + |Y(\omega)|^2 W_s(\omega) \right)d\omega. \]  
(3.2.10)

The optimum filter that minimizes the average of \( e(t)^2 \) is
\[ Y(\omega) = \frac{W_s(\omega)}{W_s(\omega) + W_n(\omega)} e^{i\omega a}. \]  
(3.2.11)

although this may not be a physically realizable filter. To derive (3.2.11) write
\[ Y(\omega) = C(\omega)e^{i\beta(\omega)} \]  
(3.2.12)

where \( C(\omega) \) and \( \beta(\omega) \) are real and \( C(\omega) \) is positive for all \( \omega \). Substituting (3.2.12) into (3.2.10), we obtain
\[ \bar{e}(t)^2 = \int_{-\infty}^{\infty} \left( [C(\omega)^2 + 1 - 2C(\omega)cos(\omega a - \beta(\omega))]W_s(\omega) + C(\omega)^2 W_n(\omega) \right)d\omega. \]

This integral is minimized with respect to \( \beta(\omega) \) when \( cos(\omega a - \beta(\omega)) \) is maximized, which occurs when
\[ \beta(\omega) = \omega a. \]  
(3.2.13)

We rewrite \( \bar{e}(t)^2 \) as
\[ \bar{e}(t)^2 = \int_{-\infty}^{\infty} \left( \sqrt{W_s(\omega) + W_n(\omega)}C(\omega) - \frac{W_s(\omega)}{\sqrt{W_s(\omega) + W_n(\omega)}} \right)^2 + \frac{W_s(\omega)W_n(\omega)}{W_s(\omega) + W_n(\omega)} \right)d\omega. \]

This integral is minimized with respect to \( C(\omega) \) when the term in square brackets is minimized. That is, when
\[ \sqrt{W_s(\omega) + W_n(\omega)}C(\omega) - \frac{W_s(\omega)}{\sqrt{W_s(\omega) + W_n(\omega)}} = 0, \]
so that
\[ C(\omega) = \frac{W_s(\omega)}{W_s(\omega) + W_n(\omega)} \]  
(3.2.14)

Now substitute (3.2.13) and (3.2.14) into (3.2.12) to give the optimal filter (3.2.11).
It might be preferable to work with $Y$ in the time domain instead of in the frequency domain. We can perform a Fourier transform of the vector $Y(\omega)$ to obtain

$$
\hat{Y}(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} Y(\omega) e^{i\omega t} d\omega = \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{W_x(\omega)}{W_x(\omega) + W_y(\omega)} e^{i\omega(t+\alpha)} d\omega.
$$

To form $Y(\omega)$ we need to compute the Fourier transforms of $x_T(t)$ and $n_T(t)$. The calculation of $X_T(\omega)$ can be accomplished using the FFT discussed in the previous section. Characteristics of the noise signal, such as wavelength and amplitude information, or probability distribution, are assumed, and a corresponding $N_T(\omega)$ chosen. Next we compute $W_x(\omega)$ and $W_y(\omega)$ for a finite number of $\omega$ values, after which we are ready to convolve the output $s(t)$ with a discretized version of the filter $\hat{Y}(t)$ to obtain a good approximation to $x(t+\alpha)$.

### 3.3. Iterative Methods

In Section 3.1 we considered the case where $B$ is a circulant matrix in solving the normal equations. If $B$ is not circulant then $B^T B$ is not a Toeplitz matrix and the fast solution methods do not apply. As an example we run the FFT algorithm of Cooley and Tukey to solve a square Toeplitz matrix system. Figure 3.1 displays the actual solution, and Figure 3.2 displays the computed solution. The graphs do not match up well at all; the peaks in the computed solution are shifted away from their correct positions, and their amplitudes are not too close to the actual values.

We could use direct methods to solve the normal equations, but they cost a lot and do not make use of any inherent structure in the matrices. So we explore the various iterative methods.

There are classical methods of Jacobi, Gauss-Seidel, and Successive Overrelaxation that can be applied to the normal equations. These require the user to form $A = B^T B$ and $g = B^T s$. Decompose $A$ as

$$
A = D - L - U,
$$

where $D$ is diagonal, $L$ is lower triangular, and $U$ is upper triangular. These classical methods to solve for $x$ in $Ax = g$ can be written in matrix form, respectively, as

$$
x^{k+1} = \begin{cases} 
D^{-1}(L + U)x^k + D^{-1}g, \\
(D - L)^{-1}Ux^k + (D - L)^{-1}g, \\
(I - \omega D^{-1}L)^{-1}(I - \omega I + \omega D^{-1}U)x^k + \omega(I - \omega D^{-1}L)^{-1}D^{-1}g,
\end{cases}
$$

where $x^k$ is the $k$-th approximation to $x$ (Varga [1982]). The drawback to the first two methods is that $A = B^T B$ is generally not diagonally dominant, so that convergence cannot be guaranteed. We choose not to explore SOR further because it, also, does not take advantage of the structure in the matrix $B$, and its convergence rate depends on the parameter $\omega$.

Although $A$ may not be diagonally dominant, it is always symmetric and positive semidefinite. If $B$ is nonsingular then $A$ is positive definite. One iterative method that ensures convergence when the matrix is symmetric and positive definite is the conjugate gradient method. We discuss this method in
more detail in the next section, where it will become clear why this is our method of choice.

Kailath, Kung, and Morf [1970] have developed a criterion, called the displacement rank, which determines how close a matrix is to being Toeplitz. They use algorithms similar to that of Levinson based on displacement rank. It would certainly be interesting to compare these algorithms to the iterative methods examined in this study, and future work may contain these comparisons.

3.4. The Conjugate Gradient Method

The Conjugate Gradient (CG) method of Hestenes and Stiefel [1952] has several attractive features that help explain its wide use in solving the matrix equation $Ax = b$, where $A$ is an $n \times n$ matrix.

1. There are no parameters to choose.

2. It is necessary to store only $Av$ for a vector $v$, rather than the entire matrix $A$. Not every application of advantage of this feature but, fortunately, a Toeplitz matrix application can.

3. For a symmetric and positive definite matrix convergence is theoretically guaranteed in $n$ iterations.

A brief introduction to the derivation of the method (see Golub and Van Loan [1983]) follows. Assume that we want to solve for $x$ in the equation

$$Ax = b, \quad A \in \mathbb{R}^{n \times n}, \quad b \in \mathbb{R}^{n}. \quad (3.4.1)$$

If $A$ is symmetric and positive definite then this problem is equivalent to solving for the minimizer of

$$\phi(x) = \frac{1}{2} x^T Ax - x^T b. \quad (3.4.2)$$

The solutions of (3.4.1) and (3.4.2) are identical, namely $x = A^{-1}b$.

The simplest strategy for minimizing $\phi(x)$ is the method of steepest descent, where the step at each iteration is along the negative gradient direction

$$-\nabla \phi(x_j) = b - Ax_j \iff r_j.$$

We call $r_j$ the residual of $x_j$. To minimize $\phi(x)$ along $-\nabla \phi(x)$ for each $x_j$, we need to find an $\alpha^* \in \mathbb{R}$ such that

$$\phi(x + \alpha^* r) < \phi(x + \alpha r) \text{ for all } \alpha \in \mathbb{R}.$$

The minimizes to this problem at a given $x$ and $r$ is

$$\alpha = \frac{r^T r}{r^T Ar}. \quad (3.4.3)$$

The drawback to the method of steepest descent is that it is prohibitively slow. Hence we consider minimizing $\phi$ along a set of directions distinct from the negative gradient direction. Once again, the $\alpha$ that minimizes $\phi(x + \alpha p)$ is given by

$$\alpha = \frac{p^T r}{p^T Ap}. \quad (3.4.4)$$

The question now is how to characterize this set of direction vectors $\{p_i\}$. Notice that
\[ \phi(x + \alpha p) = \phi(x) - \frac{1}{2} \frac{(p^T r)^2}{p^T Ap}. \]

Because \( A \) is positive definite, \( p^T Ap > 0 \), and hence
\[ \frac{1}{2} \frac{(p^T r)^2}{p^T Ap} > 0. \]

To minimize \( \phi(x + \alpha p) \) we want to subtract off a positive number from \( \phi(x) \). Consequently it is natural to impose the condition that \( p^T r \neq 0 \), so that our set of direction vectors will not be perpendicular to the residuals.

The framework we have so far for the algorithm is

**Algorithm 3.0 Preliminary Form.**

Take as initial point \( x_0 = 0 \).

For \( j = 0, 1, \ldots \) until convergence determine

\[ r_j = b - Ax_j, \]
choose \( p_j \) such that \( p_j^T r_j \neq 0 \),
\[ \alpha = \frac{p_j^T r_j}{p_j^T Ap_j}, \]
\[ x_{j+1} = x_j + \alpha p_j. \]

Test for convergence.

End.

If, somehow, we could choose the \( p_j \) to be linearly independent with the added property that each \( x_j \) solves the subproblem

\[ \min_{x \in \mathbb{R}^{m \times 1} \mid \langle x, p_j \rangle = 0} \phi(x), \]

then we could guarantee convergence and finite termination at the \( n^{th} \) step.

Not only can this be done, but there exists a recursive relationship among the \( p_j \). Let \( \langle v, w \rangle \) denote the \( l_2 \) inner product of \( v \) and \( w \). Let
\[ \beta_j = -\frac{\langle p_{j-1}, Ap_{j-1} \rangle}{\langle p_{j-1}, Ap_{j-1} \rangle}. \]

Then
\[ p_j = r_{j-1} + \beta_j p_{j-1}. \]

Because there is a recursive relationship among the residuals, the form of \( \beta_j \) can be written in terms of already computed quantities, as can be seen in the final version of the Conjugate Gradient Method.
Algorithm 3.1 The Conjugate Gradient Method

Initialize $x_0 = 0$, $r_0 = b$, $p_0 = r_0$.

For $j = 0, 1, \ldots$ until convergence determine

$$\alpha = \frac{<r_j, r_j>}{<p_j, Ap_j>},$$

$$x_{j+1} = x_j + \alpha p_j,$$

$$r_{j+1} = r_j - \alpha Ap_j,$$

$$\beta = \frac{<r_{j+1}, Ap_j>}{<p_j, Ap_j>},$$

$$p_{j+1} = r_{j+1} - \beta p_j.$$

Test for convergence.

End.

Notice that each iteration of Algorithm 3.1 requires a matrix-vector multiplication; overall three extra storage vectors are required.

Suppose we apply the CG method to $Ax = b$ where

$$A = B^T B$$

and $b = A^T b$.

After appropriate simplifications we have an algorithm which we call CGNE which is the conjugate gradient method applied to the normal equations.

Algorithm 3.2 The Conjugate Gradient Method For The Normal Equations

Initialize $x_0 = 0$, $r_0 = b$, $p_0 = B^T r_0$.

For $j = 0, 1, \ldots$ until convergence determine

$$\alpha = \frac{<r_j, Bp_j>}{<Bp_j, Bp_j>},$$

$$x_{j+1} = x_j + \alpha p_j,$$

$$r_{j+1} = r_j - \alpha Bp_j,$$

$$\beta = \frac{<BB^T r_{j+1}, Bp_j>}{<Bp_j, Bp_j>},$$

$$p_{j+1} = B^T r_{j+1} - \beta p_j.$$

Test for convergence.

End.

3.4.1. Preconditioning

To analyze the error bounds for the iterates in the CG algorithm, we require two additional definitions. The $A$-norm on $\mathbb{R}^n$ is defined by

$$||w||_A = w^T A w.$$

The condition number of $A$, denoted $\kappa(A)$, is defined by

$$\kappa(A) = ||A|| \cdot ||A^{-1}||.$$

If $Ax = b$ then

$$||x - x_j||_A = (x - x_j)^T A (x - x_j) = 2\phi(x_j) + b^T A b.$$

Since $A^{-1}$ is positive definite provided $A$ is positive definite, minimizing
\[ \| x - x_j \|_A \leq \| x - x_0 \|_A \left( \frac{1 - \sqrt{\kappa(A)}}{1 + \sqrt{\kappa(A)}} \right)^{2j} \]

We can try to lower this bound by controlling \( \kappa(A) \). Smaller values of \( \kappa(A) \) yield smaller values of the constant multiplying \( \| x - x_0 \|_A \). A decrease in the condition number is achieved by preconditioning. To see how this is done, observe that, for a nonsingular matrix \( F \), the solution of the matrix equation

\[ (F^{-1})^T B^T B F^{-1} F x = (F^{-1})^T B^T b \]

is equivalent to the solution of

\[ B^T B x = B^T b. \]

Define the transformations

\[ \tilde{B} = BF^{-1}, \tilde{x} = F x, \tilde{b} = b. \]

Then we can rewrite (3.4.3) as

\[ \tilde{B}^T \tilde{B} \tilde{x} = \tilde{B}^T \tilde{b}. \]

Applying the CGNE algorithm 3.2 to (3.4.4) yields the solution \( x \) in considerably fewer iterations if

\[ \kappa(\tilde{B}^T \tilde{B}) < \kappa(B^T B). \]

The key idea then is to choose an \( F \) in the transformation such that

1) \( \kappa(\tilde{B}^T \tilde{B}) \) is as close to 1 as possible; and

2) \( F^T F z = v \) is an easy system to solve.

The reason for the second statement will be evident from Algorithm 3.4.

Let us apply the CGNE algorithm to (3.4.4), where for consistency we put a tilde on all the vectors.

**ALGORITHM 3.3 CGNE applied to (3.4.4)**

Initialize \( \tilde{x}_0 = 0, \tilde{r}_0 = \tilde{b}, \tilde{p}_0 = \tilde{b}^T \tilde{r}_0. \)

For \( j = 0, 1, \ldots \), until convergence determine

\[ \tilde{\alpha} = \frac{\langle \tilde{r}_j, \tilde{B} \tilde{p}_j \rangle}{\langle \tilde{B} \tilde{p}_j, \tilde{B} \tilde{p}_j \rangle}, \]

\[ \tilde{x}_{j+1} = \tilde{x}_j + \tilde{\alpha} \tilde{p}_j, \]

\[ \tilde{r}_{j+1} = \tilde{r}_j - \tilde{\alpha} \tilde{B} \tilde{p}_j, \]

\[ \tilde{\beta} = \frac{\langle \tilde{B} \tilde{r}_{j+1}, \tilde{B} \tilde{p}_j \rangle}{\langle \tilde{B} \tilde{p}_j, \tilde{B} \tilde{p}_j \rangle}, \]

\[ \tilde{p}_{j+1} = \tilde{B} \tilde{r}_{j+1} - \tilde{\beta} \tilde{p}_j. \]

Test for convergence.

End.

Suppose that we make the following change of variables.

\[ M = F^T F, p = F^{-1} p, r = \tilde{r}, z = M^{-1} B^T r, \]

and substitute these values into Algorithm 3.3, rewriting the formulas in terms of the old variables from (3.4.5). For brevity we exclude the subscripts when possible. Then

\[ \tilde{\alpha} = \frac{\langle \tilde{r}, \tilde{B} \tilde{p} \rangle}{\langle \tilde{B} \tilde{p}, \tilde{B} \tilde{p} \rangle} = \frac{\langle r, B^{-1} F p \rangle}{\langle B^{-1} F p, B^{-1} F p \rangle} = \frac{\langle r, Bp \rangle}{\langle Bp, Bp \rangle} = \alpha. \]
\[ x_{j+1} = x_j + \beta p_j \rightarrow Fx_{j+1} = Fx_j + \alpha p_j \rightarrow x_{j+1} = x_j + \alpha p_j. \]

\[ r_{j+1} = r_j - \alpha Bp_j \rightarrow r_{j+1} = r_j - \alpha B^{-1}Fp_j \rightarrow r_{j+1} = r_j - \alpha Bp_j. \]

\[ \beta = -\frac{\langle Bp, r \rangle}{\langle Bp, Bp \rangle} = -\frac{\langle Bp, r \rangle}{\langle Bp, r \rangle} = -\frac{\langle Bp, Bp \rangle}{\langle Bp, Bp \rangle} \]

Now we are ready to rewrite Algorithm 3.3 in terms of the original variables.

We call Algorithm 3.4 the PCGNE algorithm because it is the preconditioned conjugate gradient method applied to the normal equations.

### Algorithm 3.4 PCGNE

Initialize \( x_0 = 0 \) and \( r_0 = b \).

Solve \( Mz_1 = B^T r_0 \) for \( z_1 \). Set \( p_0 = z_0 \).

For \( j = 0, 1, \ldots \), until convergence determine

\[ \alpha = \frac{\langle r_j, Bp_j \rangle}{\langle Bp_j, Bp_j \rangle}, \]

\[ x_{j+1} = x_j + \alpha p_j, \]

\[ r_{j+1} = r_j - \alpha Bp_j, \]

solve \( Mz_{j+1} = B^T r_{j+1} \) for \( z_{j+1} \),

\[ \beta = \frac{\langle Bz_{j+1}, Bp_j \rangle}{\langle Bp_j, Bp_j \rangle}, \]

\[ p_{j+1} = z_{j+1} - \beta p_j, \]

Test for convergence.

End.

At each iteration we need to solve the system of equations

\[ Mz = B^T r, \]

where \( M = F^T F \).

Now it should be evident why the structure of \( F \) is so important. If solving (3.4.6) were too cumbersome then the advantage of preconditioning would be lost. Hence, we would like \( F^T F \) to be banded or triangular.

In the next section we present a candidate preconditioner which satisfies the two requirements we require of any preconditioner.

### 3.4.2. A Specific Preconditioner

Recall that Cybenko's method (Section 3.1) produces the QR factors of an \( n \times p \) circulant matrix in \( O(np) \) flops:

\[ CT = \tilde{Q}. \]

(3.4.7)

With very little effort we can extend this factorization to find the diagonal matrix \( D = \text{diag}(d_1, d_2, \ldots, d_n) \) such that

\[ \tilde{Q}D^{1/2} = \tilde{Q}, \]

where

\[ d_i = (Q^T Q)^{-1} \]

and \( Q \) has orthonormal columns. Notice that

\[ \tilde{Q}^T \tilde{Q} = (QD^{1/2}Q^T QD^{1/2}) = D^{1/2}Q^T QD^{1/2} = D. \]

(3.4.8)

We consider the case where \( B \) is a Toeplitz matrix and we want to solve

\[ B^T B \xi = B^T s. \]

(3.4.9)
Our idea is to transform $B$ into a circulant matrix $C$. To see the advantage of this idea, consider using the matrix

$$TD^{1/2}$$

as a preconditioner on the system

$$CTC = CTs.$$  

Using (3.4.7) and (3.4.8) we have

$$CTC = (QT^{-1}TQT^{-1}) = (T^{-1})^TQT^{-1} = (T^{-1})^TD(T^{-1}).$$  

(3.4.10)

Applying the preconditioner to (3.4.10) yields

$$(TD^{-1/2})^TCTC(TD^{-1/2}) = (TD^{-1/2})^T(D(T^{-1})(TD^{-1/2})$$

$$= D^{-1/2}T(T^{-1})^TD(T^{-1})TD^{-1/2}$$

$$= D^{-1/2}DD^{-1/2}$$

$$= I.$$ 

Let

$$F = (TD^{-1/2})^{-1}$$

be the preconditioner. Then the preconditioned system

$$(F^{-1})^TCTCFx = (F^{-1})^TC^Ts$$

will converge in one iteration because

$$(F^{-1})^TCTCF^{-1} = I.$$ 

Our idea is to use this same preconditioner on (3.4.9). We temporarily form $C$ and find its Cybenko factorization, use the factors $T$ and $D^{-1/2}$ to form the preconditioner $F$, and then solve the system

$$(F^{-1})^TB^TBFx = (F^{-1})^TB^Ts.$$ 

Recall that Algorithm 3.4 requires solving for the vector $v$ in

$$Mv = y,$$  

where $M = F^TF$. Expanding $M$ gives

$$M = F^TF = (TD^{-1/2})^T(TD^{-1/2})$$

$$= T^{-1}TD^{1/2}D^{1/2}T^{-1}$$

$$= T^{-1}DT^{-1}.$$ 

Solving (3.4.11) is equivalent to solving

$$(T^{-1})^TD^{-1}v = y$$

for $v$. But then

$$v = T^TD^{-1}Ty.$$  

(3.4.12)

All the matrices on the right hand side of (3.4.12) are known, so we can carry out the multiplication rather than solve a linear matrix system.

The primary question, of course, is which circulant matrix $C$ to choose. Recall that a good preconditioner estimates the inverse of $B$. That is, we want

$$\kappa(B^TB) = \kappa((F^{-1})^TB^TF^{-1}) \approx 1,$$

so that $F$ should be a close approximation to $B$. The matrix $F$ is composed of the factors of the circulant approximation $C$ to $B$. If $C$ is to be close to $B$, then we need to examine

$$\|C^{-1}B - I\| = \|C^{-1}(B - C)\|.$$ 

When $\|\cdot\|$ is a matrix norm, an upper bound is $\|C^{-1}\| \|B - C\|$. Finding a lower bound is a rather difficult problem.

A natural candidate for $C$ is the triangularization of $B$, that is, replacing the lower portion of $B$ by zeros. Schematically, the process is
Toeplitz matrix \( B = \begin{pmatrix} a & 0 & 0 \\ b & a & 0 \\ c & b & a \\ d & c & b \end{pmatrix} \) → \( \begin{pmatrix} a & 0 & 0 \\ b & a & 0 \\ c & b & a \\ d & c & b \end{pmatrix} = \text{circulant matrix } C \)

It is interesting to note that this \( C \) is actually not the closest circulant matrix to \( B \) in the Frobenius norm. The \( C \) satisfying

\[
\min_{C} \| C - B \|_F = \min_{C} \left( \sum_{i,j} (C_{ij} - B_{ij})^2 \right)
\]

is given by a matrix \( \hat{C} \) such that the lower and upper triangles are weighted multiples of the original elements. The Frobenius norm is not a matrix norm and hence \( \hat{C} \) may not be a good approximation to \( B \) in a matrix norm. More important, the test results presented in the next section confirm that our straightforward version of \( C \) yields a very satisfactory preconditioner.

Notice that our preconditioner is dependent on both the number of columns and the size of the elements in the matrix. If the magnitude of the elements in the lower triangle of the Toeplitz matrix are too large in proportion to the remaining elements then when we transform the matrix into a circulant we discard too much information. Hence \( C \) is not close to \( B \) and so the help from the preconditioner may be negligible. Physically, the preconditioner works well when most of the information in the Green's function is near the front of the time series. That is, as time progresses the peaks in the Green's function are fewer and much smaller, hopefully negligible. Then the first column in \( B \) has larger entries near the beginning, and smaller entries near the end. Figure 3.1 gives an example of such a Green's function.

Another case in which the preconditioner is effective is when the "energy" is equally distributed throughout the Green's function, so that the peaks resemble each other, but repeated many times. Here, again, we are not losing much information by chopping off the end of the series. Figure 3.2 gives an example of this kind of Green's function.

The preconditioner is also more effective when the matrix contains fewer columns because then the size of the lower triangle to be discarded is smaller. Physically, this is equivalent to saying that the preconditioner is more effective when the time window is more localized. If we consider the extreme case of when the number of columns equals the number of rows, then the circulant approximation is just a diagonal matrix, which usually does not lead to an effective preconditioner.

Based on the previous discussion, it is obvious that we can generate examples in which the preconditioner does not work well at all. Instead, we show that the preconditioner is effective when we predicted it would be, and that these conditions occur frequently in real life experiments.

### 3.5. Results In The Linear Case

To test the performance of our preconditioner we create normal equations comprised separately of circulant and Toeplitz matrices. Specifically, let \( C \) be a circulant matrix with 100 rows whose first column contains the numbers 1 through 25 by increments of one quarter. The number of columns varies over
the three values 25, 50, and 75. Thus the square normal matrices $C^T C$ have row and column dimensions equal to 25, 50, and 75. Let $\epsilon$ take on the ten values $10^{-4}$ for $a=0, 1, \ldots, 9$. For each value of $\epsilon$ we create a new $C_\epsilon$ where all the elements on the upper and lower triangles of $C$ are multiplied by $\epsilon$. The matrix $C_\epsilon \in \mathbb{R}^{100 \times 100}$ has the form

$$
C_\epsilon = \begin{bmatrix}
\epsilon c_1 & \epsilon c_2 & \cdots & \epsilon c_{100-1} \\
\epsilon c_2 & \epsilon c_1 & \cdots & \epsilon c_{100-2} \\
\vdots & \vdots & \ddots & \vdots \\
\epsilon c_{100-1} & \epsilon c_{100-2} & \cdots & \epsilon c_1
\end{bmatrix}
$$

Notice that $C_\epsilon$ is still a circulant matrix. To create comparable Toeplitz matrices set $B_\epsilon$ equal to $C_\epsilon$ except that zeros replace the upper right triangle:

$$
B_\epsilon = \begin{bmatrix}
\epsilon c_1 & 0 & \cdots & 0 \\
\epsilon c_2 & \epsilon c_1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\epsilon c_{100-1} & \epsilon c_{100-2} & \cdots & \epsilon c_1
\end{bmatrix}
$$

Now we run the preconditioned conjugate gradient method on the two systems

$$
C_\epsilon^T C_\epsilon x = C_\epsilon^T s, \quad (3.5.1)
$$

and

$$
B_\epsilon^T B_\epsilon x = B_\epsilon^T s \quad (3.5.2)
$$

for the 10 values of $\epsilon$, at two tolerance levels, $10^{-4}$ and $10^{-9}$. The vector $s$ has the form $(2, 1, 2, \ldots, 2)$.

Table 3.1a provides the results for this experiment, run with a tolerance of $10^{-4}$, and Table 3.1b shows the results for the tighter tolerance. The entries are the number of iterations until convergence for $\epsilon$ equal to its largest and smallest values. Also included are the number of iterations averaged over all ten values of $\epsilon$. Notice that because this number is an average it will not be an integer, but it will give some idea of the range of results for the varying values of $\epsilon$. We impose an upper limit of 150 iterations.

<table>
<thead>
<tr>
<th>Number of Iterations Until Convergence</th>
<th>Precon.</th>
<th>No Precon.</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Columns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>17</td>
<td>17.1</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>26</td>
<td>25.5</td>
</tr>
<tr>
<td>75</td>
<td>1</td>
<td>31</td>
<td>31.3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>151</strong></td>
<td><strong>110</strong></td>
<td><strong>103.3</strong></td>
</tr>
</tbody>
</table>

Table 3.1a Artificial Matrices: Tolerance $= 10^{-4}$
Table 3.1b Artificial Matrices: Tolerance = 10^{-9}

<table>
<thead>
<tr>
<th>Number of Iterations Until Convergence</th>
<th>Circulant</th>
<th>Toeplitz</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Rows = 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of Columns</td>
<td>25 50 75</td>
<td>25 50 75</td>
</tr>
<tr>
<td>$\epsilon = 1$</td>
<td>1 1 1</td>
<td>40 85 100</td>
</tr>
<tr>
<td>Precon.</td>
<td>1 1 1</td>
<td>1 2</td>
</tr>
<tr>
<td>No Precon.</td>
<td>34 58 75</td>
<td>120 150 150</td>
</tr>
<tr>
<td>$\epsilon = 10^{-9}$</td>
<td>1 1 1</td>
<td>6.9 11.1 15</td>
</tr>
<tr>
<td>Precon.</td>
<td>1 1 1</td>
<td>Average</td>
</tr>
<tr>
<td>No Precon.</td>
<td>34 57 70</td>
<td>35 57 69</td>
</tr>
</tbody>
</table>

Theoretically without a preconditioner the iterates should converge in $n$ iterations for an $n \times n$ system. Equations (3.5.1) and (3.5.2) have dimensions 25x25, 50x50, and 75x75 for the three column lengths. The average number of iterations are certainly less than the theoretical number of iterations of 25, 50, and 75 for the cases with no preconditioner, but when $\epsilon$ is 1 the number of iterations are much higher. The 10 systems are definitely ill-conditioned. Fortunately our preconditioner greatly helps the convergence rate.

In Table 3.1b the number of iterations are larger due to the tighter tolerance restriction. In solving (3.5.2) with our preconditioner the cases eventually converge in two or three iterations with small enough $\epsilon$. Only in the case of the row dimension equal to 25 did the number of iterations reduce to one, and that was for the smallest value of $\epsilon$. Nevertheless, the average convergence of at most 15 iterations is a significant improvement over convergence in upwards of 34 iterations.

Table 3.2 gives some supplemental information to Table 3.1a. Namely, how small must $\epsilon$ be in order to converge in one iteration. For each row dimension, Table 3.2 gives the largest value of $\epsilon$ such that the preconditioned conjugate gradient algorithm converges in one iteration.

Table 3.2 Artificial Toeplitz Matrices With Preconditioner

<table>
<thead>
<tr>
<th>Achieve Convergence in 1 Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Columns</td>
</tr>
<tr>
<td>Size of $\epsilon$</td>
</tr>
</tbody>
</table>

One more feature of interest is a comparison of the condition numbers of $B^T B$, and the preconditioned system $(F^{-1})^T B^T B F^{-1}$. Table 3.3 displays the condition numbers for the Toeplitz and preconditioned systems, for the three row dimensions, and for $\epsilon$ equal to its largest and smallest values.

Table 3.3 Artificial Matrices: Condition Numbers

<table>
<thead>
<tr>
<th></th>
<th>Toeplitz System</th>
<th>Preconditioned System</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Rows</td>
<td>No. of Columns</td>
<td>No. of Columns</td>
</tr>
<tr>
<td>100</td>
<td>25 50 75</td>
<td>25 50 75</td>
</tr>
<tr>
<td>$\epsilon = 1$</td>
<td>$10^3$ $10^5$ $10^9$</td>
<td>$10^5$ $10^8$ $10^{10}$</td>
</tr>
<tr>
<td>$\epsilon = 10^{-9}$</td>
<td>$10^4$ $10^4$ $10^3$</td>
<td>1.0 1.0 1.0</td>
</tr>
</tbody>
</table>

This experiment is repeated but now the first column of $C$ is formed with elements from analytic Green's functions used by Michaels [1984]. In her elastic
plate experiments the receivers were placed close to the source point, in which case the Green's functions for the horizontal and vertical displacements have only a few well separated peaks. The derived matrix system is already well conditioned and does not need much improvement. Table 3.4, following the same presentation as in Table 3.1, gives the number of iterations until convergence using Michaels' Green's functions.

Table 3.4 Green's Functions From Michaels

<table>
<thead>
<tr>
<th>Number of iterations Until Convergence</th>
<th>Circulant</th>
<th>Toeplitz</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Rows = 200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of Columns</td>
<td>Circulant</td>
<td>Toeplitz</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>(\epsilon = 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precon.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>No Precon.</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(\epsilon = 10^{-9})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precon.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>No Precon.</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precon.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>No Precon.</td>
<td>9</td>
<td>4.4</td>
</tr>
</tbody>
</table>

Notice that for \(\epsilon\) equal to one the preconditioner does not help in two of the three cases for the Toeplitz matrix, although when averaged over all ten values of \(\epsilon\) there is some slight improvement.

We would like to simulate the more ill-conditioned cases when the receivers are much further away from the source. The Green's functions have a lot more peaks, densely scattered throughout the time window. As we do not have the actual Green's functions for these conditions, we form hybrid versions of our own by combining Michaels' Green's functions. We scale, add, and shift her functions to derive our Green's functions. Figures 3.1 and 3.2 display, respectively, our hybrid horizontal and vertical displacement Green's functions.

We repeat our experiment, forming the right hand side vector \(s\) so that at the solution the problem has a small residual. Table 3.5 shows the results for the lower tolerance case.

Table 3.5 Hybrid Green's Functions: Tolerance = \(10^{-4}\)

<table>
<thead>
<tr>
<th>Number of iterations Until Convergence</th>
<th>Circulant</th>
<th>Toeplitz</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Rows = 200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of Columns</td>
<td>Circulant</td>
<td>Toeplitz</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>(\epsilon = 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precon.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>No Precon.</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>(\epsilon = 10^{-9})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precon.</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>No Precon.</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precon.</td>
<td>10.2</td>
<td>13.1</td>
</tr>
</tbody>
</table>

For every column dimension size, the Toeplitz matrices need only one iteration with a preconditioner to converge when \(\epsilon\) is equal to \(10^{-4}\). Table 3.6 displays the number of iterations for the tighter tolerance level.
Table 3.6 Hybrid Green’s Functions: Tolerance = $10^{-9}$

<table>
<thead>
<tr>
<th>Number of Iterations Until Convergence</th>
<th>Toeplitz</th>
<th>Circulant</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Rows = 200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of Columns</td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>$\epsilon = 1$</td>
<td>Precon.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No Precon.</td>
<td>19</td>
</tr>
<tr>
<td>$\epsilon = 10^{-9}$</td>
<td>Precon.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No Precon.</td>
<td>19</td>
</tr>
<tr>
<td>Average</td>
<td>Precon.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No Precon.</td>
<td>19</td>
</tr>
</tbody>
</table>

The noticeable difference between this experiment and the artificial matrix version is in those cases with no preconditioner. Many fewer iterations are required to achieve convergence because the span between the largest and smallest eigenvalues is much smaller when we compose the matrices out of the Green’s functions. This behavior shows up in Table 3.7, which displays the condition numbers of the Toeplitz and preconditioned systems. Notice that the condition numbers in all the cases are much smaller than their counterparts in Table 3.3.

Table 3.7 Hybrid Green’s Functions: Condition Numbers

<table>
<thead>
<tr>
<th></th>
<th>Toeplitz System</th>
<th>Preconditioned System</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Rows</td>
<td>No. of Columns</td>
<td>No. of Columns</td>
</tr>
<tr>
<td>200</td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>$\epsilon = 1$</td>
<td>15.0</td>
<td>34.0</td>
</tr>
<tr>
<td>$\epsilon = 10^{-9}$</td>
<td>16.6</td>
<td>27.8</td>
</tr>
</tbody>
</table>

It is very reassuring to observe a dramatic decrease in the number of iterations when we invoke the preconditioned conjugate gradient code. However, we should note that the use of a preconditioner involves a startup cost, namely factoring the matrix, and the cost of applying the preconditioner at each iteration. We need to ask what is the crossover point at which the savings from the preconditioner is outweighed by its cost.

The best way to answer this question is to compare flops, which is the number of floating point operations. Since multiplications (and divisions) are more time-consuming on the computer than additions (and subtractions) we will only keep track of the first. To see if the use of a preconditioner saves us any flops we check for the percentage savings as defined by

$$\% \text{ savings} = \frac{\text{flops}_{\text{no pre}} - \text{flops}_{\text{pre}}}{\text{flops}_{\text{no pre}}} \times 100.$$ 

We need to calculate flops$_{\text{no pre}}$ and flops$_{\text{pre}}$. Assume that our matrix $B_i$ has dimensions $n \times p$. The predominate cost of calculating a Cybenko factorization of $B_i$ is
the cost per iteration of the CG code using a preconditioner is
\[ p^2 + 3np + 4p + 5n, \]
and without a preconditioner is
\[ -p^2 + 3np + 3p + 5n. \]
Suppose that the PCG code takes \( c \) iterations, and that the CG code takes \( d \) iterations. Then
\[ \text{flops}_{\text{no pre}} = d[-p^2 + 3np + 3p + 5n] \]
and
\[ \text{flops}_{\text{pre}} = 5np + p^2 + c[p^2 + 3np + 4p + 5n]. \]
Now we set
\[ \frac{\text{flops}_{\text{no pre}} - \text{flops}_{\text{pre}}}{\text{flops}_{\text{no pre}}} > \alpha \quad (3.5.3) \]
where \( \alpha \in [0, 1] \), and solve for \( d \) in terms of \( c \). This will inform us how many fewer iterations we need in order to benefit from the preconditioner. The number of columns \( p \) is smaller than the number of rows \( n \), and so we assign
\[ p = \frac{n}{\gamma} \quad \text{for} \quad \gamma \in [1, \frac{n}{p}] \quad (3.5.4) \]
Then (3.5.3) simplifies to the approximation
\[ 1 - \frac{c+2}{d} > \alpha \]
so that
\[ d > \frac{c+2}{1-\alpha}. \]
To achieve a percentage savings of more than 50%, \( d \) must be greater than \( 2c+4 \). For a 10% savings \( d \) must be greater than \( c+2 \). On reviewing the previous set of tables we see that we achieve at least a 30% savings in almost all cases, sometimes as high as 90%.

Suppose that instead of using a PCG method we form the normal equations and use Gaussian elimination to reduce \( B^TB \) to a triangular form, then apply back substitution to calculate the solution. All the calculations involved in this direct method comprise \( \text{flops}_{\text{GE}} \). We investigate the relationship between the row dimension size and the efficiency of this direct method versus our iterative method.

We assume that the PCG code takes \( c \) iterations, and we find conditions on \( c \) in terms of \( n \) in the expression
\[ \text{flops}_{\text{GE}} - \text{flops}_{\text{pre}} > 0. \quad (3.5.5) \]
Using equality (3.5.4) to help simplify (3.5.5) eventually yields
\[ c < \frac{(1+3\gamma) - \epsilon}{(3\gamma)\epsilon}, \quad (3.5.6) \]
where \( \epsilon \) approaches zero as \( n \) gets large. Inequality (3.5.6) can be written approximately as
\[ c < \frac{1}{\epsilon}, \quad \text{where} \quad \epsilon \to 0 \quad \text{as} \quad n \to \infty. \quad (3.5.7) \]
The interpretation of (3.5.7) is that as long as \( \epsilon \) is less than \( 1/\epsilon \), we attain a savings in \( \text{flops} \) by using our iterative method with a preconditioner. As \( n \) grows, so does \( 1/\epsilon \), and the gap in \( \text{flops} \) between the two methods is even more pronounced.
We conclude that our preconditioner devised from Cybenko's factorization of a circulant matrix significantly speeds up the convergence of a Toeplitz system and lowers the computational costs.

Figure 3.1 Actual Solution
Figure 3.2: Hybrid version of horizontal displacement.

Figure 3.2: Solution using Cooley and Tukey's method.
CHAPTER 4

Nonlinear Deconvolution Problems

In this chapter we discuss various ways of solving the nonlinear least squares (NLS) problem

\[
\min_{y \in \mathbb{R}^n} f(y) = \frac{1}{2} \| R(y) \|_2^2,
\]

(NLS)

where \( R(y): \mathbb{R}^p \rightarrow \mathbb{R}^q \). The natural choice of algorithm is to use Newton's method to solve for \( y \) in

\[
\nabla f(y) = 0.
\]

Section 4.1 presents several variations of Newton's method to ensure convergence to a local minimizer from any starting point, and to simplify the computation. Section 4.2 discusses the idea of solving the problem coarsely when the iterates are far away from the solution, gradually achieving higher accuracy as they approach the solution.

The particular nonlinear problem under consideration is bilinear. Section 4.3 presents our applications. Algorithms that solve problem NLS, that take advantage of the inherent structure in the problem are presented in Section 4.4. One drawback to bilinearity is that there may not be a unique solution, and in Section 4.5 we show how to handle this difficulty.
4.1. The Quasi-Newton Framework

In this section we discuss various implementations of Newton's method for solving problem NLS. The background material presented in Section 4.1 can be found in more detail in Dennis and Schnabel [1983, Chapter 6].

Problem NLS is a special case of the general unconstrained minimization problem

$$\min_{y \in \mathbb{R}^p} f(y) : \mathbb{R}^p \to \mathbb{R}.$$  

Algorithm 4.1 outlines the basic Newton's method.

**ALGORITHM 4.1 Newton's Method for Unconstrained Minimization.**

Given $y_0 \in \mathbb{R}^p$, $f(y) : \mathbb{R}^p \to \mathbb{R}$.

For $j = 0, 1, \ldots$, until convergence determine

a) Solve $\triangledown^2 f(y_j) s_j = -\triangledown f(y_j)$ for $s_j$,

b) $y_{j+1} = y_j + s_j$,

c) Test for convergence.

End.

There are several drawbacks to this algorithm:

1. It is not globally convergent for many problems. For many starting points, the iterates do not converge to a local minimizer.

2. It requires computing the Hessian $\triangledown^2 f$ of $f$ at each iteration; this can involve a considerable amount of computing.

3. It involves solving a system of linear equations which may be ill-conditioned or singular.

We would like to eliminate these drawbacks and yet retain a strong point of the algorithm, that is be q-quadratically convergent from starting points close enough to the solution. The end result is a hybrid algorithm which always tries Newton's method first. If $f$ decreases enough then take the step; otherwise use some type of global strategy to determine what step to take. The global strategy tries to eliminate all the drawbacks of Algorithm 4.1, and yet retain fast convergence locally. An algorithm of this hybrid nature is called quasi-Newton.

**ALGORITHM 4.2 Quasi-Newton Method for Unconstrained Minimization.**

Given $y_0 \in \mathbb{R}^p$, and $f(y) : \mathbb{R}^p \to \mathbb{R}$.

For $j = 0, 1, \ldots$, until convergence determine

1) Compute $\triangledown f(y_j)$ and test for convergence,

2) Compute $H_j$ to be $\triangledown^2 f(y_j)$ or an approximation to it,

3) Perturb $H_j$ if necessary to ensure nonsingularity and good conditioning,

4) Solve $H_j s_j = -\triangledown f(y_j)$ for $s_j$,

5) Decide whether to take a Newton step $y_{j+1} = y_j + s_j$, or to choose $y_{j+1}$ by a global strategy.

End.
There are two schools of thought about the global strategy in step (5) of Algorithm 4.2. The first is the *line search* approach that proposes backtracking along the quasi-Newton direction $s_j$ until we reach an acceptable step. The framework for performing this search is given below.

**Algorithm 4.3** Backtracking.

Given $\alpha \in (0, \frac{1}{2})$, $0 < \nu < \mu < 1$, $\lambda_j = 1$,
while $f(y_j + \lambda_j s_j) > f(y_j) + \alpha \lambda_j \nabla f(y_j)^T s_j$ determine
$\lambda_j \leftarrow \rho \lambda_j$, for some $\rho \in [\nu, \mu]$,
$y_{j+1} \leftarrow y_j + \lambda_j s_j$.

End.

The common values to assign the parameters in Algorithm 4.3 are $\alpha = 10^{-4}$, $\nu = 0.1$ and $\mu = 0.5$.

On the first backtrack we make a quadratic model $m(y_j)$ of our function using $f(y_j)$, $f(y_j + s_j)$, and $\nabla f(y_j)^T s_j$ to determine the $\lambda_j$ such that $m(y_j + \lambda_j s_j)$ is minimized. If the step $y_j + \lambda_j s_j$ is still unacceptable, then we make a cubic model of our function, and continue to do so until we reach an acceptable point, or exceed our backtracking limit.

The other school of thought about a global strategy is the *model trust region (MTR)* approach which specifies a region in which we trust our model of $f(y)$. In a line search we shorten the step length in the Newton direction. In the MTR approach, if the Newton step is unacceptable then we choose an
acceptable step, in another direction, with bounded length. At $y_\epsilon$ we try to find an $s$ satisfying

$$
\begin{align*}
\text{minimize } & m(y_\epsilon + s) = f(y_\epsilon) + \nabla f(y_\epsilon)^T s + \frac{1}{2} s^T H s \\
\text{subject to } & \|s\|_2 \leq \delta_\epsilon.
\end{align*}
$$

(4.1.1)

Levenberg [1944] and Marquardt [1963] are responsible for first applying the MTR idea to solving problem NLS. There are various ways of solving for $s$ in (4.1.1), and the solution fits into the MTR framework via Algorithm 4.4.

**Algorithm 4.4** The Model Trust Region Step.

Given $y_\epsilon \in \mathbb{R}^p$, $f(y) : \mathbb{R}^p \to \mathbb{R}$, $\delta_\epsilon > 0$,
$H_\epsilon \in \mathbb{R}^{p \times p}$ symmetric and positive definite.

Do until $y_\epsilon$ is an acceptable next point

1) $s_\epsilon$ is an approximate solution to (4.1.1),
2) $y_\epsilon \leftarrow y_\epsilon + s_\epsilon$,
3) Calculate new value of $\delta_\epsilon$.

End do

4) $\delta_\epsilon \leftarrow \delta_\epsilon$.

When $f$ is the Euclidean norm of residuals, as in problem NLS then, letting

$$
J(y) = \nabla R(y),
$$

we have

$$
\nabla f(y) = J(y)^T R(y),
$$

(4.1.2)

and
\[
\n\nabla^2 f(y) = J(y)^T J(y) + \sum_{i=1}^n R_i(y) + \nabla^2 R_i(y).
\]

(4.1.3)

In this study we use the Gauss-Newton approximation to the Hessian

\[ H_k = J(y_k)^T J(y_k), \]

throwing away the second order information about \( R(y) \).

The algorithm framework for a quasi-Newton method on problem NLS with a MTR approach is given below.

**ALGORITHM 4.5 Quasi-Newton Method for Problem NLS.**

Given \( y_0 \in \mathbb{R}^p \) and \( R(y) : \mathbb{R}^p \to \mathbb{R}^n \).

For \( j = 0, 1, \ldots \), until convergence determine

1) compute \( \nabla f(y_j) \) and test for convergence,

2) solve for the Gauss-Newton step \( s_J^N \) in

\[
J(y_j)^T J(y_j) s_{J}^N = -J(y_j)^T R(y_j),
\]

(4.1.4)

3) \( y_{j+1} = y_j + s_{J}^N \) if acceptable

else \( y_{j+1} = y_j + \delta_j \) where \( \delta_j \) is found by Algorithm 4.4.

End.

4.2. Inexact Newton Methods

One of the drawbacks to Algorithm 4.5 is that solving (4.1.4) can be very expensive. Dembo, Eisenstat, and Steihaug [1982] propose a class of inexact Newton methods. Rather than solve for \( s \) exactly in (4.1.4), they solve for an \( s \) satisfying

\[
J_k^T J_k s_k = -J_k^T R_k(y_k) + \epsilon_k,
\]

where

\[
\frac{||\epsilon_k||}{||J_k^T R(y_k)||} \leq \eta_k, \text{ and } J_k = J(y_k).
\]

By controlling the size and rate of decrease of \( \eta_k \) they can control the rate of convergence. Dembo and Steihaug [1983] describe how they use the Conjugate Gradient (CG) methods on (4.1.4) with a trust region implementation. For each \( j \), the CG method stops when

\[
\frac{||r_{j+1}||}{||J(y_j)^T R(y_j)||} \leq \eta_j.
\]

(Refer to Algorithm 3.1 for a description of the CG method.) By choosing

\[
\eta_j = \min\left\{ 1/j, \frac{||J(y_j)^T R(y_j)||}{t} \right\}
\]

for some \( t \in (0,1] \), we have a method of order \( 1 + t \).

4.3. Our Particular Application

This section gives the specifics of how we solve our nonlinear problem. Recall from Section 2.3 that the particular nonlinear problem we are solving is

\[
\minimize_{y \in \mathbb{R}^n} f(y) = \frac{1}{2} ||R(y)||^2.
\]

where

\[
R(y) = \begin{bmatrix}
u^1(y) - \hat{u}^1(y) \\
\vdots \\
u^N(y) - \hat{u}^N(y)
\end{bmatrix},
\]
and \( u^*(y) \) has the form

\[
u^*(y) = \sum_{m=1}^{M} x_m \sum_{j=1}^{N} G_m(u_{j+1}) \alpha(j),\]

where

\[
y = (x_1, \ldots, x_M, \alpha(1), \ldots, \alpha(K)), \quad M + K < N.
\]

When we separate \( y \) into its two sets of unknowns \( x \) and \( \alpha \), then the residual \( R \) can be represented by two different matrix forms:

\[
R(x, \alpha) = A(\alpha) x - b, \quad A(\alpha) \in \mathbb{R}^{N \times M},
\]

\[
= B(x) \alpha - b, \quad B(x) \in \mathbb{R}^{N \times K}, \quad (4.3.1)
\]

Note that \( b \) is the vector \((u^1, \ldots, u^N)^T\) of measured values of \( u \).

It’s important to study the form of the Jacobian \( J \) of \( R \). If we make the following assignments

\[
A_{ij} = G_j(i) \alpha(1) + \cdots + G_j(i) \alpha(i),
\]

\[
\overline{A}_{ij} = G_j(i) \alpha(1) + \cdots + G_j(i-K+1) \alpha(K),
\]

\[
B_i = G_j(i) x_1 + \cdots + G_M(i) x_M,
\]

then \( J \) has the form

\[
J = \begin{bmatrix}
A_{11} & \cdots & A_{1M} & B_1 & 0 & \cdots & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
A_{K1} & \cdots & A_{KM} & B_K & B_{K-1} & \cdots & B_1 \\
\overline{A}_{K+1,1} & \cdots & \overline{A}_{K+1,M} & B_{K+1} & B_K & \cdots & B_2 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\overline{A}_{N,1} & \cdots & \overline{A}_{NM} & B_N & B_{N-1} & \cdots & B_{N-K+1}
\end{bmatrix}
\]

The Jacobian is the concatenation of the matrices \( A \) and \( B \)

\[
J(y) = \begin{bmatrix} A(\alpha) & B(x) \end{bmatrix}.
\]

Notice that \( B \) is a Toeplitz matrix and that \( A \), although dense, has few columns and so is a rectangular matrix with a large number of rows.

Solving (4.1.4) by the CG method is ideal for this application because we can adapt our preconditioner of Section 3.4. Using (4.3.2) we have

\[
J^T J = \begin{bmatrix} A^T A & A^T B \\
B^T A & B^T B \end{bmatrix}.
\]

We expand our preconditioner \( TD^{-1/2} \) by the identity and zero matrices to obtain our extended preconditioner

\[
F^{-1} = \begin{bmatrix} I & 0 \\
0 & TD^{-1/2} \end{bmatrix}.
\]

The preconditioned matrix in (4.1.4) is

\[
P \equiv (F^{-1})^T J^T J F^{-1}
\]

\[
= \begin{bmatrix} I & 0 \\
0 & D^{-1/2} T^{-T} \end{bmatrix} \begin{bmatrix} A^T A & A^T B \\
B^T A & B^T B \end{bmatrix} \begin{bmatrix} I & 0 \\
0 & TD^{-1/2} \end{bmatrix}
\]

\[
= \begin{bmatrix} A^T A & A^T B \overline{T} \overline{D}^{-1/2} \\
\overline{A}^T B \overline{T} \overline{D}^{-1/2} & \overline{A}^T \overline{B} \overline{T} \overline{D}^{-1/2} \end{bmatrix}.
\]

If \( B \) were a circulant matrix then \( P \) would reduce to

\[
P = \begin{bmatrix} A^T A & A^T Q \\
Q^T A & I \end{bmatrix},
\]

which we suspect has more clustered eigenvalues than \( J^T J \). The results in
Chapter 5 confirm that our preconditioner $F^{-1}$ can greatly reduce the amount of work required for Algorithm 4.5.

4.4. The Semilinear Least Squares Problem

Most of the algorithms presented thus far do not take advantage of the inherent structure in our application. The variables can be divided into two groups such that if the variables in one of the groups were held constant, the other variables could be calculated accurately and cheaply. We refer to a problem with this feature as being separable. By looking at our particular form of $R$ we see that the nonlinearity is caused by the product of two linear sets of variables. Hence holding either group constant results in solving a linear least squares problem. This bilinear form is a subset of the semilinear least squares problem discussed in Parks [1985]. The general form is

$$
\min_{x \in \mathbb{R}^N, \alpha \in \mathbb{R}^k} f(x, \alpha) = \frac{1}{2} \| R(x, \alpha) \|^2_2,
$$

(4.4.1)

where

$$
R(x, \alpha) = A(\alpha)x + b, \quad A(\alpha) \in \mathbb{R}^{N \times M}.
$$

(4.4.2)

To understand how the various algorithms manipulate the structure in (4.4.1) it would help if we rewrite parts of Algorithm 4.5 in factored form. Recall steps (2) and (3):

1. solve for $s$ in $J^T Js = -J^T R$,
2. $y_+ = y_+ + s$.

Using (4.4.2) we have a factored form for the Jacobian $J$ of $R$.

$$
J = \begin{bmatrix} J_x & J_\alpha \end{bmatrix} = \begin{bmatrix} A(\alpha) & DA(\alpha) \end{bmatrix},
$$

(4.4.3)

where $DA$ is the Fréchet derivative of $A$. We also divide $s$ into its $x$ and $\alpha$ components

$$
s = (s_x, s_\alpha)^T.
$$

Expanding the expression for $s$ using the factored form of $J$ and $s$ gives two sets of equations in the two sets of unknowns.

$$
A^T As_x + A^T J_\alpha s_\alpha = -A^T R,
$$

(4.4.4)

$$
J^T_\alpha As_x + J^T_\alpha J_\alpha s_\alpha = -J^T_\alpha R.
$$

(4.4.5)

To eliminate $s_x$ from the equations we multiply (4.4.4) on the left by $J^T_\alpha (A^+)^T$, and subtract the result from (4.4.5), to yield

$$
J^T_\alpha (I - AA^+)J_\alpha s_\alpha = -J^T_\alpha (I - AA^+)R.
$$

(4.4.6)

Here, $A^+$ satisfies the four Penrose [1955] conditions for a generalized matrix inverse:

$$
AA^+A = A,
$$

$$
A^+AA^+ = A^+,
$$

$$
(A^+A)^T = A^+A,
$$

$$
(AA^+)^T = AA^+.
$$

Define

$$
Q_A = I - AA^+
$$

as the projection onto the orthogonal complement of the row space of $A$.

Substituting $Q_A$ into (4.4.6) gives
\[ J_a^T Q_A J_a s_a = -J_a^T Q_A R. \]  \hspace{1cm} (4.4.7)

Since a projection matrix is symmetric and idempotent, we can rewrite (4.4.7) as

\[ (Q_A J_a)^T (Q_A J_a) s_a = -(Q_A J_a)^T Q_A R. \]  \hspace{1cm} (4.4.8)

After solving for \( s_a \) in (4.4.8) set

\[ s_x = -A^T (R + J_a s_a), \]

and update \( x \) and \( \alpha \) by

\[ x_{j+1} = x_j + s_x, \]
\[ \alpha_{j+1} = \alpha_j + s_a. \]

Solving for \( s_x \) and \( s_a \) in this manner is just the Gauss-Newton method, written in factored form. Now we are in a position to compare this to other algorithms.

Wold and Lyttkens [1969], in statistical applications, solve for \( x \) given a fixed \( \alpha \), using linear least squares. Then they solve a minimization problem in \( \alpha \), holding \( x \) constant.

**Algorithm 4.6 Alternation.**

Given \( \alpha_0 \in \mathbb{R}^K \).

For \( j = 0, 1, \ldots, \) until convergence determine

1) Solve for \( x_j \) in

\[ \min_{x \in \mathbb{R}^d} \frac{1}{2} \| A(\alpha_j) x - b \|_2^2, \]

2) Test for convergence,

3) Solve \( J_a^T J_a s_a = -J_a^T R \) for \( s_a \),

4) \( \alpha_{j+1} = \alpha_j + s_a \).

End

This method does not use all of the available information about \( x \) when updating \( \alpha \). As an improvement Barham and Drane [1972] propose computing a Gauss-Newton step for \( \alpha \), and choosing as the next \( x \) an optimal partner of \( \alpha \).
Algorithm 4.7 Modified Alternation

Given $\alpha_0 \in \mathbb{R}^N$

For $j = 0, 1, \ldots$, until convergence do

1) Solve for $x_j$ in

$$\min_{x \in \mathbb{R}^N} \frac{1}{2} \| A(\alpha_j)x - b \|_2^2,$$

2) Test for convergence.

3) Solve $(Q_AJ_\alpha)^T(Q_AJ_\alpha)s_\alpha = -(Q_AJ_\alpha)^TQ_AR$.

4) $\alpha_{j+1} = \alpha_j + s_\alpha$.

End.

Golub and Pereyra [1973, 1976] eliminate $x$ entirely, producing the variable projection functional $f$

$$f(\alpha) = f(x(\alpha), \alpha)$$
$$= \frac{1}{2} \| A(\alpha)(A(\alpha)^*b) - b \|_2^2$$
$$= \frac{1}{2} \| -Q_A(\alpha)b \|_2^2$$

The reduced problem now is

$$\min_{\alpha \in \mathbb{R}^k} f(\alpha) = \frac{1}{2} \| -Q_A(\alpha)b \|_2^2,$$

which reduces the original problem by $m$ variables, but increases the complexity.

Algorithm 4.8 Reduced Problem.

1) Solve for $\hat{\alpha}$ to

$$\min_{\alpha} f(\alpha) = \frac{1}{2} \| -Q_A(\alpha)b \|_2^2.$$

2) Solve for $x$ in

$$A(\hat{\alpha})^TA(\hat{\alpha})x = A(\hat{\alpha})^Tb.$$

Ruhe and Wedin [1980] compare the convergence rates of these algorithms for solving the semilinear least squares problem. The Alternation Algorithm converges $r$-linearly, whereas Algorithms 4.7 and 4.8 converge $r$-superlinearly in general. Locally the latter two algorithms converge $q$-linearly, and display $q$-quadratic convergence on problems in which the residual at the solution is zero.

4.4.1. The Variable Projection Functional

In order to solve the minimization problem in step (1) of Algorithm 4.8 we need an expression for the gradient of $f(\alpha)$, which contains

$$D(-Q_A(\alpha)b) = D(\Lambda A^+b - b) \equiv D(\Pi)$$

Golub and Pereyra [1973, 1976] compute the Fréchet derivatives of pseudoinverses. We can simplify their expressions somewhat because in our case $A(\alpha)$ and $A(\alpha)^+$ have full rank. We use the equalities

$$DA^{-1} = -A^{-1}(DA)A^{-1},$$
$$A^+ = (A^TA)^+A^T$$

to derive an expression for $DA^+$. 
\[ DA^+ = D[(A^T A)^{-1} A^T] \]
\[ = D[(A^T A)^{-1}]A^T + (A^T A)^{-1}D A^T \]
\[ = -(A^T A)^{-1}D(A^T A)(A^T A)^{-1}A^T + (A^T A)^{-1}D A^T \]
\[ = -(A^T A)^{-1}D(A^T A)A + A^TDA[(A^T A)^{-1}A^T] + (A^T A)^{-1}D A^T \]
\[ = -(A^T A)^{-1}D(A^T A)A + A^TDA + (A^T A)^{-1}D A^T \]
\[ = (A^T A)^{-1}D A^T[I - AA^+] - A^+DAA^+. \]

We use the final expression for \( DA^+ \) to derive an expression for \( D[AA^+b - b] \):

\[ \bar{J} = \bar{D} \bar{R} = D[AA^+b - b] = [DA A^+] + (DA)A^+b \]
\[ = A[(A^T A)^{-1}D A^T[I - AA^+]] - A^+DA A^+b + (DA)A^+b \]
\[ = (A^T)DA^TQ_A b + AA^+DAA^+b + DAA^+b \]
\[ = -(A^T)DA^TQ_A b + Q_A(DA)A^+b \]
\[ = -(A^T)DA^T\bar{R} + Q_A(DA)A^+b. \]

Actually, when we substitute \( \bar{J} \) into

\[ \nabla f = \bar{J}^T \bar{R}, \]

we obtain the simpler expression

\[ \nabla f = \bar{J}^T \bar{R} = [-(\bar{R}^T(DA)A^+ + (A^+b)^T D A^T Q_A)[(-Q_A)b)] \]
\[ = \bar{R}^T DA[A^+Q_A b] - (A^+b)^T D A^T Q_A b \]
\[ = 0 - (A^+b)^T D A^T Q_A b \]
\[ = (A^+b)^T D A^T \bar{R}. \]

In this derivation we use the equality

\[ D(A^T) = (DA)^T, \]

which is straightforward, though tedious, to show.

Now we are ready to apply the Gauss-Newton method and globalization techniques to minimize the variable projection functional \( f \).

Can we simplify this procedure somewhat when \( R \) is bilinear as in (4.3.1)?

The answer is in the affirmative. Suppose that we change the role of the matrices \( A \) and \( B \) in the Reduced Problem Algorithm, by eliminating \( \alpha \) instead of \( x \). Recall that \( B \) is Toeplitz, so that we can solve step (2) rather quickly using the PCG method with our preconditioner of Section 3.4. Using a Gauss-Newton or quasi-Newton method on the minimization portion involves computing the action of \( J \) on a vector \( v \).

\[ \bar{J}v = -(B^+)^T DB^T \bar{R} + Q_BDB^+b \]
\[ = -(B^+)^T G + H \]
\[ = -(B^+)^T g + h. \]

Here, \( G \) and \( H \) are computable matrices. Their components are either known constants or elements which have been previously computed. The last step is written in terms of the vectors \( g \) and \( h \) where \( g = Gu \) and \( h = Hv \). To evaluate \( \bar{J}v \), solve

\[ (B^T B)w = g. \]

Then

\[ \bar{J}v = Dw, \]

which implies that we need to solve an extra system of equations to evaluate the action of \( J \) on a vector. To accomplish this we just use PCG when solving for \( w \) to make the process less costly.
4.5. Nonuniqueness

If we consider any component of the residual $R$,

$$R_n(y) = u^n(y) - \hat{u}^n(y)$$

$$= \sum_{m=1}^{M} \sum_{j=1}^{n} G_m(n-j+1)\alpha(j) - \hat{u}^n(y),$$

we notice that there are infinitely many $y$'s that give the same value of $R_n(y)$.

If we replace $x_j$ by $x_j/c$ and $\alpha(j)$ by $c \alpha(j)$, for any nonzero constant $c$, the value of $u^n(y)$ does not change. Consequently, the Jacobian matrix $J$ of $R$ has a zero eigenvalue, so that $J^TJ$ is positive semidefinite.

We use both matrix forms of $R$

$$R(y) = A(\alpha)x - b = B(x)\alpha - b$$

to find the eigenvector corresponding to the null eigenvalue. We have

$$J = [J_x, J_\alpha] = [A(\alpha), B(x)],$$

and we want to find a $v$ satisfying $Jv = 0$. Hence we want

$$\begin{bmatrix} A(\alpha) & B(x) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = A(\alpha)v_1 + B(x)v_2 = 0,$$

where $v_1 \in \mathbb{R}^M$ and $v_2 \in \mathbb{R}^K$.

Notice that $A(\alpha)x = B(x)\alpha$ so that we clearly have the form of a null eigenvector

$$v_0 = (x, -\alpha)^T.$$ \hspace{1cm} (4.5.1)

That is, when evaluating $J(y)$, a null eigenvector has the form

$$v_0 = (y_1, \ldots, y_M, -y_{M+1}, \ldots, -y_{M+K})^T.$$ 

Solving for $s$ in $J^TJ_s = -J^TR$ is highly unstable due to the zero eigenvalue.

We propose two remedies to deal with its presence, but first we show that there is one and only one null eigenvalue in our problems.

**Theorem 4.1** Suppose that, for any two vectors $x^1$ and $x^2$ satisfying

$$\|x^1\| = \|x^2\| = 1,$$

we have that

$$B(x^1)\alpha^1 = B(x^2)\alpha^2$$

implies

$$\alpha^1 = \alpha^2 = 0.$$ 

That is, $B(x)$ has full rank for all $x$ with norm 1, and

$$\text{Range } B(x^1) \cap \text{Range } B(x^2) = \{0\}.$$ 

If $x \neq 0$ and $\alpha \neq 0$ then the Jacobian $J$ has a one-dimensional null-space spanned by the null vector described in (4.5.1).

**Proof.** Without loss of generality, take $\|x\| = 1$. A null vector $(\delta x, \delta \alpha)$ of $J$ satisfies

$$B(\delta x)\alpha + B(x)\delta \alpha = 0.$$ 

(4.5.2)

If $\delta x = 0$, then $\delta \alpha = 0$ also because of the full rank of $B(x)$. Thus, we assume that $\delta x \neq 0$, and rewrite the null-vector equation (4.5.2) as

$$B(\delta \frac{x}{\|x\|})(-\|\delta x\|\alpha) = B(x)\delta \alpha.$$ 

Suppose first that

$$x = \pm \frac{\delta}{\|\delta x\|}.$$
Then

\[ B(x) \| \delta x \| \alpha = 0. \]

So

\[ \delta x = \pm \| \delta x \| \alpha, \]

and

\[ (\delta x, \delta \alpha) = \pm \| \delta x \| (x, \alpha). \]

That is, \((\delta x, \delta \alpha)\) is the null vector identified in (4.5.1).

Alternatively suppose that \(x\) is not collinear with \(\delta x\). Then the hypothesis implies that

\[ \delta x = \| \delta x \| \alpha = 0. \]

Since \(\alpha \neq 0\), this contradicts the assumption that \(\delta x \neq 0\).

Using results proved in Symes [1983] one can show that the elastic Green's functions used by Michaels and by us in this study fulfill the hypotheses of the theorem.

Thus we can use the following device to restore full rank to our problem.

**Theorem 4.2** Let the \(m \times n\) matrix \(A\) have rank \(n-1\), and let \(v \in \mathbb{R}^n\) be an eigenvector associated with the zero eigenvalue of \(A^T A\). Let \(\tilde{x}\) be the minimum norm solution to \(Ax - \tilde{b}\). Let \(\tilde{A}\) and \(\tilde{v}\) be the augmented matrix and vector, respectively,

\[
\tilde{A} = \begin{bmatrix} A \\ v^T \end{bmatrix} \in \mathbb{R}^{(m+1) \times n},
\]

\[
\tilde{b} = \begin{bmatrix} b \\ 0 \end{bmatrix} \in \mathbb{R}^{m+1}.
\]

Then

1. \(\tilde{A}\) has full rank, and
2. \(\tilde{x}\) is the minimum norm solution to \(\tilde{A}x - \tilde{b}\).

**Proof.** Suppose there is a vector \(y \in \mathbb{R}^n\) such that \(\tilde{A}y = 0\), that is,

\[
\begin{bmatrix} Ay \\ v^T y \end{bmatrix} = 0 \rightarrow v^T y = 0.
\]

Let \(y = y_r + y_n\), where \(y_r \in R(A)\), the range of \(A\), and \(y_n \in N(A)\), the null space of \(A\). Then

\[
0 = Ay = A(y_r + y_n) = Ay_r + Ay_n = Ay_r.
\]

So \(y_r = 0\) and \(y = y_n\). Since \(y\) is in the null space of \(A\) it must be a multiple of the null eigenvector \(v\). So \(y_n = cv\) for some constant \(c\). Then

\[
0 = v^T y = v^T cv = cv^T v = c \| v \| \rightarrow c = 0 \text{ for nonzero } v.
\]

Thus \(y_n = 0v = 0\) and \(y = 0\). We have shown that if \(\tilde{A}y = 0\) then \(y = 0\), which implies that \(\tilde{A}\) has full rank.

Since \(\tilde{x}\) is the minimum norm solution to \(\tilde{A}x - \tilde{b}\), \(\tilde{x}\) has the form

\[
\tilde{x} = (U \Sigma V^T)^+ \tilde{b} = V \Sigma^+ U^T \tilde{b},
\]

where \(U \Sigma V^T\) is the Singular Value Decomposition of \(A\) (Strang [1980]). The minimum norm solution to \(\tilde{A}x - \tilde{b}\) must satisfy

\[
\tilde{A}^T \tilde{A}x = \tilde{A}^T \tilde{b}.
\]

Expanding this equation gives
\[(A^T v) \begin{bmatrix} A^T \\ v^T \end{bmatrix} x = (A^T v) \begin{bmatrix} b \\ 0 \end{bmatrix} \]
\[= (A^T A + vv^T) x = A^T b.\]

Notice that
\[v^T \hat{x} = v^T \Sigma^+ U^T b = e_n \Sigma^+ U^T b = 0 U^T b = 0.\]
Consequently \(vv^T \hat{x} = 0\) which implies that
\[(A^T A + vv^T) \hat{x} = A^T A \hat{x} = A^T b,\]
which completes the proof.

A different approach to elimination of the rank degeneracy is embodied in the following Corollary to Theorem 4.1.

**Corollary 4.3** Under the hypothesis of Theorem 4.1, the derivative of the restricted map

\[R : SM^{-1} \times \mathbb{R}^K \rightarrow \mathbb{R}^N\]

has full rank, where

\[SM^{-1} = \{x \in \mathbb{R}^N : \|x\| = 1\}\]
in the unit \((M-1)\) sphere.

**Proof.** The derivative of \(R\) is the restriction of \(R\) to the tangent space of \(SM^{-1} \times \mathbb{R}^K\). In particular, for a tangent vector \((\delta x, \delta \alpha)\) at \((x, \alpha)\), we have \(\delta x^T x = 0\).

Since the null space of \(J\) is the line through \((- x, \alpha)\), its intersection with the tangent space is \{(0,0)\}. Thus, the Jacobian \(\tilde{J}\) of \(\tilde{R}\) has full rank.

## CHAPTER 5

**Results In The Nonlinear Case**

This chapter features the results in solving the nonlinear least squares problem presented in Section 4.2. Recall that the NLS problem is to minimize the Euclidean norm of the residual \(R:\)

\[
\text{minimize}_{z \in \mathbb{R}^N, \alpha \in \mathbb{R}^K} f(x, \alpha) = \frac{1}{2} \| R(x, \alpha) \|_2^2 \tag{NLS}
\]

where

\[R(x, \alpha) = B(x) \alpha - b \in \mathbb{R}^N.\]

The residual represents the difference between a collection of measured values and the computed values. The Green's functions used are our hybrid versions discussed in Section 3.5. The two vectors \(x\) and \(\alpha\) can be solved simultaneously as if they were one vector \((x, \alpha)\), in which case we are solving the full problem. Or, \(x\) and \(\alpha\) can be treated separately, in which case we are solving what we refer to as the separable problem.

We use three algorithms: one designed for the full problem and two designed for the separable problem. Section 5.1 describes the test performed, and Sections 5.2, 5.3, and 5.4 give the results for each of the three algorithms. Section 5.5 gives a brief summary of our results.
5.1. A Description of the Tests

In all the test cases for the NLS problem $N$ equals 200 and $M$ equals 2. We vary the size of $K$ over the four values 23, 48, 73, and 98. Recall that $a(k), k=1, \ldots, K$ represents the time variable. The preconditioner may be zeroing out elements of proportionally large magnitude. When $K$ is large the preconditioner zeroes out a large portion of the matrix, possibly losing too much information.

The number of columns in the Jacobian is $K+M$, which varies over the four values 25, 50, 75, and 100. We record the number of iterations until convergence, the value of $f$, and the norm of the gradient of $f$. Comparing the number of iterations between algorithms is meaningless because the subproblems we solve at each iteration for the various algorithms are completely different. We include the iteration number as a means of comparison, within a particular algorithm, between the preconditioned and nonpreconditioned system. A much more reliable means of comparing the algorithms is by counting the number of multiplications computed until convergence. Every inner product, norm, matrix-vector product, etc. is included in the count. We would like to emphasize that our objective is not to run an efficiency comparison between the three algorithms as there are many computational aspects which we do not address. The objective is to investigate the savings in each algorithm to be gained by employing our PCG method from Chapter 3.

It may appear that $f$ should be close to zero. This is not the case, and indeed the minimum value of $f$ could have any value, although generally on the small side.

On the other hand, since we are solving for $y$ satisfying $\nabla f(y) = 0$, the norm of $\nabla f$ should be as close to zero as possible.

In each of the algorithms we are repeatedly calculating the solution to the normal equations. Here is where the effects of the preconditioner from Section 3.4 are most noticeable. Its use may not affect the overall number of iterations, but it does decrease the number of multiplications.

We note one important point. Since the matrices considered are either completely or partially Toeplitz, we do not need to store the full matrix. Only the first column of the Toeplitz portion needs to be stored because we have formulas for calculating the matrix-vector products in terms of just the elements in that one column. This savings becomes hidden in the actual listing of the algorithm but it is a distinct advantage.

We now discuss the specific algorithms.

5.2. The Full Problem - INQNTR

This first algorithm uses the ideas presented in Section 4.1 and Section 4.3. It solves the full NLS problem for the unknown vector $y=(x, \alpha)$. We call it INQNTR for a quasi-Newton algorithm using a trust region approach and incorporating the inexact Newton idea. An outline of the algorithm is presented.
below. The superscript \((\infty)\) on the step \(s_j\) denotes the inexact Newton step, and \(J\) denotes the Jacobian of \(R\).

**Algorithm 5.1 INQNTR.**

Given \(y_0 \in R^{M+K}\).

For \(j = 1, 2, \ldots\), until convergence determine

1) Calculate \(R(y_j), f(y_j), J(y_j),\) and \(\nabla f(y_j) = J(y_j)^T R(y_j)\).

2) Test for convergence.

3) Solve for \(s_j^{\infty}\) in \(J(y_j) s_j^{\infty} = -\nabla f(y_j)\)

such that \(\|s_j^{\infty}\| < \delta_j\).

This is the inexact Newton step dependent on \(\eta_j\).

4) \(y_{j+1} = y_j + s_j^{\infty}\).

5) If \(f(y_{j+1}) < f(y_j) + 0.0001 \nabla f(y_j)^T s_j^{\infty}\) then do

5.1) \(\delta_j = 2\delta_j; \) Go to (3)

6) Backtrack, if necessary, along \(s_j^{\infty}\) until we have

an acceptable \(y_{j+1}\).

7) Update \(\delta_j\) and \(\eta_j\).

End.

The Jacobian is singular as it has one zero eigenvalue. Hence, to solve for the step \(s_j^{\infty}\), we augment the Jacobian by a row consisting of a null eigenvector, which is the remedy proposed in Theorem 4.1.

The results are displayed in Table 5.1. The first column has the size of \(K\) which equals the dimension of \(\alpha\). The second column specifies whether a preconditioner is used. The next column is the number of iterations, followed by \(f\) and \(\|\nabla f\|\). Finally, the last column has flops, that is, the number of multiplications computed.

<table>
<thead>
<tr>
<th>K</th>
<th>Prec.</th>
<th>Iter</th>
<th>(f)</th>
<th>(|\nabla f|)</th>
<th>flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>no</td>
<td>6</td>
<td>3.33</td>
<td>.824e-6</td>
<td>1,248,750</td>
</tr>
<tr>
<td>23</td>
<td>yes</td>
<td>6</td>
<td>3.33</td>
<td>.1205-5</td>
<td>1,050,250</td>
</tr>
<tr>
<td>48</td>
<td>no</td>
<td>18</td>
<td>1.48</td>
<td>.295e-5</td>
<td>6,847,700</td>
</tr>
<tr>
<td>48</td>
<td>yes</td>
<td>9*</td>
<td>1.48</td>
<td>.0633e-3</td>
<td>3,041,400</td>
</tr>
<tr>
<td>73</td>
<td>no</td>
<td>32</td>
<td>1.17</td>
<td>.264e-3</td>
<td>14,400,700</td>
</tr>
<tr>
<td>73</td>
<td>yes</td>
<td>19*</td>
<td>1.17</td>
<td>.333e-2</td>
<td>11,217,150</td>
</tr>
<tr>
<td>98</td>
<td>no</td>
<td>17</td>
<td>.987</td>
<td>.727e-4</td>
<td>14,450,650</td>
</tr>
<tr>
<td>98</td>
<td>yes</td>
<td>17</td>
<td>.987</td>
<td>.480e-4</td>
<td>17,010,950</td>
</tr>
</tbody>
</table>

* = stalled: backtracked too many times

To see if the use of a preconditioner saves us any multiplications we check for the percentage savings as defined in Section 3.5. A negative value implies that we actually lose out by using the preconditioner. The benefits of the preconditioner may never surface due to the startup cost of setting it up. Table 5.2 displays the percentage savings caused by the preconditioner for each value of \(K\). With this particular run it is unfair to compare savings for the stalled out cases as the percentage is meaningless. But we include them for completeness.
Table 5.2 INQNTR: Percentage Savings on Small Residual Problem

<table>
<thead>
<tr>
<th>K</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>16</td>
</tr>
<tr>
<td>48</td>
<td>56</td>
</tr>
<tr>
<td>73</td>
<td>22</td>
</tr>
<tr>
<td>98</td>
<td>-22</td>
</tr>
</tbody>
</table>

We now try this algorithm on a problem with consistent data. Table 5.3 gives the results for the zero residual case, followed by Table 5.4, consisting of the percentage savings. The results are even worse in this problem; the preconditioner works against us every time.

Table 5.3 The INQNTR Algorithm: Zero Residual

<table>
<thead>
<tr>
<th>K</th>
<th>Prec.</th>
<th>Iter</th>
<th>f</th>
<th>|V|</th>
<th>flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>no</td>
<td>2</td>
<td>.116e-7</td>
<td>.279e-3</td>
<td>269,430</td>
</tr>
<tr>
<td>48</td>
<td>no</td>
<td>2</td>
<td>.730e-10</td>
<td>.302e-4</td>
<td>497,060</td>
</tr>
<tr>
<td>73</td>
<td>no</td>
<td>2</td>
<td>.386e-11</td>
<td>.850e-5</td>
<td>805,333</td>
</tr>
<tr>
<td>98</td>
<td>no</td>
<td>2</td>
<td>.633e-12</td>
<td>.384e-5</td>
<td>900,308</td>
</tr>
<tr>
<td>23</td>
<td>yes</td>
<td>3</td>
<td>.121e-7</td>
<td>.427e-3</td>
<td>793,540</td>
</tr>
<tr>
<td>48</td>
<td>yes</td>
<td>3</td>
<td>.850e-7</td>
<td>.125e-3</td>
<td>1,278,456</td>
</tr>
<tr>
<td>73</td>
<td>yes</td>
<td>3</td>
<td>.500e-8</td>
<td>.225e-3</td>
<td>1,693,518</td>
</tr>
</tbody>
</table>

Table 5.4 INQNTR: Percentage Savings on Zero Residual Problem

<table>
<thead>
<tr>
<th>K</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>-16</td>
</tr>
<tr>
<td>48</td>
<td>-50.7</td>
</tr>
<tr>
<td>73</td>
<td>-58.8</td>
</tr>
<tr>
<td>98</td>
<td>-71</td>
</tr>
</tbody>
</table>

These results are in sharp contrast to the results for the methods which take advantage of the bilinearity. In the test cases for the next two methods we show considerable savings from using our preconditioner.

5.3. The Alternation Algorithm

This section gives the results of an algorithm geared for the separable characteristic of the NLS problem. We choose the Alternation algorithm described in Section 4.4 as Michaels [1984] has used this algorithm with success. Holding the \( \alpha \)'s constant we solve a linear least squares problem in the \( x \)'s. Then, fixing the \( x \)'s, we solve another linear least squares problem for the \( \alpha \)'s. To deal with the presence of the zero eigenvalue we use the remedy prescribed in Theorem 4.2, namely, we normalize the \( x \)'s.

Michaels actually forms the normal equations in both of the linear least squares subproblems and uses Gaussian elimination to calculate the solution. As we noted in Section 3.5, our PCG code is more efficient than the direct method used by Michaels.
Table 5.5 has an equivalent format to Table 5.1.

Table 5.5 The Alternation Algorithm: Small Residual

| K | Prec. | Iter | $f$     | $ || \nabla f ||$ | flops  |
|---|-------|------|---------|-----------------|--------|
| 23 | no    | 12   | .580    | .987e-5         | 2,143,404 |
| 23 | yes   | 12   | .580    | .947e-5         | 1,289,634 |
| 48 | no    | 10   | .480    | .504e-5         | 4,121,190 |
| 48 | yes   | 10   | .480    | .463e-5         | 2,452,710 |
| 73 | no    | 11   | .452    | .532e-5         | 7,056,278 |
| 73 | yes   | 11   | .452    | .781e-5         | 4,587,523 |
| 98 | no    | 19   | .380    | .768e-5         | 18,735,549 |
| 98 | yes   | 19   | .380    | .950e-5         | 14,347,888 |

Again we compare the savings we achieve with our preconditioner. Notice that the percentage savings peaks for K around 73.

Table 5.6 Alternation: Percentage Savings On Small Residual Problem

<table>
<thead>
<tr>
<th>K</th>
<th>% Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>39.8</td>
</tr>
<tr>
<td>48</td>
<td>40.5</td>
</tr>
<tr>
<td>73</td>
<td>42.3</td>
</tr>
<tr>
<td>98</td>
<td>23.4</td>
</tr>
</tbody>
</table>

We now run this algorithm on a zero residual problem.

Table 5.7 The Alternation Algorithm: Zero Residual

| K  | Prec. | Iter | $f$     | $ || \nabla f ||$ | flops  |
|----|-------|------|---------|-----------------|--------|
| 23 | no    | 7    | 1.15e-7 | .524e-5         | 1,547,112 |
| 23 | yes   | 7    | 1.15e-7 | .524e-5         | 965,283  |
| 48 | no    | 0    | 3.84e-8 | .220e-5         | 5,172,481 |
| 48 | yes   | 0    | 3.84e-8 | .220e-5         | 2,920,511 |
| 73 | no    | 10   | 5.19e-8 | .284e-5         | 11,363,068 |
| 73 | yes   | 10   | 5.19e-8 | .284e-5         | 6,088,035 |
| 98 | no    | 12   | 1.41e-7 | .619e-5         | 17,883,828 |
| 98 | yes   | 12   | 1.41e-7 | .622e-5         | 12,976,364 |

The percentage savings are presented in Table 5.8, and again, the savings peaks for K equal to 73.

Table 5.8 Alternation: Percentage Savings On Zero Residual Problem

<table>
<thead>
<tr>
<th>K</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>37.6</td>
</tr>
<tr>
<td>48</td>
<td>43.4</td>
</tr>
<tr>
<td>73</td>
<td>40.6</td>
</tr>
<tr>
<td>98</td>
<td>27.4</td>
</tr>
</tbody>
</table>

5.4. The Reduced Problem – VPINTR

The third and final algorithm we use is the one proposed by Golub and Pereyra in Section 4.4. We form the reduced problem by eliminating one set of the variables, and minimize the variable projection function for the remaining set of variables. Once again our name for this algorithm is a hybrid of the first
letters in several key words. We call it VPINTR which refers to solving the variable projection functional using an inexact Newton approach with a model trust region implementation.

Dealing with the nonuniqueness problem is subtle and difficult in this approach. The Jacobian of $f$, the function of the reduced problem, has full rank. Yet the iterates change extremely slowly, barely converging. They just zigzag along as in a steepest descent algorithm. To rectify this situation we constrain the solution to the unit sphere, as proposed in theorem 4.2. Recall that the major iterative step in the algorithm is to solve for $\hat{x}$ in

$$
\begin{align*}
\minimize_{x \in \mathbb{R}^d} & f(x) = \frac{1}{2} \| -Q_B(x)b \|^2_2. \\
\text{subject to} & \quad \| x \|_2 = 1.
\end{align*}
$$

We add the constraint that

$$
\| x \|_2 = 1. 
$$

The combined equations (5.4.1) and (5.4.2) form a constrained nonlinear problem which is another whole area of study. In order to retain the unconstrained aspect of the problem we impose a coordinate system locally on the unit ball in $\mathbb{R}^M$.

Suppose that we have a reference point $x_{ref}$ on the unit ball. Let $N$ be a matrix in $\mathbb{R}^{M \times (M-1)}$ whose columns form a basis for the tangent space to the unit ball at $x_{ref}$, and let $y$ be a vector in $\mathbb{R}^{M-1}$. Then we make the following coordinate change

$$
x_c(y) = \frac{x_{ref} + Ny}{\| x_{ref} + Ny \|}
$$

Notice that $\| x_c(y) \| = 1$. Essentially this transformation is picking a plane tangent to the unit ball at the point $x_{ref}$ and bending and curving the plane to fit over a partial area of the ball. When $\| y \|$ becomes too large we recoordinate using our most recent $x_c(y)$ as our new reference point $x_{ref}(y)$.

The unknown variable in (5.4.1) is now $y$ instead of $x$:

$$
\begin{align*}
\minimize_{y \in \mathbb{R}^{M-1}} & f(y) = \frac{1}{2} \| -Q_B(x(y))b \|^2_2.
\end{align*}
$$

We reach a local minimizer when $\nabla f(y) = 0$. An outline of the algorithm is presented below.
Algorithm 5.2 VPINTR.

Given $x_0 \in \mathbb{R}^M$, normalize $x_0$ and set $x_{ref} = x_0$.

Set $y = 0$.

For $j = 1, 2, \ldots$ until convergence determine

1) Calculate $x(y_j)$ using (5.4.3)

2) Calculate $R(x(y_j)), f(x(y_j)), J(y_j),$ and $\nabla f(x(y_j)) = J(y_j)^T R(y_j)$

3) If $\nabla f(x(y_j)) < \text{gradtol}$ then convergence; stop.

4) Solve for $s_j^{n*} \in \mathbb{R}^{M-1}$ in $J(y_j)^T J(y_j) s_j^{n*} = -\nabla f(x(y_j))$

such that $\|s_j^{n*}\| < \delta_j$.

This is the inexact Newton step dependent on $\eta_j$.

5) $y_{j+1} = y_j + s_j^{n*}$. Compute $x(y_{j+1})$.

6) If $f(x(y_{j+1})) < f(x(y_j)) + 0.0001 \nabla f(x(y_j))^T s_j^{n*}$ then accept step.

Else backtrack along $s_j^{n*}$ until we can accept step.

7) If $\|y_j\| > 0.5$ then calculate $x_{ref} = x(y_j)$ and set $y_j = 0$

8) Update $\delta_j$ and $\eta_j$.

End.

This is the inexact Newton step dependent on $\eta_j$.

$y_{j+1} = y_j + s_j^{n*}$. Compute $x(y_{j+1})$.

If $f(x(y_{j+1})) < f(x(y_j)) + 0.0001 \nabla f(x(y_j))^T s_j^{n*}$ then accept step.

Else backtrack along $s_j^{n*}$ until we can accept step.

If $\|y_j\| > 0.5$ then calculate $x_{ref} = x(y_j)$ and set $y_j = 0$

Update $\delta_j$ and $\eta_j$.

End.

Table 5.9 has an equivalent format to Table 5.1.

Table 5.9 The VPINTR Algorithm: Small Residual

<table>
<thead>
<tr>
<th>$K$</th>
<th>Prec.</th>
<th>Iter</th>
<th>$f$</th>
<th>$|\nabla f|$</th>
<th>flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>no</td>
<td>5</td>
<td>.580</td>
<td>.160e-5</td>
<td>5,307,511</td>
</tr>
<tr>
<td>23</td>
<td>yes</td>
<td>4</td>
<td>.580</td>
<td>.355e-5</td>
<td>1,836,003</td>
</tr>
<tr>
<td>48</td>
<td>no</td>
<td>5</td>
<td>.486</td>
<td>.258e-5</td>
<td>12,966,926</td>
</tr>
<tr>
<td>48</td>
<td>yes</td>
<td>5</td>
<td>.486</td>
<td>.202e-5</td>
<td>5,874,708</td>
</tr>
<tr>
<td>73</td>
<td>no</td>
<td>11</td>
<td>.452</td>
<td>.172e-5</td>
<td>205,858,200</td>
</tr>
<tr>
<td>73</td>
<td>yes</td>
<td>11</td>
<td>.452</td>
<td>.173e-5</td>
<td>27,143,324</td>
</tr>
<tr>
<td>98</td>
<td>no</td>
<td>4*</td>
<td>.380</td>
<td>.196</td>
<td>41,256,185</td>
</tr>
<tr>
<td>98</td>
<td>yes</td>
<td>4*</td>
<td>.380</td>
<td>.195</td>
<td>29,256,597</td>
</tr>
</tbody>
</table>

* = stalled: backtracked too many times

In comparing the percentage savings with our preconditioner we see that the savings are substantial.

Table 5.10 VPINTR: Percentage Savings On Small Residual Problem

<table>
<thead>
<tr>
<th>$K$</th>
<th>% Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>65.8</td>
</tr>
<tr>
<td>48</td>
<td>54.7</td>
</tr>
<tr>
<td>73</td>
<td>80.8</td>
</tr>
<tr>
<td>98</td>
<td>20.1</td>
</tr>
</tbody>
</table>

Our final run is on a zero residual problem with this algorithm. The results are displayed in Table 5.11, followed by the percentage savings.
Table 5.11 The VPINTR Algorithm: Zero Residual

| K | Prec. | Iter | /f     | |||/f|| | flops |
|---|-------|------|--------|--------|--------|-------|
| 23 | no    | 7    | .787e-6| .786e-3| 14,494,170 |
| 23 | yes   | 7    | .221e-6| .801e-3| 3,392,577  |
| 48 | no    | 11*  | .230e-5| .245e-4| 87,760,856 |
| 48 | yes   | 6    | .905e-6| .145e-2| 10,285,700 |
| 73 | no    | 6    | .999e-6| .144e-2| 34,108,444 |
| 73 | yes   | 6    | .980e-6| .147e-2| 21,408,834 |
| 98 | no    | 5    | .532e-6| .924e-3| 36,937,811 |
| 98 | yes   | 5    | .281e-6| .880e-3| 35,873,771 |

* = stalled: backtracked too many times

Table 5.12 VPINTR: Percentage Savings On Zero Residual Problem

<table>
<thead>
<tr>
<th>K</th>
<th>% Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>78.8</td>
</tr>
<tr>
<td>48</td>
<td>88.3</td>
</tr>
<tr>
<td>73</td>
<td>37.1</td>
</tr>
<tr>
<td>98</td>
<td>2.9</td>
</tr>
</tbody>
</table>

5.5. A Brief Summary

The three algorithms seem to converge at their theoretical rates, which is at most q-linear for the small residual problems. Two of the three had some trouble with stalling out. We have no explanation for why this occurred. It is rather astonishing, though, that the Alternation algorithm, whose convergence rate is the slowest, shows up as the most efficient. But we draw no conclusions as we feel that our test results are still at the tentative stage in terms of comparing algorithms. We understand why the VPINTR algorithm should cost about twice as much as the Alternation algorithm per iteration, but not why it takes an inordinate amount of flops overall, even when the number of iterations is much lower than for the other two algorithms.

The trend in our results is that the larger computational savings due to the preconditioner occurs for K between 48 and 73. When K is 98 we have only twice as many rows as columns, and the triangular portion of zeros is large.

Clearly the preconditioner makes an impressive change in the amount of computations required. We do not have an explanation for why the percentage savings should be higher for the VPINTR algorithm but this question can be posed as part of our future investigation.
CHAPTER 6

Concluding Remarks

In this study we mathematically formulate prototypes of linear and nonlinear deconvolution problems. We frequently do not have an exact solution because the recorded data is inconsistent, especially as the measurements are taken in finite precision. So we minimize the squared error of the residual, resulting in either a linear least squares or a nonlinear least squares problem.

When considering computational methods for linear deconvolution problems we choose an iterative method, the preconditioned conjugate gradient algorithm. This method can take advantage of the Toeplitz structure of our matrix by storing only the first column. Most importantly, we develop a preconditioner, based on Cybenko's QR factorization of a circulant matrix, which can significantly speed up the number of iterations and decrease the work required to achieve convergence. Our numerical results confirm that our preconditioner helps the most when the Green's function for an experiment either contains most of the information in the beginning of its time window, or else when it repeats the information in a pattern. The numerical procedure chops off the end of the Green's function to form the preconditioner and hence our idea will work better with the less information we chop off.

In the nonlinear case we examine several nonlinear algorithms. A particular application of ours yields a bilinear problem, which is a paradigm of structure-monitoring experiments. We consider several algorithms which take advantage of the bilinearity.

A problem of scale degeneracy exists for the bilinear formulation. There are infinitely many solutions, all equal to within a scale factor, and we need to be able to find just one solution. We prove two theorems which present two remedies to the ill-conditioning caused by the degeneracy.

We choose three of the nonlinear algorithms. One is tailored for the full problem in which all the unknown variables are solved for simultaneously at each iteration. The other two are tailored for the separable problem in which the unknown variables are divided into two groups in a particular fashion, and solved for separately. A subproblem of all three algorithms is to calculate the solution to a linear least squares problem. Here is the opportunity to put our preconditioner to work and we show that it can amply cut down on the computational load.

Some future work can be spent in investigating these algorithms in more detail. We need to better understand how to help the variable projection algorithm, from a geometric viewpoint.

A logical next step is to investigate the use of our preconditioner in harder nonlinear problems, perhaps where only one set of the variables appears linearly.


Levinson, N. [1947]. The wiener rms error criterion in filter design and prediction. *Journal of Mathematical Physics, 25:* 261-278.


Houston, Texas.


