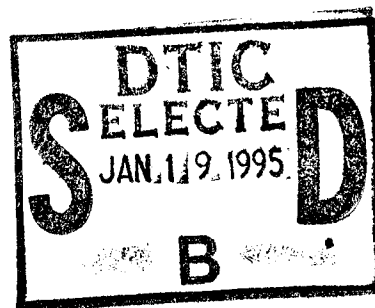


NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

ANALYSIS OF MULTIGRID TECHNIQUES
FOR SYSTEM MODELING WITH
TOEPLITZ APPROXIMATION

by

John S. Volk III

September 1994

Thesis Advisors:

Murali Tummala
Van Emden Henson

Approved for public release; distribution is unlimited

DTIC QUALITY INSPECTED 3

19950117 041

REPORT DOCUMENTATION PAGEForm Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 1994 September	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE ANALYSIS OF MULTIGRID TECHNIQUES FOR SYSTEM MODELING WITH TOEPLITZ APPROXIMATION			5. FUNDING NUMBERS
6. AUTHOR(S) John S. Volk III			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 words) An empirical analysis on the applicability of multigrid techniques to system modeling using system identification techniques is presented. Multigrid with the Toeplitz approximation algorithm is used to model an infinite impulse response (IIR) system with a finite impulse response (FIR) model. Matrix analysis experiments are conducted on the Toeplitz iteration matrix. A comparison of alternative multigrid operators for solving the system modeling problem is presented. The extension of multigrid techniques to system modeling of odd order filters is explored. Computer simulations are developed for analyzing the effectiveness of multigrid techniques in system modeling.			
14. SUBJECT TERMS multigrid techniques; infinite impulse response modeling; Toeplitz approximation algorithm; iterative methods; digital signal processing; system modeling; computer simulations			15. NUMBER OF PAGES 166
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Approved for public release; distribution is unlimited.

Analysis of Multigrid Techniques
for System Modeling with
Toeplitz Approximation

by

John S. Volk III
B.S., California Polytechnic State University -
San Luis Obispo, 1985

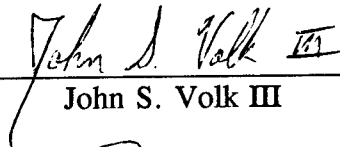
Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

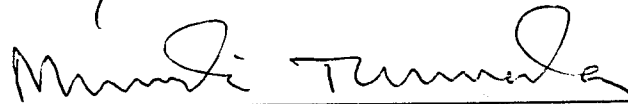
from the

NAVAL POSTGRADUATE SCHOOL
September 1994

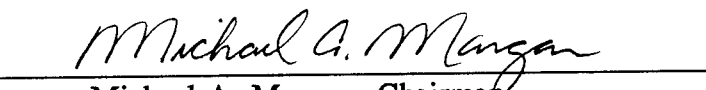
Author:


John S. Volk III

Approved by:


Murali Tummala, Thesis Advisor


Van Emden Henson, Thesis Advisor


Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

An empirical analysis on the applicability of multigrid techniques to system modeling using system identification techniques is presented. Multigrid with the Toeplitz approximation algorithm is used to model an infinite impulse response (IIR) system with a finite impulse response (FIR) model. Matrix analysis experiments are conducted on the Toeplitz iteration matrix. A comparison of alternative multigrid operators for solving the system modeling problem is presented. The extension of multigrid techniques to system modeling of odd order filters is explored. Computer simulations are developed for analyzing the effectiveness of multigrid techniques in system modeling.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist.	Avail and/or Special
A-1	

TABLE OF CONTENTS

I. INTRODUCTION	1
A. SYSTEM MODELING WITH MULTIGRID	1
B. THESIS OUTLINE	3
II. SYSTEM MODELING TECHNIQUES	5
A. THEORY OF SYSTEM MODELING	5
B. SYSTEM IDENTIFICATION	6
III. ANALYSIS OF MULTIGRID COMPONENTS	12
A. THEORY OF MULTIGRID	12
1. Multigrid Model Problem	13
B. RELAXATION METHODS	15
1. Jacobi Method	17
2. Gauss-Seidel Method	20
3. Toeplitz Approximation Algorithm	21
C. COARSE GRID CORRECTION	24

D. INTERGRID TRANSFER OPERATORS	29
1. Restriction Operators	31
2. Prolongation Operators	36
3. Coarse Grid Operator	38
4. Effect of Intergrid Transfer Operators on Error	39
IV. COMPUTER SIMULATIONS	40
A. SIMULATION METHODOLOGY	40
B. TOEPLITZ ITERATION MATRIX ANALYSIS	41
1. Effects of Input Signal Length	41
2. Effects of Random Number Seeds	43
3. Properties of Iteration Matrix Eigenvectors	48
C. TOEPLITZ APPROXIMATION ALGORITHM ANALYSIS	56
1. Convergence to a Zero Solution	56
2. Solving the Weiner-Hopf Equation	75
D. MULTIGRID ANALYSIS	84
1. Nested Multigrid Evaluation	84
2. Applying Multigrid for Even Order FIR Filters	92
3. Applying Multigrid for Odd Order FIR Filters	100

V. CONCLUSION	106
A. SUMMARY OF RESULTS	106
B. RECOMMENDATIONS FOR FUTURE WORK	109
APPENDIX: MATLAB CODE	111
REFERENCES	152
INITIAL DISTRIBUTION LIST	154

LIST OF TABLES

Table 4.1: Frequency Content of Toeplitz Iteration Matrix Eigenvectors	55
Table 4.2: Comparison of Error PSDs for Zero Solution	74
Table 4.3: Comparison of Multigrid Techniques for Modeling 126 th Order Filter	95
Table 4.4: Comparison of Multigrid Techniques for Modeling 127 th Order Filter	101

LIST OF FIGURES

Figure 2.1: System Modeling with System Identification	6
Figure 3.1: Relaxation Method Effects on Local Errors	17
Figure 3.2: Nested Iteration Multigrid	25
Figure 3.3: Fine to Coarse Grid Transfer	27
Figure 3.4: V-cycle Method of Multigrid	28
Figure 3.5: FMV-cycle Method of Multigrid	29
Figure 3.6: Restriction by the Full Weighting Operator for $N=8$	33
Figure 3.7: Restriction by the Injection Operator for $N=8$	34
Figure 3.8: Restriction by the Row Lumping Operator for $N=8$	35
Figure 3.9: Prolongation by the Linear Interpolation Operator for $N=8$	37
Figure 4.1: Maximum Eigenvalue of the Iteration Matrix vs Signal Length ..	43
Figure 4.2: All Eigenvalues of the Iteration Matrix for 8 Signals	45
Figure 4.3: Maximum Eigenvalues vs Signal Length for 12 Seeds	46
Figure 4.4: Average of Maximum Eigenvalues vs Signal Length for 12 Seeds	47
Figure 4.5: Iteration Matrix Eigenvectors for 253 Point Signal	49
Figure 4.6: PSDs of Iteration Matrix Eigenvectors for 253 Point Signal ...	50
Figure 4.7: Iteration Matrix Eigenvectors for 350 Point Signal	51
Figure 4.8: PSDs of Iteration Matrix Eigenvectors for 350 Point Signal ...	52
Figure 4.9: Iteration Matrix Eigenvectors for 1000 Point Signal	53

Figure 4.10: PSDs of Iteration Matrix Eigenvectors for 1000 Point Signal . . .	54
Figure 4.11: Eigenvalues and Initial Guesses for 253 Point Signal	59
Figure 4.12: Zero Solution Error Norms for 253 Point Signal	60
Figure 4.13: PSD of Error for 253 Point Signal, Low Eigenvalue	61
Figure 4.14: PSD of Error for 253 Point Signal, Medium Eigenvalue	62
Figure 4.15: PSD of Error for 253 Point Signal, High Eigenvalue	63
Figure 4.16: Eigenvalues and Initial Guesses for 350 Point Signal	64
Figure 4.17: Zero Solution Error Norms for 350 Point Signal	65
Figure 4.18: PSD of Error for 350 Point Signal, Low Eigenvalue	66
Figure 4.19: PSD of Error for 350 Point Signal, Medium Eigenvalue	67
Figure 4.20: PSD of Error for 350 Point Signal, High Eigenvalue	68
Figure 4.21: Eigenvalues and Initial Guesses for 1000 Point Signal	69
Figure 4.22: Zero Solution Error Norms for 1000 Point Signal	70
Figure 4.23: PSD of Error for 1000 Point Signal, Low Eigenvalue	71
Figure 4.24: PSD of Error for 1000 Point Signal, Medium Eigenvalue	72
Figure 4.25: PSD of Error for 1000 Point Signal, High Eigenvalue	73
Figure 4.26: Weiner-Hopf Solution Error Norms for 253 Point Signal	78
Figure 4.27: Weiner-Hopf Solution PSD of Error for 253 Point Signal	79
Figure 4.28 Weiner-Hopf Solution Error Norms for 350 Point Signal	80
Figure 4.29: Weiner-Hopf Solution PSD of Error for 350 Point Signal	81
Figure 4.30: Weiner-Hopf Solution Error Norms for 1000 Point Signal	82
Figure 4.31: Weiner-Hopf Solution PSD of Error for 1000 Point Signal	83

Figure 4.32: Nested Multigrid Error Norms for 253 Point Signal	86
Figure 4.33: Nested Multigrid PSD of Error for 253 Point Signal	87
Figure 4.34: Nested Multigrid Error Norms for 350 Point Signal	88
Figure 4.35: Nested Multigrid PSD of Error for 350 Point Signal	89
Figure 4.36: Nested Multigrid Error Norms for 1000 Point Signal	90
Figure 4.37: Nested Multigrid PSD of Error for 1000 Point Signal	91
Figure 4.38: 126 th Order Multigrid Solution PSD of Error for 253 Point Signal	96
Figure 4.39: 126 th Order Multigrid Solution PSD of Error for 350 Point Signal	97
Figure 4.40: 126 th Order Multigrid Solution PSD of Error for 1000 Point Signal	99
Figure 4.41: 127 th Order Multigrid Solution PSD of Error for 253 Point Signal .	102
Figure 4.42: 127 th Order Multigrid Solution PSD of Error for 350 Point Signal .	103
Figure 4.43: 127 th Order Multigrid Solution PSD of Error for 1000 Point Signal	105

ACKNOWLEDGEMENT

This thesis is the result of the knowledge and support of several dedicated people. Dr. Jeffery Knorr helped develop a unique curriculum that allowed completion of my graduate course work in one year with all thesis work conducted in absentia. LCDR Gregory Skinner inspired my interest in digital signal processing and introduced me to Dr. Murali Tummala, the originator of my thesis topic. Dr. Tummala and Dr. Van Henson combined their extensive knowledge of multigrid from their respective engineering and mathematics viewpoints to guide me through an in-depth analysis of multigrid in system modeling. Dr. Paul Papayoanou helped conduct research and obtain reference material for the thesis. Mrs. Ann Volk and our children, John IV and Derek, were extremely supportive and understanding throughout my three years of graduate study. I wish to express sincere gratitude to all of you for making this educational experience so rewarding.

I. INTRODUCTION

A. SYSTEM MODELING WITH MULTIGRID

Many technological advances rely on a combination of principles and ideas derived from the engineering and mathematics disciplines. System modeling using multigrid techniques is an example of one endeavor to solve an engineering problem by applying concepts which originated in the mathematics community. The purpose of this thesis is to expand on previous research which experimentally demonstrated that the extension of multigrid techniques to the system modeling problem is viable [Ref. 1].

Multigrid techniques using full weighting and linear interpolation intergrid transfer operators have been applied to solve a linear system modeling problem through system identification. Multigrid techniques were implemented to solve the Weiner-Hopf equation of the system modeling problem with the Toeplitz approximation algorithm. A 22^{nd} order elliptic IIR bandpass filter was modeled with a 126^{th} order FIR filter. Experiments indicated that modeling even order filters (odd coefficient vector length) with multigrid techniques was possible. Multigrid techniques appeared to be best suited for modeling filters that have a coefficient vector of length of 2^m-1 , where m is an integer, $m \geq 1$. [Ref. 1]

Although the results in [Ref. 1] demonstrated that multigrid could be used for system modeling, it is important to examine the practicality of such an application of multigrid. The limited success of multigrid in solving the Weiner-Hopf equation warrants a

theoretical analysis of the application of multigrid to system modeling. Previous work has demonstrated such a theoretical analysis on the application of multigrid to solving certain partial differential equation (PDE) boundary value problems, commonly referred to as the multigrid model problems. The effectiveness of the Jacobi and Gauss-Seidel iteration algorithms when used with multigrid techniques on the model problems has been examined theoretically and shown through experimentation. Experiments to examine the properties of the Jacobi and Gauss-Seidel iteration matrices, including a thorough analysis of the eigenvalues and eigenvectors of the iteration matrices, demonstrate that the Jacobi and Gauss-Seidel algorithms can be successfully applied with multigrid techniques on the model problem. [Ref. 2]

Properties of the Weiner-Hopf equation are much different from those of the multigrid model problem. A numerical solution of the boundary value problem does not require the residual used to measure the error to go to zero. The solution need only be as good as the truncated error that is introduced in the discretization of the PDE through a Taylor series approximation. For the Weiner-Hopf equation, however, a solution is achieved when the residual is zero since an actual system of linear equations is being solved. It is important to note that the original intent of multigrid was to improve the capability to approximate a solution to a PDE through a system of linear equations, not to serve as a general linear system solver.

Despite the differences in nature of the Weiner-Hopf equation and the multigrid model problem, previous work described in [Ref. 1] indicates that multigrid can indeed be applied to system modeling. The research conducted for this thesis expands on the

multigrid work in [Ref. 1] by using empirical experimentation to acquire the same type of information generally obtained through pencil and paper theoretical analysis. While the classical approach to theoretical analysis was useful for the Jacobi and Gauss-Seidel algorithms, the properties of the Toeplitz approximation algorithm make such analysis difficult. Therefore, the Toeplitz approximation algorithm and its use as a multigrid smoother is examined empirically. Matrix analysis experiments are conducted on the Toeplitz iteration matrix. The extension of multigrid techniques to system modeling of odd order filters is explored. As a result, this thesis increases the understanding of the applicability of multigrid with the Toeplitz approximation algorithm to the system modeling problem.

B. THESIS OUTLINE

The intent of this thesis is to thoroughly examine certain aspects of the application of multigrid techniques to system modeling. The first three chapters provide detailed discussions on system modeling and multigrid techniques. Chapter I highlights previous applications of multigrid to the system modeling problem and summarizes the content of the remainder of the thesis. Chapter II discusses system modeling with system identification, including a discussion of direct, iterative, and multigrid methods of system identification. Chapter III presents a theoretical discussion of multigrid with an emphasis on the three major components of multigrid: relaxation methods, coarse grid correction, and intergrid transfer operators. Potential multigrid operators for the system modeling problem are introduced and a multigrid operator for modeling odd order filters is formulated.

Chapter IV details several MATLAB computer simulations that are the heart of this research. Experiments are conducted to analyze the Toeplitz iteration matrix used with multigrid techniques to solve a system modeling problem. Computer simulations running the Toeplitz approximation algorithm are performed, including an application of nested multigrid techniques. Alternative multigrid operators for use in system modeling are explored. These experiments were chosen to provide the type of theoretical analysis that has previously been conducted on the Jacobi and Gauss-Seidel algorithms for use with multigrid on the model problem. Chapter V presents conclusions about the results obtained from the computer simulations and offers recommendations for applying the results to further research on system modeling with multigrid. The appendix contains the MATLAB source code developed for the computer simulations.

II. SYSTEM MODELING TECHNIQUES

A. THEORY OF SYSTEM MODELING

System modeling is an important problem in the area of digital signal processing. System modeling is the process of developing mathematical models in an attempt to explain the behavior of an actual system. The two main goals in developing the mathematical models are accuracy and speed. Optimal accuracy is obtained by minimizing the error between the actual system output and the output from the model. Optimal speed is achieved by minimizing the time required to determine the system model. Computer algorithms are developed to implement the mathematical models with a focus on obtaining an accurate solution as quickly as possible.

One of several methods of system modeling is system identification. System identification is a least squares optimal filtering technique by which a discrete sample of known inputs and outputs of a linear system are used to design a filter that estimates the parameters of the actual system. A classical application of system identification is to use Wiener filtering techniques to develop a linear FIR filter model to estimate the parameters of a system. The following section provides a detailed discussion of how system identification can be used to solve this system modeling problem. [Ref. 3]

B. SYSTEM IDENTIFICATION

System identification is a widely used technique for modeling an unknown linear system with an optimal linear filter. The goal of system identification is to design a linear filter that behaves like the linear system that is being modeled. Thus, for a given input signal, the linear filter model should produce similar output to that of the linear system. To model a linear system with a FIR filter model, discrete samples of a white noise input signal and the corresponding output signal from the actual system are used to estimate the $M+1$ coefficients of the M^{th} order optimal FIR filter (see Figure 2.1).

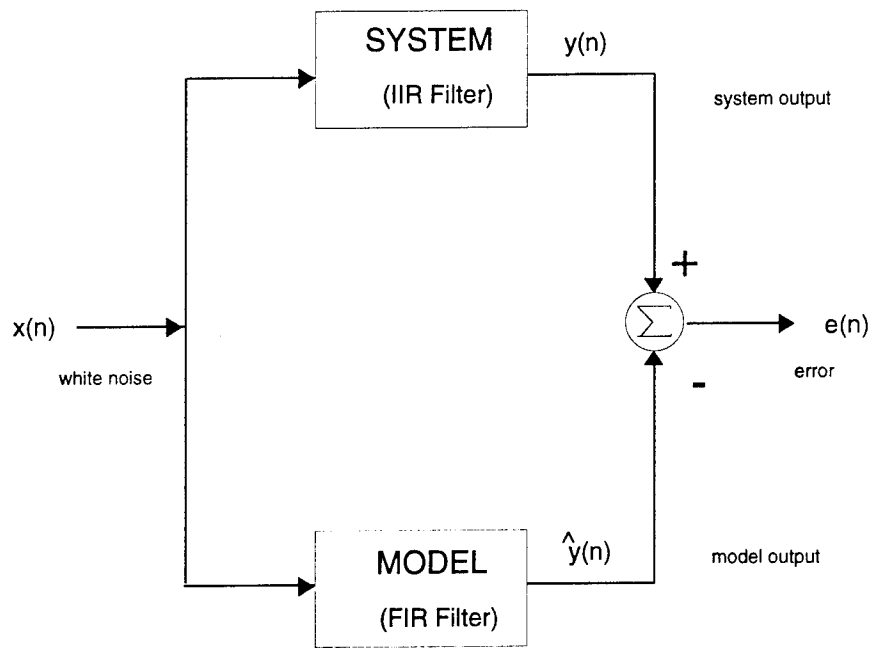


Figure 2.1: System Modeling with System Identification

The output of the M^{th} order FIR model is given by

$$\hat{y}(n) = x(n)b_0 + x(n-1)b_1 + \dots + x(n-M)b_M \quad (2.1)$$

$$= [x(n) \ x(n-1) \ \dots \ x(n-M)] \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_M \end{bmatrix} = \mathbf{x}_n^T \mathbf{b},$$

where $\mathbf{x}_n$ is a vector of input samples at time n and $\mathbf{b}$ is the FIR filter coefficient vector.

The output error $e(n)$ at time n is then determined by

$$e(n) = y(n) - \hat{y}(n) = y(n) - \mathbf{x}_n^T \mathbf{b}, \quad (2.2)$$

where $y(n)$ is the actual system output at time n . Least squares techniques can be used to minimize the sum of squared errors

$$S = \sum_{n=n_I}^{n_F} |e(n)|^2 = \sum_{n=n_I}^{n_F} |y(n) - \mathbf{x}_n^T \mathbf{b}|^2 = \|\mathbf{e}\|^2 \quad (2.3)$$

where n_I and n_F represent an initial and final time point, respectively. Least squares techniques invariably result in a set of linear equations requiring a matrix equation solution. The Wiener-Hopf equation provides one such least squares solution to the system identification problem. [Ref. 3]

The Wiener-Hopf equation is used to determine the optimal FIR filter coefficients $\mathbf{b}$ that will minimize the output error

$$\begin{bmatrix} e(M) \\ e(M+1) \\ \vdots \\ e(N_S-1) \end{bmatrix} = \begin{bmatrix} y(M) \\ y(M+1) \\ \vdots \\ y(N_S-1) \end{bmatrix} - \begin{bmatrix} x(M) & x(M-1) & \dots & x(0) \\ x(M+1) & x(M) & \dots & x(1) \\ \vdots & \vdots & & \vdots \\ x(N_S-1) & x(N_S-2) & \dots & x(N_S-M-1) \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_M \end{bmatrix} \quad (2.4)$$

which is derived from (2.2). In simplified notation, the output error can be expressed as

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{b}. \quad (2.5)$$

Note that the data matrix $\mathbf{X}$ consists of input vectors of the form $\mathbf{x}_n$ as shown in (2.1).

A unique solution of the Wiener-Hopf equation for an M^{th} order FIR filter model requires $N_S \geq 2M+1$ samples of the input signal to form the data matrix and to generate samples of the actual output signal. Minimizing the error vector in (2.4) yields the Wiener-Hopf equation

$$\mathbf{X}^T \mathbf{X} \mathbf{b} = \mathbf{X}^T \mathbf{y}. \quad (2.6)$$

Defining the correlation matrix as $\mathbf{R} = \mathbf{X}^T \mathbf{X}$ and the cross-correlation vector as $\mathbf{r} = \mathbf{X}^T \mathbf{y}$ results in the more familiar notation of the Wiener-Hopf equation

$$\mathbf{R} \mathbf{b} = \mathbf{r} \quad (2.7)$$

which will be used throughout this thesis. [Ref. 3]

It is important to point out that the properties of the correlation matrix are heavily dependent on the nature of the input signal used to form the matrix. The correlation matrix is non-sparse, positive semidefinite, and symmetric, but not necessarily Toeplitz. The correlation matrix can be poorly conditioned depending on the number of input samples available. The correlation matrix does approach a Toeplitz matrix as the number

of samples of the input signal increases. This property will prove to be important in later discussions.

The following example shows how the Wiener-Hopf equation would be formulated for a 2nd order FIR filter model if the input sequence is $\mathbf{x} = [1, -3, 2, 0, 4]$ and the corresponding system output sequence is $\mathbf{y} = [1, -4, 7, -8, 8]$. The first step is to form the data matrix

$$\mathbf{X} = \begin{bmatrix} 2 & -3 & 1 \\ 0 & 2 & -3 \\ 4 & 0 & 2 \end{bmatrix}$$

from the input sequence. The data matrix is then used to form the correlation matrix

$$\mathbf{R} = \mathbf{X}^T \mathbf{X} = \begin{bmatrix} 2 & 0 & 4 \\ -3 & 2 & 0 \\ 1 & -3 & 2 \end{bmatrix} \begin{bmatrix} 2 & -3 & 1 \\ 0 & 2 & -3 \\ 4 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 20 & -6 & 10 \\ -6 & 13 & -9 \\ 10 & -9 & 14 \end{bmatrix}.$$

The data matrix and the system output sequence form the cross-correlation vector

$$\mathbf{r} = \mathbf{X}^T \mathbf{y} = \begin{bmatrix} 2 & 0 & 4 \\ -3 & 2 & 0 \\ 1 & -3 & 2 \end{bmatrix} \begin{bmatrix} 7 \\ -8 \\ 8 \end{bmatrix} = \begin{bmatrix} 46 \\ -37 \\ 47 \end{bmatrix}.$$

The Wiener-Hopf equation for the optimal FIR filter is then

$$\mathbf{R}\mathbf{b} = \mathbf{r} \rightarrow \begin{bmatrix} 20 & -6 & 10 \\ -6 & 13 & -9 \\ 10 & -9 & 14 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 46 \\ -37 \\ 47 \end{bmatrix}.$$

Solution of the Wiener-Hopf equation for this example would determine a coefficient vector of $\mathbf{b} = [1, -1, 2]$ corresponding to the FIR filter given by

$$\hat{y}(n) = x(n) - x(n-1) + 2x(n-2).$$

Various direct, iterative, and multigrid techniques have been proposed to solve the Wiener-Hopf equation for the optimal FIR filter coefficients. The nature of the correlation matrix directs which method is most effective for solving the Wiener-Hopf equation. The classical method of solving the Wiener-Hopf equation for the optimal FIR filter coefficients requires an inversion of the correlation matrix such that

$$\mathbf{b} = \mathbf{R}^{-1}\mathbf{r}. \quad (2.9)$$

Several direct methods are available for inverting the correlation matrix, such as Gaussian elimination, singular value decomposition, and LU decomposition. However, direct matrix inversion is often numerically unstable since direct methods are very dependent on the condition of the correlation matrix. The correlation matrix must be non-singular (full-rank) for an inverse to exist, and a significant input signal sample size is required to increase the likelihood of formulating a non-singular correlation matrix. Even stable direct methods for inverting matrices have their shortcomings. Stable direct methods are computationally inefficient and impractical for large matrices. The cost for inverting an $N \times N$ matrix using direct methods is typically $O(N^3)$. Thus, direct methods are often ineffective for solving the Wiener-Hopf equation. [Ref. 4]

Iterative methods, also known as relaxation techniques, are an alternative to direct methods for determining optimal FIR filter coefficients. Iterative methods generally require much fewer computations per iteration than direct methods. The cost for inverting an $N \times N$ matrix using iterative methods ranges from $O(N)$ to $O(N^2)$ per sweep, depending on how dense is the matrix being inverted [Ref. 4]. Existing iterative methods such as the Jacobi, Gauss-Seidel, and Toeplitz approximation algorithms facilitate solving matrix

equations without matrix inversion. The properties of the iteration matrices used by each of these algorithms determine their suitability for solving certain types of matrix equations. Relaxation algorithms perform optimally when the iteration matrix is well-conditioned and stable [Ref. 2]. For solving the Weiner-Hopf equation, the choice of which iterative method to use is directed by the nature of the correlation matrix. Although iterative methods provide some distinct advantages over direct methods for solving the Weiner-Hopf equation, iterative methods are often slow to converge to a solution and, like direct methods, often require a large sample size of the input signal to be successful [Ref. 1].

Multigrid methods attempt to overcome many of the pitfalls of direct and iterative methods for solving certain types of matrix equations. Originally developed for solving boundary value problems [Ref. 2], multigrid techniques have been applied to solve a variety of partial differential equations, Navier-Stokes equations, and Euler equations [Ref. 5]. While multigrid is not intended to be a general linear system solver, the set of problems to which multigrid can be applied continues to grow. The success of multigrid in solving matrix equations suggests that multigrid techniques may be helpful in solving the Weiner-Hopf equation used in system identification. Multigrid is a relatively new field in mathematics and its usefulness is not universally recognized. The goal of this thesis is to extend the set of problems to which multigrid can be applied. The following chapter discusses the components that give multigrid its power.

III. ANALYSIS OF MULTIGRID COMPONENTS

A. THEORY OF MULTIGRID

Multigrid techniques are a possible alternative to direct methods of matrix inversion for solving a system of linear equations. Multigrid techniques accelerate stand-alone iterative methods for solving a system of linear equations by improving the rate of convergence of the recursive algorithms. In simple terms, multigrid improves the performance of iterative methods by reducing the original matrices to a smaller sample space, performing iterations on a coarser grid, and using the iteration results in the original sample space. Multigrid is based on the intuitively obvious notion that it is cheaper and faster to solve a smaller problem than a large problem (i.e., fewer elements in the matrix means less work for the iterative algorithm). This chapter presents a detailed discussion of how multigrid techniques help reduce the complexity, refine the accuracy, and improve the speed of iterative algorithms.

The success of multigrid is based on a fundamental idea that the different scales of error that result from an iterative method can each be treated on their own grid. In multigrid, local errors are best handled on a fine grid, while global errors can better be dealt with on a coarse grid. Multigrid has two distinct purposes for transferring matrices of a linear system equation to a coarser grid. First, multigrid can use a coarser grid to obtain a better initial guess for the iterative method that is to be performed on the original fine grid. This concept of using coarser grids to obtain better initial guesses is known as

nested iteration. Second, and most importantly, multigrid can use a coarser grid to correct the global approximation of the solution that is produced by the iterative method on the original fine grid. This technique of improving the global approximation of the solution, known as coarse grid correction, is the real strength of multigrid. [Ref. 2]

Multigrid is composed of three major components which must work together to solve a system of linear equations. A given multigrid problem consists of a relaxation method, a coarse grid correction scheme, and intergrid transfer operators. Each of these components of multigrid performs a specific function in determining a solution for a linear system. How well each of the multigrid components work for a particular problem depends on the nature of the matrices involved in the linear equations. Thus, the success of multigrid in solving a system of linear equations is very problem dependent. It has been demonstrated that matrices in certain partial differential equation (PDE) boundary value problems, known as the multigrid model problems, have properties that are well suited for multigrid. The following section describes the one-dimensional case of the multigrid model problem that will be used as a basis for comparison in the remainder of this thesis.

1. Multigrid Model Problem

The multigrid model problem attempts to solve the second order ordinary differential equation given by

$$-u''(x) + \sigma u(x) = f(x), \quad 0 < x < 1, \quad \sigma \geq 0 \quad (3.1)$$

where $u(0)=0$ and $u(1)=0$ are the boundary conditions. The model problem is discretized through a Taylor series expansion to form a system of linear equations for a finite

difference problem. It is important to note that the level of discretization will ultimately determine the minimum error in the solution to the model problem. Partitioning the domain of the model problem into N equal intervals yields $N-1$ linear equations corresponding to $N-1$ second-order finite difference equations that approximate the original differential equation. The N intervals can be thought of as forming a grid with a constant grid spacing of $h=1/N$. Note that the grid spacing referred to in the model problem is analogous to the sampling period for discrete signals in the system modeling problem. Thus, while h in the model problem represents a spacial separation, h in system modeling represents a time separation and is generally replaced with Δt . For the sake of analogy between the model problem and the system modeling problem, the symbol Ω^h will be used to represent the original fine grid on which a solution of the $N-1$ linear equations is desired. The system of linear equations can be represented in matrix form as

$$\frac{1}{h^2} \begin{bmatrix} 2+\sigma h^2 & -1 & & & \\ & -1 & 2+\sigma h^2 & -1 & \\ & & \dots & & \\ & & & \dots & \\ & & & & -1 \\ & & & -1 & 2+\sigma h^2 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N-1} \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{N-1} \end{bmatrix} \quad (3.2)$$

where $\mathbf{r} = [r_1, \dots, r_{N-1}]^T$ approximates the solution to the system of linear equations for $\mathbf{b} = [f(x_1), \dots, f(x_{N-1})]^T$.

For the sake of consistency, the model problem can be expressed in matrix notation in the same manner as the Weiner-Hopf equation, namely

$$\mathbf{Rb} = \mathbf{r}. \quad (3.3)$$

The notation in (3.3) will be repeated for linear systems equations throughout this thesis to draw an analogy between the model problem and the Weiner-Hopf equation of the system modeling problem. Multigrid techniques can be applied to solve the Weiner-Hopf equation by transferring the correlation matrix $\mathbf{R}$ to coarser grids using fine to coarse grid operators and then performing iterations of a recursive technique on the coarse grid to minimize the error or provide a better initial guess for solving the problem on a finer grid. However, the effectiveness of applying multigrid to the system modeling problem is dependent on how the correlation matrix is handled by each of the components of multigrid. The following sections provide a discussion of each of the multigrid components in relation to the system modeling problem and to the multigrid model problem.

B. RELAXATION METHODS

Relaxation (iterative) methods can be used to solve a system of linear equations without applying multigrid techniques. Relaxation methods attempt to solve equations of the form

$$\mathbf{Rb} = \mathbf{r} \quad (3.4)$$

by applying an initial guess $\mathbf{b}_0$ to an iterative algorithm until the error

$$\mathbf{e} = \mathbf{b} - \hat{\mathbf{b}} \quad (3.5)$$

between the approximation $\hat{b}$ and the exact solution b is either eliminated or minimized. Since b is an unknown, iterative algorithms attempt to minimize the residual d in the residual equation

$$d = Re = r - R\hat{b}. \quad (3.6)$$

Since the residual equation relating error to the residual uses the same matrix as the original equation, solving the residual equation is as good as solving the original matrix equation.

While relaxation methods can be used to solve a system of linear equations, iterative algorithms have limitations when applied alone. Many relaxation methods possess a smoothing property which results in rapid reduction of the high frequency (oscillatory) components of the error but has little effect on the low frequency (smooth) components. The high and low frequency components of the error are those in the upper half and lower half of the frequency spectrum, respectively. Figure 3.1 shows how the effect of relaxation methods on local errors eliminates only the high frequency components of the error. The top plot shows the smooth and oscillatory components that are present in the error prior to relaxation. The bottom plot shows the error components after relaxation with most of the oscillatory components of the error eliminated. [Ref. 2]

Since most problems contain errors with both high and low frequency modes, iterative algorithms typically show rapid improvement for the first few iterations and then stall after the high frequency components of the error are eliminated. The limitations caused by the smoothing property can be minimized by applying relaxation methods in conjunction with multigrid. The following sections discuss properties of the Jacobi and

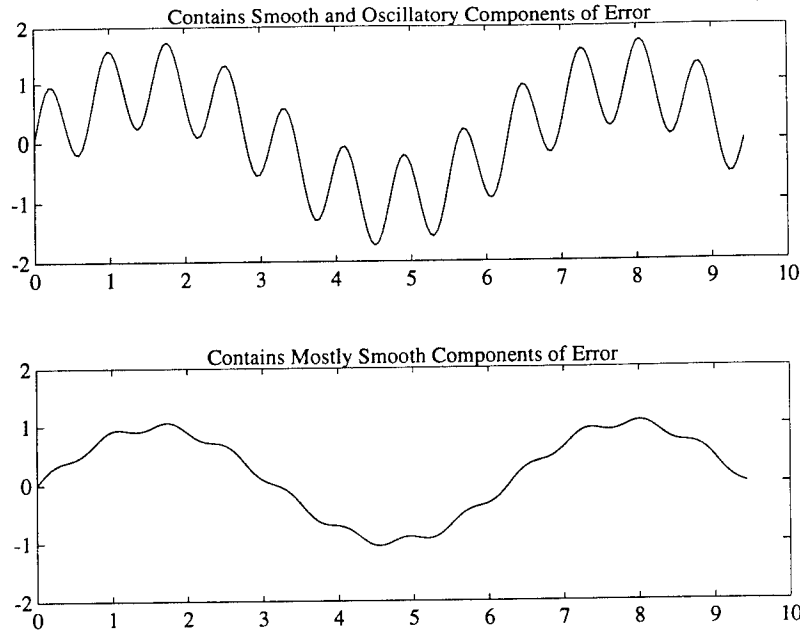


Figure 3.1: Relaxation Method Effects on Local Errors

Gauss-Seidel relaxation methods in relation to the multigrid model problem and introduce the Toeplitz approximation algorithm as a candidate for solving the Wiener-Hopf equation of the system modeling problem.

1. Jacobi Method

The Jacobi relaxation method has been successfully applied with multigrid techniques on the model problem. In terms of any general matrix problem, the Jacobi method solves (3.4) by splitting $\mathbf{R}$ into a diagonal matrix $\mathbf{D}$, a lower triangular matrix $\mathbf{L}$, and an upper triangular matrix $\mathbf{U}$ such that

$$\mathbf{R} = \mathbf{D} - \mathbf{L} - \mathbf{U} \quad (3.7)$$

and

$$(\mathbf{D} - \mathbf{L} - \mathbf{U}) \mathbf{b} = \mathbf{r}. \quad (3.8)$$

The equation may be rewritten as

$$\mathbf{b} = \mathbf{D}^{-1} (\mathbf{L} + \mathbf{U}) \mathbf{b} + \mathbf{D}^{-1} \mathbf{r}. \quad (3.9)$$

The Jacobi method iteration then becomes

$$\mathbf{b}^{(k+1)} = \mathbf{P}_J \mathbf{b}^{(k)} + \mathbf{D}^{-1} \mathbf{r} \quad (3.10)$$

where the Jacobi iteration matrix is

$$\mathbf{P}_J = \mathbf{D}^{-1} (\mathbf{L} + \mathbf{U}). \quad (3.11)$$

The estimate $\mathbf{b}^{(j)}$ becomes $\mathbf{D}^{-1} \mathbf{r}$ for an initial guess of $\mathbf{b}^{(0)}$ set to all zeros. Note that the only matrix inversion involved in the Jacobi method is the simple inversion of the diagonal matrix $\mathbf{D}$, by taking the reciprocal of each element. In practice, the weighted Jacobi method is usually implemented. The weighted Jacobi method uses an average of the previous vector and the current vector computed by the Jacobi iteration. [Ref. 2]

An application of the weighted Jacobi method to the model problem provides a clearer picture of the effects of the Jacobi iteration. Suppose the weighted Jacobi method was applied to the model problem where $f(x)=0$ using single Fourier modes as initial guesses of the solution. Of course, the exact solution is $b=0$, so the current approximation gives the error in this case. The Fourier modes are

$$\mathbf{v}_{k_j} = \sin\left(\frac{jk\pi}{N}\right) \quad (3.12)$$

where $\mathbf{v}_{k_j}$ is the vector whose j^{th} entry is equal to the right-hand side and k is the wave number of the Fourier mode. Using initial guesses consisting of the Fourier modes demonstrates that the error diminishes quickly when the initial guess is a high frequency

(oscillatory) wave (i.e., a higher k value). The weighted Jacobi method converges slowly when the initial guess is a slow frequency (smooth) wave. When the initial guess contains both high and low frequency modes, the weighted Jacobi method efficiently removes the oscillatory components of the error but the smooth components of the error are eliminated much more slowly. Low frequency waves are considered to be those for which the wave number is less than $N/2$ ($0 < k < N/2$). Conversely, high frequency waves have wave numbers greater than $N/2$ ($N/2 < k < N$). [Ref. 2]

It is interesting to note that the eigenvectors of both the Jacobi iteration matrix P_J and the matrix R in the model problem are the Fourier modes discussed above. This observation seems natural since the Fourier modes are the eigenfunctions of the second derivative function. Another important feature of the weighted Jacobi method is that it doesn't mix modes, meaning that a mode will be represented in the next iteration as it was in the previous iteration with only the amplitude of the error being affected. These features are important to the success of the weighted Jacobi method when applied with multigrid techniques. The weighted Jacobi method will effectively eliminate the oscillatory components of the error with a smoothing factor of at least $1/3$ (33% reduction in high frequency error components with each iteration). Another section will discuss how multigrid can be used to make the smooth modes of the error appear to be oscillatory on a coarser grid so that the iterative algorithm can continue to reduce the error in solving the model problem. [Ref. 2]

2. Gauss-Seidel Method

The Gauss-Seidel relaxation method also has been successfully applied with multigrid techniques on the model problem. As with the Jacobi method, the Gauss-Seidel method solves (3.4) by splitting $\mathbf{R}$ into diagonal, lower triangular, and upper triangular matrices to obtain (3.7) and (3.8). However, in the Gauss-Seidel method, the equation is rewritten as

$$\mathbf{b} = (\mathbf{D} - \mathbf{L})^{-1} \mathbf{U} \mathbf{b} + (\mathbf{D} - \mathbf{L})^{-1} \mathbf{r}. \quad (3.13)$$

The Gauss-Seidel method iteration then becomes

$$\mathbf{b}^{(k+1)} = \mathbf{P}_G \mathbf{b}^{(k)} + (\mathbf{D} - \mathbf{L})^{-1} \mathbf{r} \quad (3.14)$$

where the Gauss-Seidel iteration matrix is

$$\mathbf{P}_G = (\mathbf{D} - \mathbf{L})^{-1} \mathbf{U}. \quad (3.15)$$

The estimate $\mathbf{b}^{(1)}$ becomes $(\mathbf{D} - \mathbf{L})^{-1} \mathbf{r}$ for an initial guess of $\mathbf{b}^{(0)}$ set to all zeros.

Unlike the Jacobi method, the eigenvectors of the Gauss-Seidel iteration matrix $\mathbf{P}_G$ are different from those of the model problem matrix $\mathbf{R}$. Also, the Gauss-Seidel iteration mixes modes so that the frequency components of the error may look different from iteration to iteration. Although the Gauss-Seidel method does not share the same properties of the Jacobi method, the Gauss-Seidel method behaves similarly in its elimination of error components. As with the Jacobi method, the Gauss-Seidel method is effective at removing the oscillatory components of the error but has trouble eliminating the smooth components of the error [Ref. 2]. Thus, multigrid can be used to

transfer the smooth component of the error to the upper frequency domain on a coarser grid to aid the iterative algorithm.

3. Toeplitz Approximation Algorithm

While the Jacobi and Gauss-Seidel methods are useful for solving the model problem, the Toeplitz approximation algorithm appears to be ideally suited for solving the Weiner-Hopf equation, due to the near Toeplitz structure of the correlation matrix R . The Toeplitz approximation algorithm solves (3.4) by splitting R into a Toeplitz matrix T (average the diagonal elements of R) and a residual matrix S such that

$$R = T + S \quad (3.16)$$

and

$$(T + S)b = r. \quad (3.17)$$

The equation may be rewritten as

$$Tb = r - (R - T)b, \quad (3.18)$$

by substituting for S and rearranging the terms. Solving for b yields

$$b = T^{-1}(T - R)b + T^{-1}r. \quad (3.19)$$

The Toeplitz approximation algorithm iteration then becomes

$$b^{(k+1)} = P_T b^{(k)} + T^{-1}r \quad (3.20)$$

where the Toeplitz iteration matrix is

$$P_T = T^{-1}(T - R). \quad (3.21)$$

The estimate of $\mathbf{b}^{(1)}$ becomes $\mathbf{T}'\mathbf{r}$ for an initial guess of $\mathbf{b}^{(0)}$ set to all zeros. The inversion of the Toeplitz matrix $\mathbf{T}$ can be obtained by implementing the Levinson recursion.

The Levinson recursion was originally developed to solve normal equations. However, following the same sequence of steps that provide a normal equation solution can also lead to an efficient method for determining the Toeplitz matrix inverse. The Levinson recursion begins with the initial conditions

$$\mathbf{a}_0 = [1] ; \quad \mathbf{r}_0 = R[1] ; \quad \sigma_0^2 = R[0] . \quad (3.22)$$

where $R[0]$ and $R[1]$ are the first and second elements in the top row of an $M+1 \times M+1$ Toeplitz matrix $\mathbf{R}$. The following Levinson recursion is then repeated for $i=1$ to M .

$$\gamma_i = \frac{\mathbf{r}_{(i-1)} \tilde{\mathbf{a}}_{(i-1)}}{\sigma_{(i-1)}^2} \quad (3.23)$$

$$\mathbf{a}_i = \begin{bmatrix} \mathbf{a}_{(i-1)} \\ \text{--} \\ 0 \end{bmatrix} - \gamma_i \begin{bmatrix} 0 \\ \text{--} \\ \tilde{\mathbf{a}}_{(i-1)} \end{bmatrix} \quad (3.24)$$

$$\mathbf{r}_i = [R[1] \quad R[2] \quad \dots \quad R[i+1]] \quad (3.25)$$

$$\sigma_i^2 = (1 - |\gamma_i|^2) \sigma_{(i-1)}^2 \quad (3.26)$$

The notation $\tilde{\mathbf{a}}$ means the reversal of vector $\mathbf{a}$ and a "--" between $\mathbf{a}$ and 0 means concatenate $\mathbf{a}$ and 0 to form a new vector. Completion of the Levinson recursion results in the vectors

$$\mathbf{a}_0 = [1], \quad \mathbf{a}_1 = \begin{bmatrix} 1 \\ a_{11} \end{bmatrix}, \quad \mathbf{a}_2 = \begin{bmatrix} 1 \\ a_{21} \\ a_{22} \end{bmatrix}, \quad \dots, \quad \mathbf{a}_M = \begin{bmatrix} 1 \\ a_{M1} \\ a_{M2} \\ \vdots \\ a_{MM} \end{bmatrix} \quad (3.27)$$

which are used to form the matrix

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ a_{11} & 1 & 0 & 0 & \dots & 0 \\ a_{22} & a_{21} & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \dots & 0 \\ a_{MM} & a_{M(M-1)} & a_{M(M-2)} & a_{M(M-3)} & \dots & 1 \end{bmatrix}. \quad (3.28)$$

The diagonal matrix

$$\mathbf{D} = \begin{bmatrix} \sigma_0^2 & 0 & 0 & 0 & \dots & 0 \\ 0 & \sigma_1^2 & 0 & 0 & \dots & 0 \\ 0 & 0 & \sigma_2^2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & \sigma_M^2 \end{bmatrix} \quad (3.29)$$

is then formed using the σ^2 elements obtained from the Levinson recursion. Note that the diagonal matrix satisfies the equation

$$\mathbf{D} = \mathbf{LRL}^T \quad (3.30)$$

so that the inverse of the Toeplitz matrix is

$$\mathbf{R}^{-1} = \mathbf{L}^T \mathbf{D}^{-1} \mathbf{L}. \quad (3.31)$$

The inversion of the diagonal matrix $\mathbf{D}$ is simply obtained by taking the reciprocal of each element on the main diagonal. [Ref. 3]

Thus, the Levinson recursion can be used in conjunction with the Toeplitz approximation algorithm to solve the Wiener-Hopf equation without resorting to direct matrix inversion. Since the Toeplitz approximation algorithm is known to converge to a solution of the Wiener-Hopf equation faster than most other iterative algorithms, the Toeplitz algorithm is a very useful tool in system modeling [Ref. 1]. Experiments described in Chapter IV will examine the properties of the Toeplitz iteration matrix to determine if application of multigrid with the Toeplitz approximation algorithm can further increase the utility of the algorithm.

C. COARSE GRID CORRECTION

The key to multigrid is the ability to further reduce the error in solving a system of linear equations by performing iterations on a coarser grid once relaxation methods begin to stall in the original problem domain. As mentioned previously, there are two reasons for performing iterations on a coarse grid: 1) to obtain a better initial guess for the relaxation method on a finer grid and 2) to correct the global approximation of the solution produced on a finer grid. The two techniques for handling these problems, nested iteration and coarse grid correction, will be discussed in more detail in this section.

Nested iteration involves solving the system on a coarse grid to obtain a better initial guess for a relaxation method on the next finer grid. Obtaining a good initial guess for the relaxation method will result in less work to solve the problem by reducing the number of iterations required to obtain a solution. Using a coarser grid to get the initial guess is relatively fast and inexpensive since the coarse grid contains fewer points than the fine grid. Nested iteration involves reducing the matrices in a problem from an initial

grid Ω^h (recall that h is the spacing between grids) to a very coarse grid through successive intergrid transfers. The chosen relaxation method is then implemented on the coarse grid until a solution is obtained. The solution is transferred back to the next finer grid and used as the initial guess for the relaxation method on the finer grid. The process of obtaining a solution on a coarse grid and using the solution as the initial guess on the next finer grid is repeated until the problem is solved on the finest grid. Note that there is limited work being done to telescope down from the finest grid to the coarsest grid. The majority of the work is done in recursively obtaining initial guesses for each successive fine grid. A pictorial representation of this process for an initial grid Ω^h and a coarsest grid Ω^{16h} is shown in Figure 3.2. [Ref. 2]

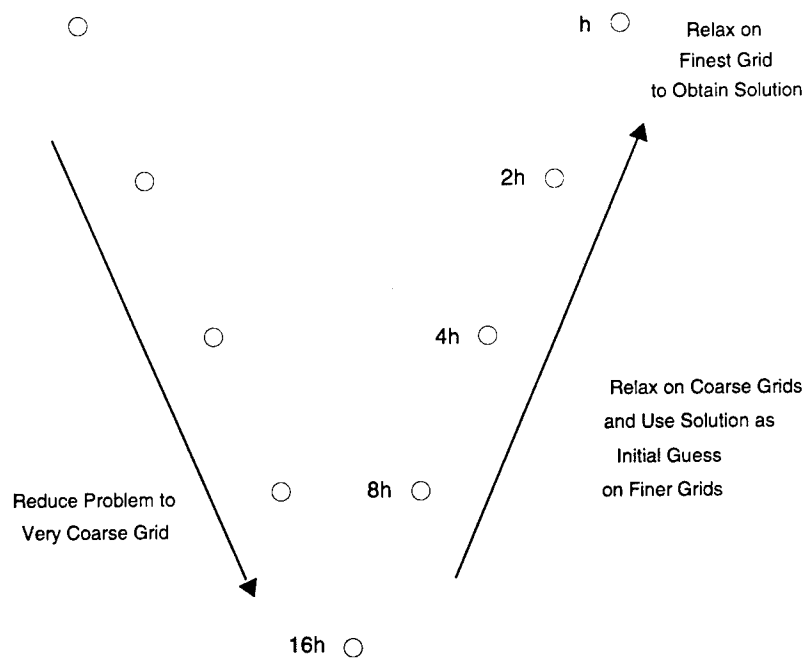


Figure 3.2: Nested Iteration Multigrid

A more powerful use of coarse grids involves coarse grid correction. Coarse grid correction is the process by which the residual equation is used to relax on the error. The

chosen relaxation method is executed on the initial grid Ω^h for a few iterations until convergence stalls. The residual equation is then transferred to the next coarser grid Ω^{2h} and solved using coarse grid correction to determine the error on the coarse grid. The error from the coarse grid is then transferred back to the original fine grid Ω^h and used to correct the solution obtained on the fine grid. [Ref. 2]

There are three major reasons why coarse grid correction makes multigrid so effective on the multigrid model problem. First, as with nested iteration, it is faster and less expensive to perform iterations on a coarse grid since the coarse grid has fewer points than the next finer grid. Second, the error on a fine grid is smooth after relaxation and can accurately be represented on a coarse grid. Third, the smooth mode looks more oscillatory on a coarse grid which increases the effectiveness of the relaxation method being implemented. For the multigrid model problem, it is known that relaxation methods are much more effective at reducing high frequency components than low frequency components of the error. Therefore, the fine to coarse grid transfer for the model problem makes the smooth components on the finer grid look like oscillatory components on a coarser grid as shown in Figure 3.3. The figure may mislead one to believe that the two errors are equally oscillatory (four peaks in each wave), but the fine grid has four peaks in 12 points while the coarse grid has four peaks in only 6 points. In other words, the signal on the fine grid contains 2 cycles out of a maximum 6 cycles that that may be resolved with 12 points (signal is 1/3 of the way up the spectrum) while the signal on the coarse grid contains 2 cycles out of a possible 3 cycles that may be resolved with 6 points (signal is 2/3 of the way up the spectrum). Hence, the error on the coarse grid is

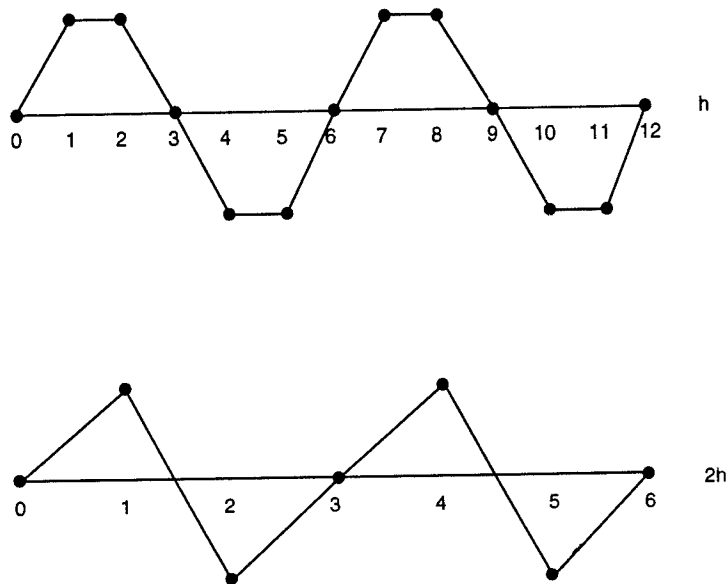


Figure 3.3: Fine to Coarse Grid Transfer

indeed more oscillatory than the error on the fine grid. Thus, to determine if nested iteration or coarse grid correction will be beneficial to solving a particular problem, it is important to understand how the problem is transferred to a coarser grid.

Various multigrid cycling schemes, such as the V-cycle and the full multigrid V-cycle (FMV-cycle), are available for transferring matrices between grids. The V-cycle method uses the coarse grid correction scheme in a recursive fashion to solve a system of linear equations. Figure 3.4 shows how the V-cycle eliminates the error in the upper half of the frequency spectrum at each successive coarse grid. A correction to the solution at each successive finer grid is made using the error approximation obtained on the next coarser grid until the algorithm solves the problem on the finest grid. The V-cycle method is effective on the model problem because the smooth components of the

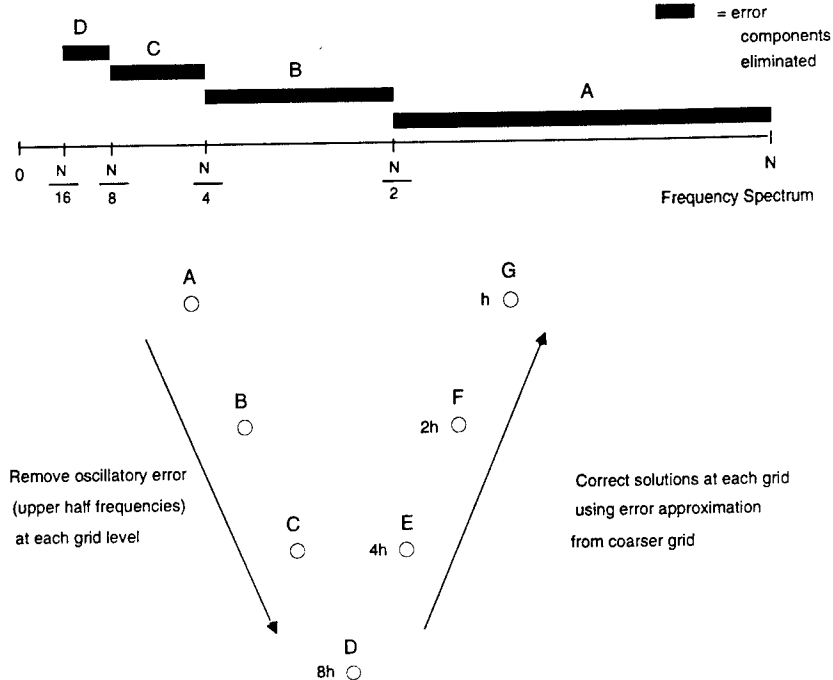


Figure 3.4: V-cycle Method of Multigrid

error are accurately represented as high frequency components on each coarser grid and the oscillatory components of the error are eliminated at each grid level by the relaxation method. Transfer of the error approximation back to finer grids works well because the smooth error on the coarser grid can easily be interpolated back to the finer grid. If high frequency components are present in the error approximation then the oscillatory components will be excited on the finer grid. However, these oscillatory components will again be removed by the relaxation method.

The FMV-cycle combines the V-cycle with nested iteration techniques to improve the initial guess at each grid level. Successive V-cycles are performed as shown in Figure 3.5 until a good initial guess can be made on the finest grid. The FMV-cycle then proceeds with the V-cycle as described above until a solution to the original problem is

Improved Initial Guesses
for V-Cycle

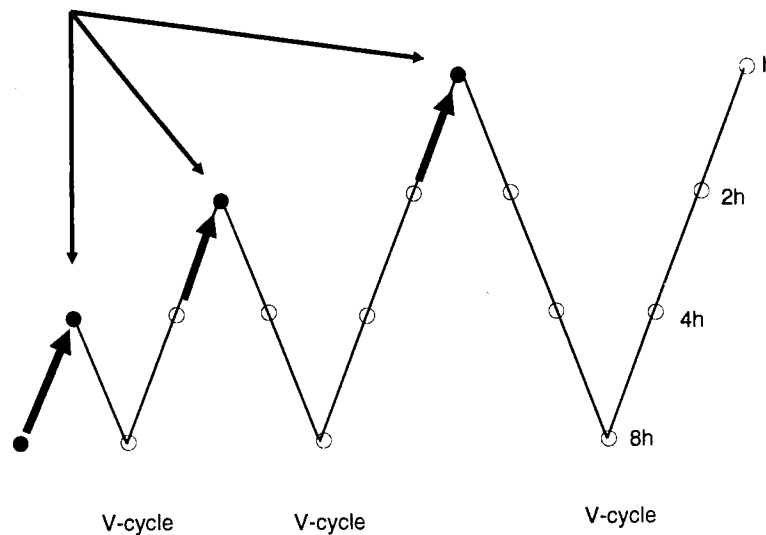


Figure 3.5: FMV-cycle Method of Multigrid

obtained. The FMV-cycle integrates all the components of multigrid into a successful algorithm which overcomes the limitations of each of the multigrid components when taken alone. The following section discusses the intergrid transfer operators which must work with the relaxation method, coarse grid correction scheme, and nested iteration to make multigrid successful.

D. INTERGRID TRANSFER OPERATORS

A difficult challenge when using multigrid techniques is determining how to represent the original problem on a coarse grid. Multigrid requires some mechanism for transferring vectors and matrices between grids. Therefore, *intergrid transfer* operators are used in multigrid to transfer vectors in a system of linear equations between grids. *Restriction* operators are used to transfer matrices from fine to coarse grids. *Prolongation*

operators are used to transfer matrices from coarse grids back to fine grids. Although operators can be developed to handle any grid spacing, most multigrid operators are designed so that each coarse grid has a grid spacing of two times that of the next finer grid.

Restriction operators of the form I_h^{2h} transfer vectors from $\mathfrak{R}^{N-1}$ (fine grid space) to $\mathfrak{R}^{N/2-1}$ (coarse grid space) for odd length vectors (even order filters) or from $\mathfrak{R}^N$ to $\mathfrak{R}^{N/2}$ for even length vectors (odd order filters). Thus, N will always be an even number. Restriction operators are responsible for making the smooth error on a finer grid look oscillatory on the next coarser grid. Prolongation operators of the form I_{2h}^h transfer vectors from $\mathfrak{R}^{N/2-1}$ (coarse grid space) to $\mathfrak{R}^{N-1}$ (fine grid space) or from $\mathfrak{R}^{N/2}$ to $\mathfrak{R}^N$. Prolongation operators are most effective when the error on the coarse grid is smooth, so the error can be successfully transferred to the next finer grid. While transferring vectors from one grid to another requires a single intergrid transfer operator, transfer of the coarse grid operator matrix requires a combination of restriction and prolongation operators for a grid transfer. [Ref. 2]

Intergrid transfer operators are used with relaxation methods and coarse grid correction schemes to make multigrid work on the model problem. The relaxation scheme removes the oscillatory component of the error on a fine grid. The restriction operator transfers the low frequency error on the fine grid to a high frequency error on a coarser grid. The coarse grid correction approximates the error on the finer grid through relaxation of the residual equation on the coarse grid. The prolongation operator then transfers the smooth error correction back to the finer grid. The following sections

discuss some potential intergrid transfer operators for multigrid. Full weighting, injection, and lumping operators can be used for fine to coarse grid transfers, and the linear interpolation operator is the predominant operator for coarse to fine grid transfers. A coarse grid operator matrix can be transferred between grids using a combination of the intergrid transfer operators.

1. Restriction Operators

Fine to coarse grid (restriction) operators are used to transfer vectors from a large sample space to a smaller sample space. The choice of which restriction operator to use depends on the type of problem being solved. Full weighting, injection, and row lumping are three potential restriction operators which will be discussed in this section. The full weighting operator has proven to be effective on the multigrid model problem [Ref. 2]. Also, the full weighting operator has been successfully applied to solving the Weiner-Hopf equation for even order FIR filters [Ref. 1]. The injection operator is another potential fine to coarse grid operator for transferring vectors for even order filters. The lumping operator, which works for both even and odd vectors, can potentially be used to extend multigrid techniques to solving the Weiner-Hopf equation for odd order FIR filters.

The full weighting operator transfers vectors from $\mathfrak{R}^{N-1}$ to $\mathfrak{R}^{N/2-1}$ (for the sake of analogy, since N is even for the multigrid model problem, the assumption that N is even will be made for extension to the Weiner-Hopf equation). This linear operator weights three neighboring points on a fine grid to form a single point on a coarser grid. For the case of $N=4$, the full weighting operator is

$$\mathbf{I}_h^{2h} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix} = \frac{1}{4} [1 \ 2 \ 1]. \quad (3.32)$$

Thus, as shown in Figure 3.6 for the case of $N=8$, the full weighting (full $\rightarrow$ values sum to 1, weighting $\rightarrow$ uses a weighted average) operator creates a coarse grid point by adding $1/2$ the value of its corresponding fine grid point to $1/4$ the value of the two neighboring fine grid points. The following example shows how the full weighting operator would transfer the vector shown in Figure 3.6 from a fine grid to a coarse grid.

$$\mathbf{I}_h^{2h} \mathbf{a}^h = \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 2 \\ 1 \\ 4 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} = \mathbf{a}^{2h}.$$

When applied to the system modeling problem, the full weighting operator requires an even order filter and is most effective when the filter order is $2^m - 1$, where m is an integer, $m \geq 1$ [Ref. 1].

Another possible restriction operator for multigrid is the injection operator. The injection operator transfers vectors from $\mathfrak{R}^{N-1}$ to $\mathfrak{R}^{N/2-1}$ by simply eliminating every other point from the fine grid to create the next coarser grid. For the case of $N=4$, the linear injection operator is

$$\mathbf{I}_h^{2h} = [0 \ 1 \ 0]. \quad (3.33)$$

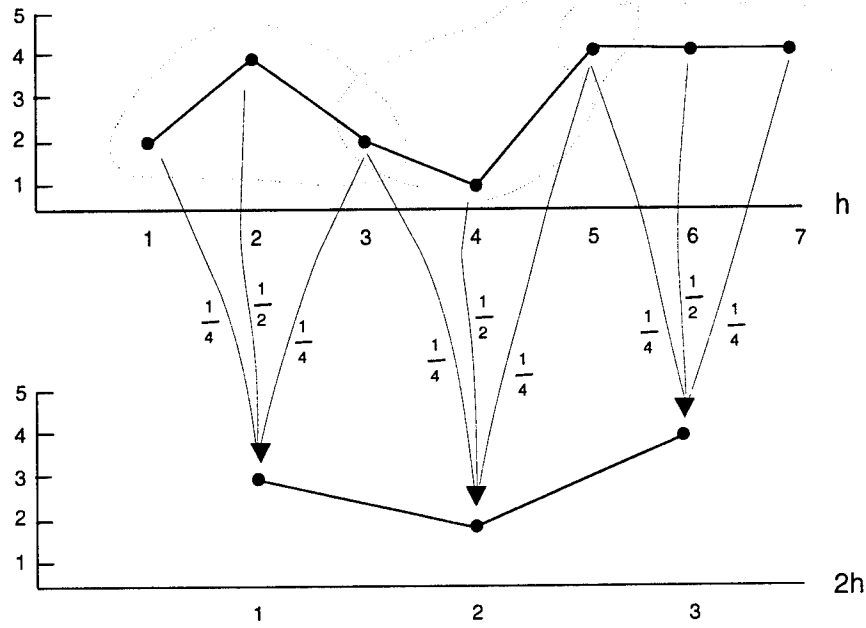


Figure 3.6: Restriction by the Full Weighting Operator for $N=8$

Figure 3.7 shows an application of the injection operator for the case of $N=8$. The following example shows how the injection operator would transfer the vector shown in Figure 3.7 from a fine grid to a coarse grid.

$$\mathbf{I}_h^{2h} \mathbf{a}^h = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 2 \\ 1 \\ 4 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 4 \end{bmatrix} = \mathbf{a}^{2h}.$$

The injection operator may have some application to the system modeling problem for even order filters.

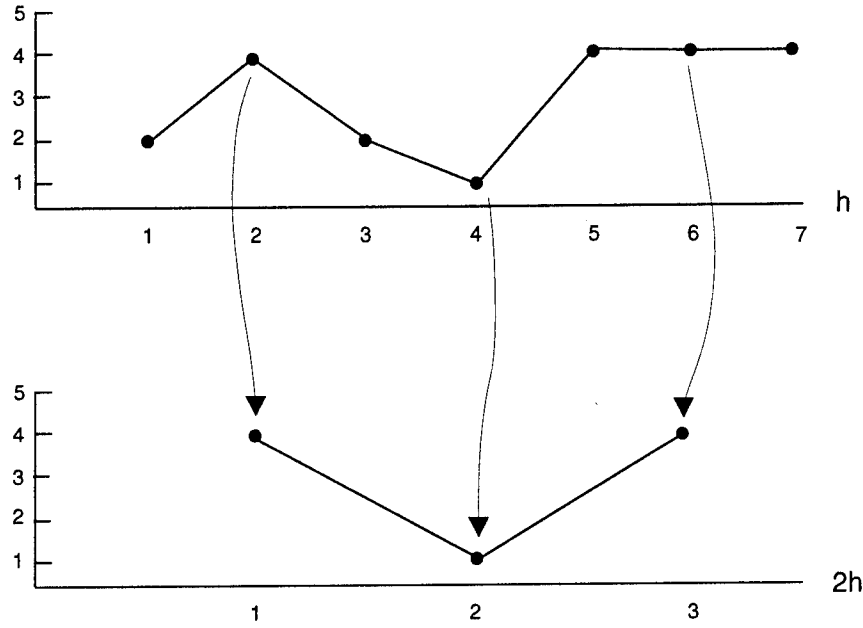


Figure 3.7: Restriction by the Injection Operator for $N=8$

Previous research showed an application of multigrid to even order filters only. The row lumping operator is a linear restriction operator which may be applied to solving the Weiner-Hopf equation for odd order filters. The row lumping operator transfers vectors from $\mathfrak{R}^N$ to $\mathfrak{R}^{N/2}$ by combining two points on a fine grid to form a single point on the next coarser grid. For the case of $N=4$, the row lumping operator is

$$\mathbf{I}_h^{2h} = \begin{bmatrix} 1 & 1 \end{bmatrix} \quad (3.34)$$

Figure 3.8 shows an application of the row lumping operator for the case of $N=8$. The following example shows how the row lumping operator would transfer the vector shown in Figure 3.8 from a fine grid to a coarse grid.

$$I_h^{2h} a^h = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 2 \\ 1 \\ 4 \\ 4 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \\ 8 \\ 7 \end{bmatrix} = a^{2h}.$$

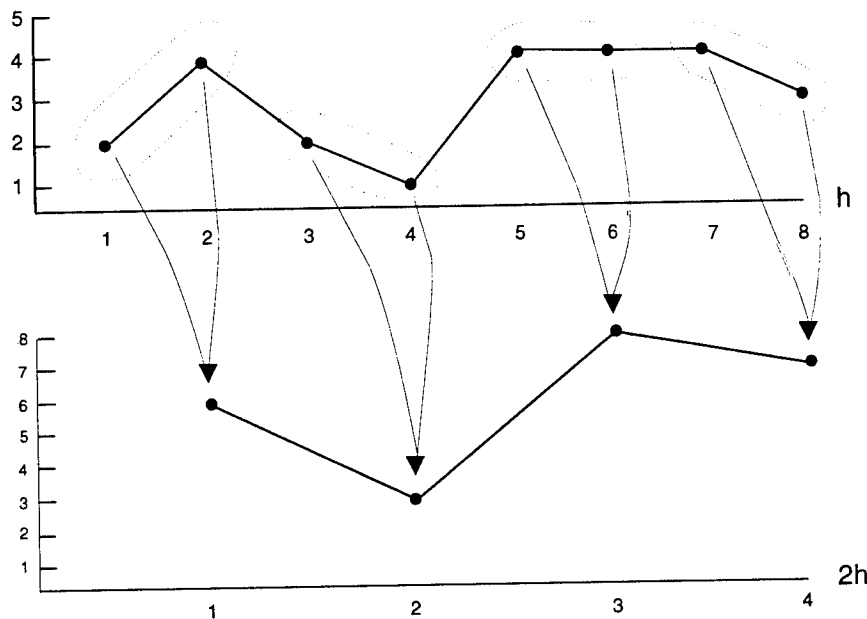


Figure 3.8: Restriction by the Row Lumping Operator for $N=8$

Restriction operators work best when the initial fine grid is on the order of 2^m or 2^m-1 , since these grids can easily be reduced to the coarsest grid of a single point. However, hybrid techniques can be used to transfer vectors to coarser grids when $2^m \times 2^m$ or $2^m-1 \times 2^m-1$ grids are not available. These hybrid techniques use a mix of restriction operators, such as the full weighting and row lumping operators, where the full weighting operator acts on odd grids and the row lumping operator acts on even grids.

This research focuses on just a few of the countless possibilities of restriction operators that can be designed for multigrid applications.

2. Prolongation Operators

Coarse to fine grid (prolongation) operators can be used to transfer a vector from a small sample space back to a larger sample space. The most common operator for this purpose is the linear interpolation operator. The linear interpolation operator has been successfully applied with the full weighting restriction operator to help solve both the multigrid model problem [Ref. 2] and the Weiner-Hopf equation of the system modeling problem for even order FIR filters [Ref. 1].

The linear interpolation operator transfers vectors from $\mathfrak{R}^{N/2-1}$ to $\mathfrak{R}^{N-1}$. Neighboring points on a coarse grid are averaged to calculate points on the next finer grid. The linear interpolation operator assumes a straight line between two points on a coarse grid to approximate a new point between the two points on the next finer grid. For the case of $N=4$, the linear interpolation operator is

$$\mathbf{I}_{2h}^h = \begin{bmatrix} \frac{1}{2} \\ 1 \\ \frac{1}{2} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}. \quad (3.35)$$

Note that the linear interpolation operator is the transpose of the full weighting restriction operator times a constant. Therefore, the linear interpolation and full weighting operators complement each other quite well. Figure 3.9 shows an application of the linear interpolation operator for the case of $N=8$. The following example shows how the linear

interpolation operator would transfer the vector shown in Figure 3.9 from a coarse grid to a fine grid.

$$\mathbf{I}_{2h}^h \mathbf{a}^{2h} = \frac{1}{2} \begin{bmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 3 \\ 2.5 \\ 2 \\ 3 \\ 4 \\ 2 \end{bmatrix} = \mathbf{a}^h.$$

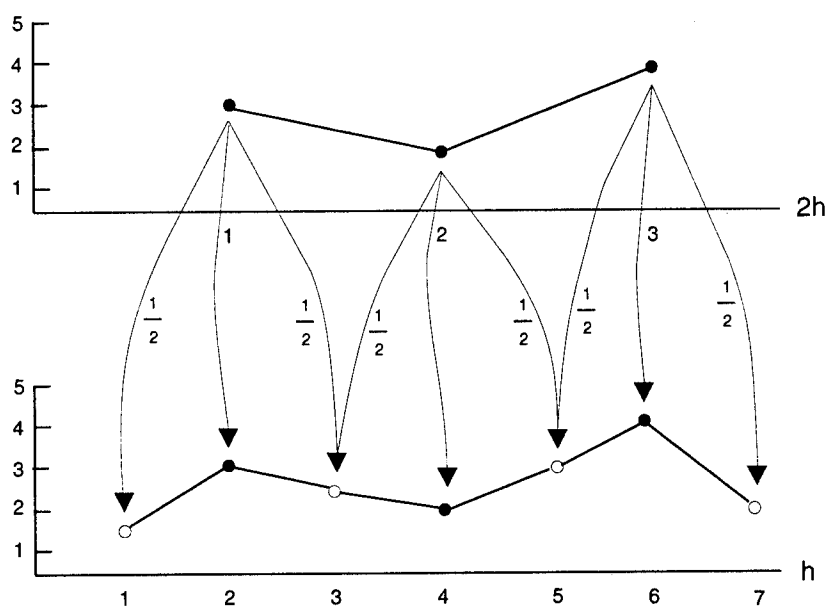


Figure 3.9: Prolongation by the Linear Interpolation Operator for $N=8$

While the linear interpolation operator is the most common prolongation operator used in multigrid, it is possible to develop other prolongation operators for transferring vectors from coarse to fine grids. Quadratic and cubic interpolation operators are possible prolongation operators that may be useful for certain types of problems. The decision as

to which prolongation operator to use is dependent on the nature of the problem and the choice of restriction operators.

3. Coarse Grid Operator

In addition to requiring a mechanism for transferring vectors between grids, multigrid requires the capability to represent the fine grid operator matrix on a coarse grid. Multigrid formulates this coarse grid operator matrix through a combination of the restriction and prolongation intergrid transfer operators, to transfer the operator matrix from $\mathcal{R}^{N-1}$ to $\mathcal{R}^{N/2-1}$. The coarse grid operator matrix is defined by

$$\mathbf{R}^{2h} = \mathbf{I}_h^{2h} \mathbf{R}^h \mathbf{I}_{2h}^h. \quad (3.36)$$

The following example shows the transfer of an fine grid operator matrix $\mathbf{R}^h$ to a coarser grid for the case of $N=8$, using the full weighting restriction operator and the linear interpolation prolongation operator.

$$\begin{aligned} \mathbf{I}_h^{2h} \mathbf{R}^h \mathbf{I}_{2h}^h &= \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} 11 & -4 & -8 & 9 & 6 & -10 & 3 \\ -4 & 10 & -2 & -7 & 7 & 9 & -9 \\ -8 & -2 & 12 & -2 & -6 & 6 & 4 \\ 9 & -7 & -2 & 12 & -3 & -7 & 6 \\ 6 & 7 & -6 & -3 & 11 & -1 & -6 \\ -10 & 9 & 6 & -7 & -1 & 10 & -3 \\ 3 & -9 & 4 & 6 & -6 & -3 & 10 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \\ &\rightarrow \mathbf{I}_h^{2h} \mathbf{R}^h \mathbf{I}_{2h}^h = \begin{bmatrix} 2\frac{7}{8} & 0 & 3\frac{7}{8} \\ 0 & 4\frac{7}{8} & -1\frac{1}{8} \\ 3\frac{7}{8} & -1\frac{1}{8} & 4\frac{1}{8} \end{bmatrix} = \mathbf{R}^{2h}. \end{aligned}$$

4. Effect of Intergrid Transfer Operators on Error

When exploring potential intergrid transfer operators for use with multigrid, it is important to consider the effect that the operators will have on the various error components of the problem. As an example, consider the effects of the full weighting and linear interpolation operators on the errors in the multigrid model problem. As previously mentioned, relaxation methods typically possess a smoothing property which makes them effective at eliminating oscillatory modes of the error, but ineffective at removing smooth modes. Therefore, the model problem exploits the fact that restriction operators make the smooth components of the error on the fine grid look oscillatory on the next coarser grid. The full weighting operator performs this function quite well for the model problem. The full weighting operator also aliases the remaining high frequency errors on the fine grid to low frequency modes on the coarse grid, but with small amplitudes.

The linear interpolation operator is most effective when the coarse grid contains mainly smooth error components. Otherwise, the linear interpolation operator will excite the oscillatory error on the next finer grid. Since the full weighting operator and relaxation on the coarse grid reduces the error to only smooth modes for the model problem, the linear interpolation operator is effective for transferring the errors back to the fine grid. It is clear that the restriction and prolongation operators must complement each other as well as the chosen relaxation method. The following chapter will explore some experimental investigations into how well the components of multigrid relate to the system modeling problem.

IV. COMPUTER SIMULATIONS

A. SIMULATION METHODOLOGY

Several computer simulations were developed to conduct various experiments for this research. The simulations were run on a UNIX based SUN SPARCstation on the Naval Postgraduate School ECE Department network. The computer algorithms were written using MATLAB, a high-level programming language designed for scientific applications requiring extensive matrix calculations and detailed graphic presentations. Modular procedures were developed so that the procedures could be used interchangeably to conduct different experiments (see the appendix for code). The MATLAB normal random number generator function 'randn' was used to create Gaussian white noise input signals for the various computer simulations. Multiple seeds were applied to the 'randn' function to increase the randomness of the signals.

The computer simulations were intended to investigate various aspects of multigrid in system modeling. Several experiments were conducted to analyze the suitability of the Toeplitz iteration matrix for use with multigrid. Analysis of the Toeplitz approximation algorithm was performed, including experiments with nested iteration multigrid to explore possible techniques for obtaining better initial guesses for multigrid in system modeling. The effectiveness of the Toeplitz approximation algorithm alone was compared to its effectiveness when used with multigrid on the system modeling problem. Potential multigrid operators for both even and odd order filters were compared through

simulations. The following sections discuss the purpose, procedure, and results of each of the computer simulation experiments.

B. TOEPLITZ ITERATION MATRIX ANALYSIS

The Toeplitz approximation algorithm has been proposed as an iterative technique for multigrid to solve the Weiner-Hopf equation for an optimal FIR filter. Analysis of the Toeplitz iteration matrix

$$\mathbf{P}_T = \mathbf{T}^{-1} (\mathbf{T} - \mathbf{R}) \quad (4.1)$$

was conducted to examine the practicality of applying the Toeplitz approximation algorithm to the Weiner-Hopf equation. Several experiments were conducted through computer simulations to analyze the properties of the eigenvalues and corresponding eigenvectors of the Toeplitz iteration matrix. Experiments were performed to investigate the effects of the input signal length and different random number seeds on the formation of the Toeplitz iteration matrix and to examine the spectral content of the eigenvectors of the iteration matrix. The experiments were designed to determine if the properties of the Toeplitz iteration matrix can be exploited by multigrid techniques.

1. Effects of Input Signal Length

An important factor in determining if an iterative method is likely to converge to a solution of a system of linear equations is the condition of the iteration matrix used by the algorithm. Examination of the eigenvalues of the iteration matrix can provide insight about the condition of the iteration matrix. To ensure convergence of the

algorithm, the maximum eigenvalue of the iteration matrix must be less than 1.0 . For the Toeplitz approximation algorithm, the condition of the Toeplitz iteration matrix is dependent on the size of the input signal sequence used to formulate the iteration matrix. Therefore, an experiment was conducted to determine the effect of input signal length on the condition of the Toeplitz iteration matrix.

The experiment involved the generation of a 127×127 Toeplitz iteration matrix for modeling a 126^{th} order FIR filter. The iteration matrix was formed for 748 different white noise input signals, ranging from 253 points (the minimum length input signal) through 1000 points. The input signals were separately formulated using a random number generator seed of 1 . The maximum eigenvalue of the iteration matrix was then calculated for each input signal. Figure 4.1 shows a plot of the maximum eigenvalues versus input signal length. Note that the plot is clipped so that maximum eigenvalues greater than 1.0 do not appear on the plot.

Ideally, the maximum eigenvalue of the iteration matrix must be less than 1.0 to ensure convergence of the Toeplitz iteration algorithm. The plot shows this condition to be met for all input signals consisting of 448 or more points. Thus, the Toeplitz approximation algorithm would be expected to converge for input signals of lengths greater than four times the filter order or 504 points for a 126^{th} order FIR filter. Also, 95 percent of the signals between 378 points (three times the filter order) and 504 point (four times the filter order) have maximum eigenvalues greater than 1.0 .

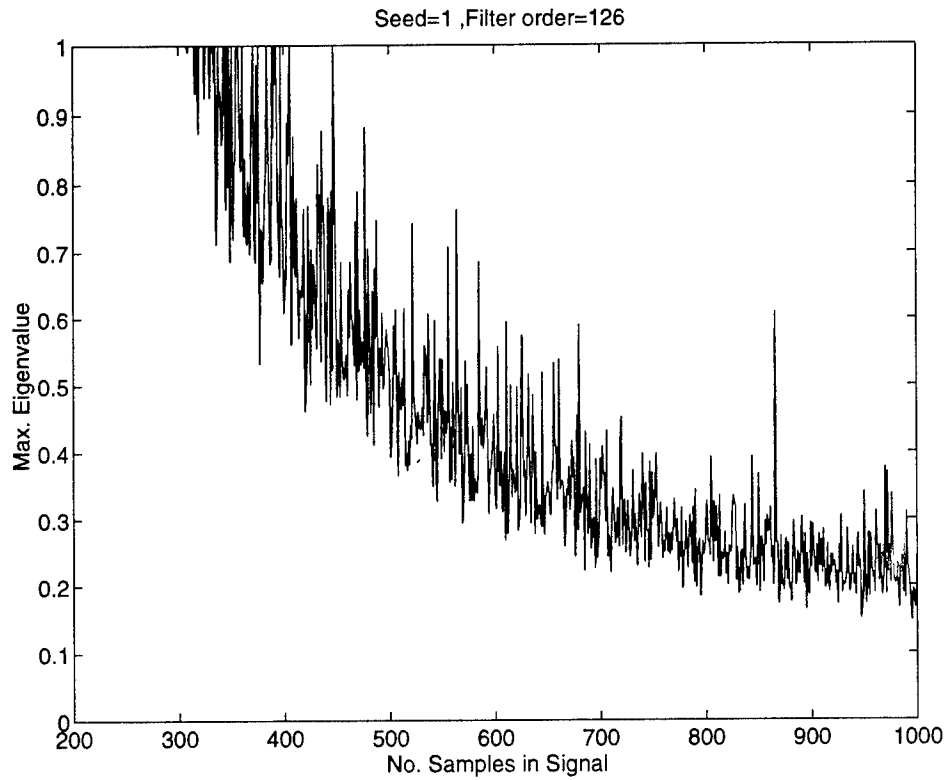


Figure 4.1: Maximum Eigenvalue of the Iteration Matrix vs Signal Length

2. Effects of Random Number Seeds

The randomness of the input signals used in the various experiments is dependent on the effectiveness of the random number generator used to create the signals. The 'randn' function in MATLAB provides a sequence of pseudo-random numbers with a random normal distribution, which can be used to simulate a Gaussian white noise input signal. The user can select a seed for the random number generator to effect the creation of the sequence. The same sequence will be generated by the 'randn' function each time a MATLAB program uses the same seed so that the results can easily be repeated. Since the selected seed effects the randomness of the input signal, the seed will also effect the

condition of the Toeplitz iteration matrix. Thus, an experiment was conducted to determine if the seed selected for the random number generator has a significant effect on the properties of the Toeplitz iteration matrix.

The experiment involved the generation of eight different input signals using a selected random number generator seed. The input signals had lengths of 253, 350, 450, 550, 650, 750, 850, and 1000 points, respectively. Each of the eight input signals were used to form a 127×127 Toeplitz iteration matrix required to determine the coefficients of a 126^{th} order FIR filter. The maximum eigenvalue of the iteration matrix was then calculated for each input signal. Figure 4.2 shows the results for input signals created with a random number generator seed of 1. The figure contains plots of all the eigenvalues of the iteration matrix for each of the eight signals and a plot of the maximum eigenvalue of the iteration matrix versus input signal length. This process was repeated for each of 12 different random number generator seeds (1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, and 31). The plot of maximum eigenvalue of the iteration matrix versus input signal length was created for each of the 12 seeds as shown in Figure 4.3. Note that eigenvalues greater than 1.0 are clipped on the plots.

The plots in Figure 4.3 show that the minimum length input signal (253 points) resulted in a Toeplitz iteration matrix with a maximum eigenvalue greater than 1.0 for each of the 12 test cases. The result indicates that the Toeplitz approximation algorithm is not likely to converge to a solution of the Weiner-Hopf equation for the minimum input signal length case. This result is not very surprising, since the minimum length input

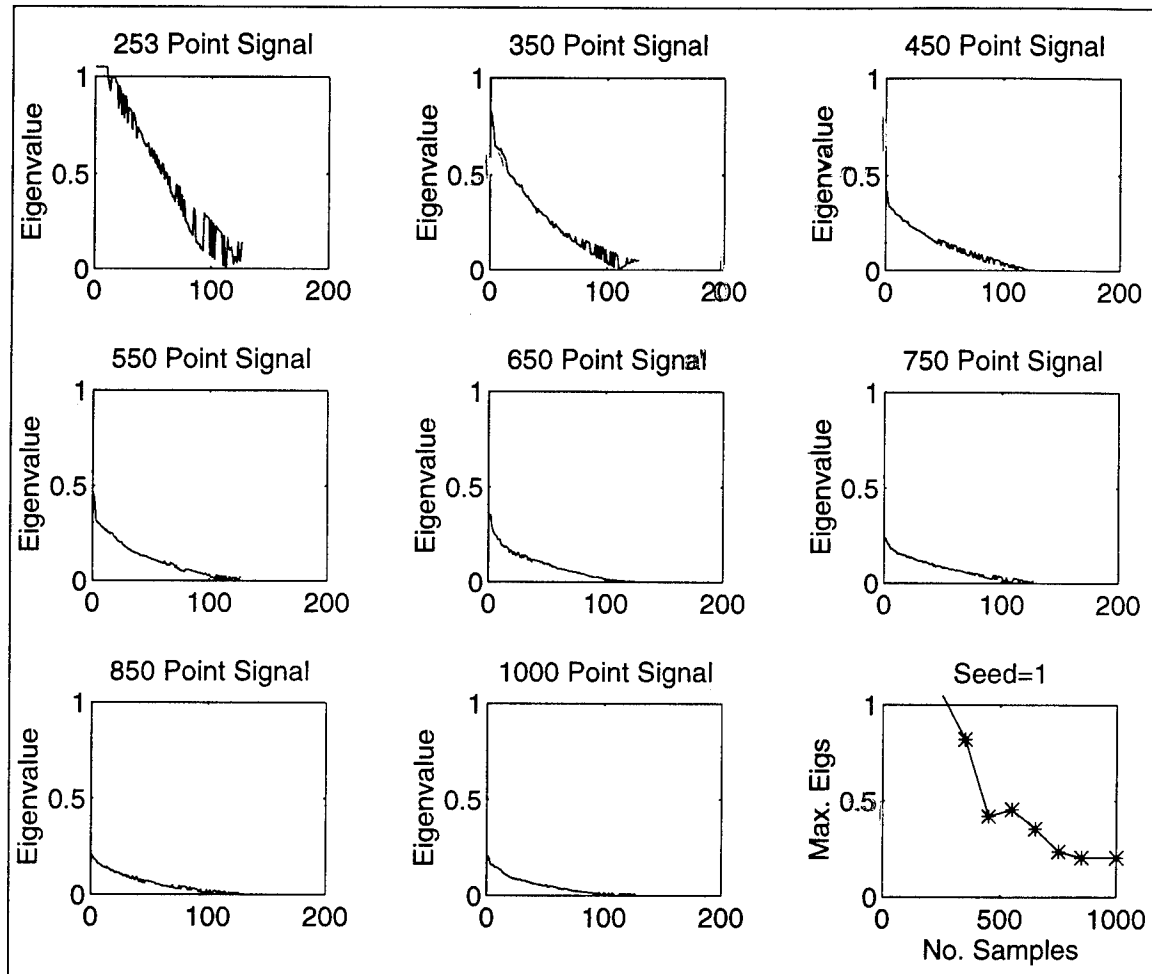


Figure 4.2: All Eigenvalues of the Iteration Matrix for 8 Signals

signal results in a data matrix X with the minimum number of rows required to form the correlation matrix R in the Wiener-Hopf equation. Such a data matrix is not likely to form a full rank correlation matrix of linearly independent rows, a required condition for a unique solution of the Wiener-Hopf equation to exist. Generally, the more rows in the data matrix, the more likely the correlation matrix will be of full rank (non-singular).

[Ref. 3]

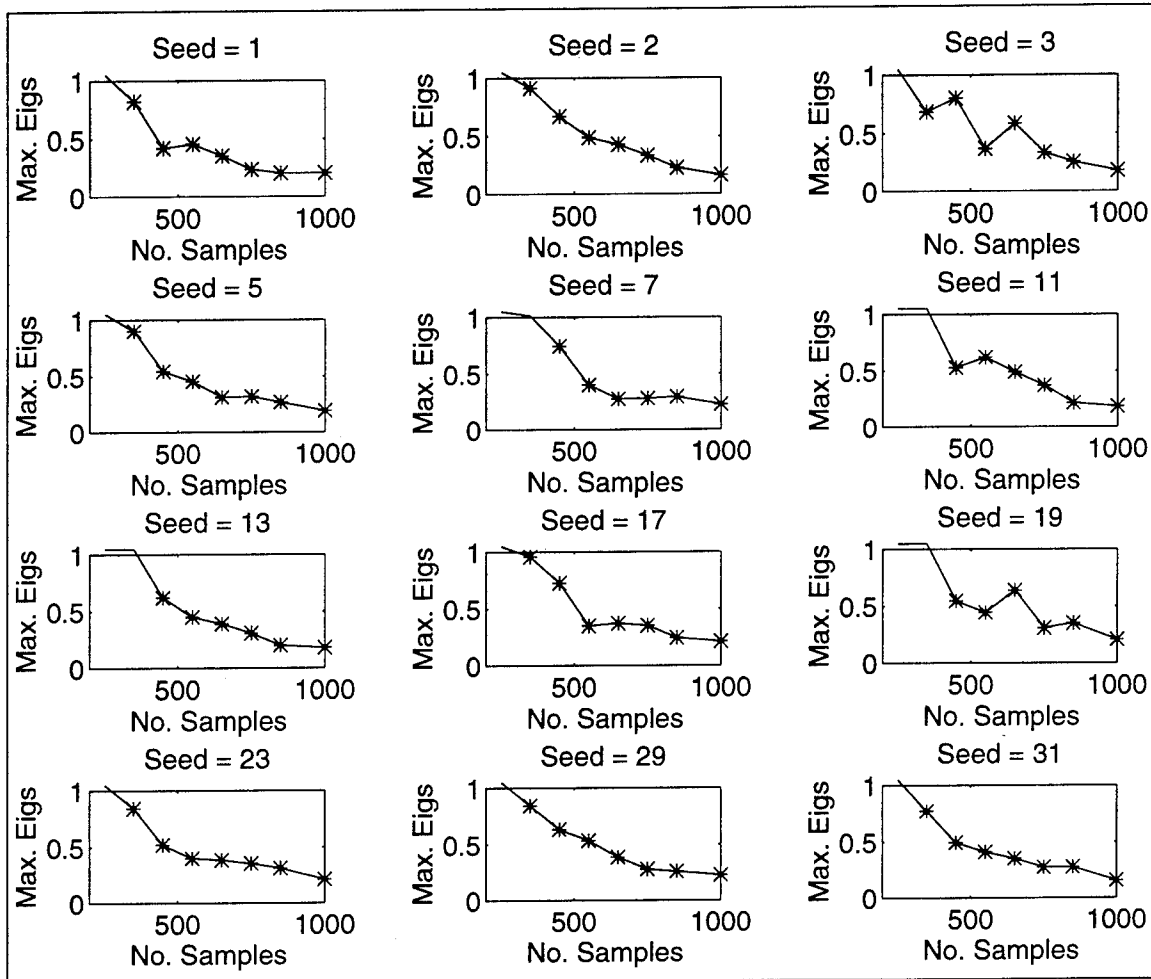


Figure 4.3: Maximum Eigenvalues versus Signal Length for 12 Seeds

The 350 point signal resulted in a Toeplitz iteration matrix with a maximum eigenvalue less than 1.0 for 75 percent of the 12 cases. The maximum eigenvalue was close to 1.0 for the other 25 percent of the 350 point signal cases. Therefore, the Toeplitz approximation algorithm should usually converge to a solution of the Weiner-Hopf equation when input signals of more than 350 points are used to form the correlation matrix. In fact, the maximum eigenvalue of the Toeplitz iteration matrix was less than

1.0 for all test cases involving signals of 450 points or more, regardless of the random number generator seed used. Figure 4.4 shows an average of the maximum eigenvalues for the 12 test cases plotted against input signal length. The results support the hypothesis from the previous experiment that the Toeplitz approximation algorithm is likely to converge to a solution of the Weiner-Hopf equation when the number of points in the input signal is at least four times the order of the FIR filter model.

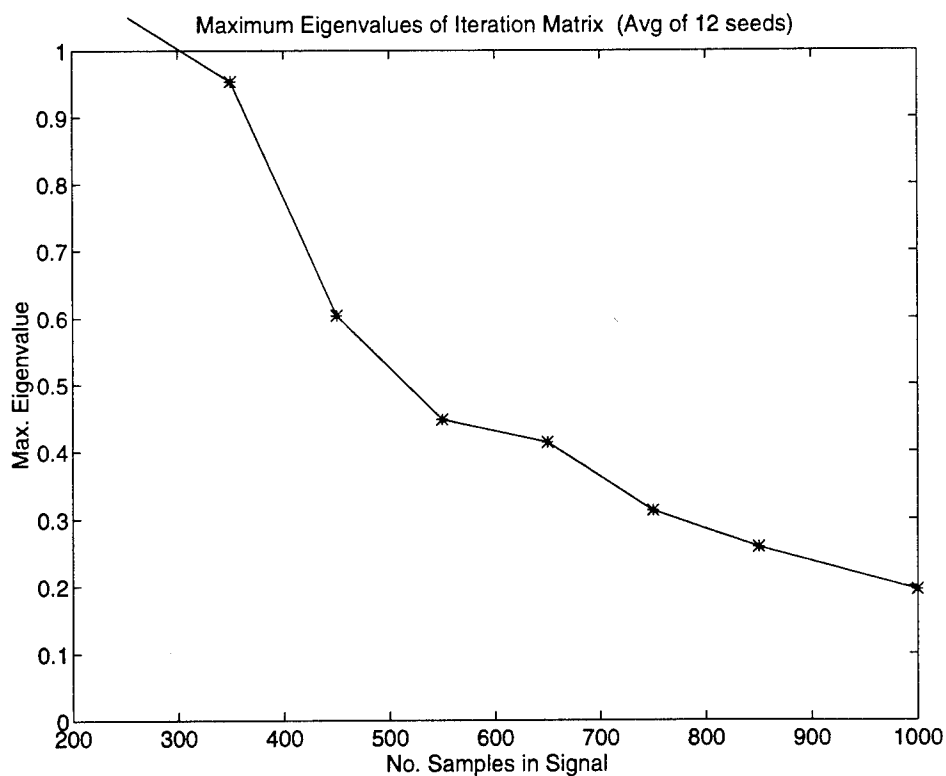


Figure 4.4: Average of Maximum of Eigenvalues vs Signal Length for 12 Seeds

3. Properties of Iteration Matrix Eigenvectors

The eigenvectors of the iteration matrix for a relaxation method often provide clues as to the likelihood of convergence of the iterative algorithm. The previous chapter mentioned the fact that the eigenvectors of both the weighted Jacobi iteration matrix and the eigenvectors of the operator matrix for the weighted Jacobi relaxation method are equivalent to the Fourier modes. It was also mentioned that the weighted Jacobi method does not mix modes. These properties of the weighted Jacobi method contribute to the success of the weighted Jacobi method when used with multigrid techniques. Since the Toeplitz approximation algorithm has been proposed as an alternative relaxation method for use with multigrid, an analysis of the eigenvectors of the Toeplitz iteration matrix seems practical. Therefore, an experiment was conducted to obtain information about the properties of the Toeplitz iteration matrix for various input signals.

As in the previous two experiments, multiple input signals were generated to independently form the 127×127 Toeplitz iteration matrix used in determining the coefficients of a 126^{th} order FIR filter. The minimum length signal (253 points), a signal for which the maximum eigenvalue of the iteration matrix is approximately 1.0 (350 points), and a significantly longer signal (1000 points) were created for a selected random number generator seed. For each signal, the eigenvalues of the iteration matrix were computed and the smallest three, middle three, and largest three eigenvalues were determined. The power spectral density (PSD) was then calculated for the eigenvectors corresponding to each of the nine selected eigenvalues. Plots of the eigenvectors and

their corresponding PSDs are shown in Figures 4.5 through 4.10 for input signals created with a random number generator seed of 1.

Figure 4.5 shows eigenvectors of the Toeplitz iteration matrix for the 253 point input signal. Figure 4.6 shows the PSDs of the same eigenvectors. The three small eigenvalues appear to have eigenvectors with narrow band, low and medium frequency modes. The eigenvectors for the three middle eigenvalues are wide band with medium modes. The eigenvectors for the three middle eigenvalues are wide band with medium

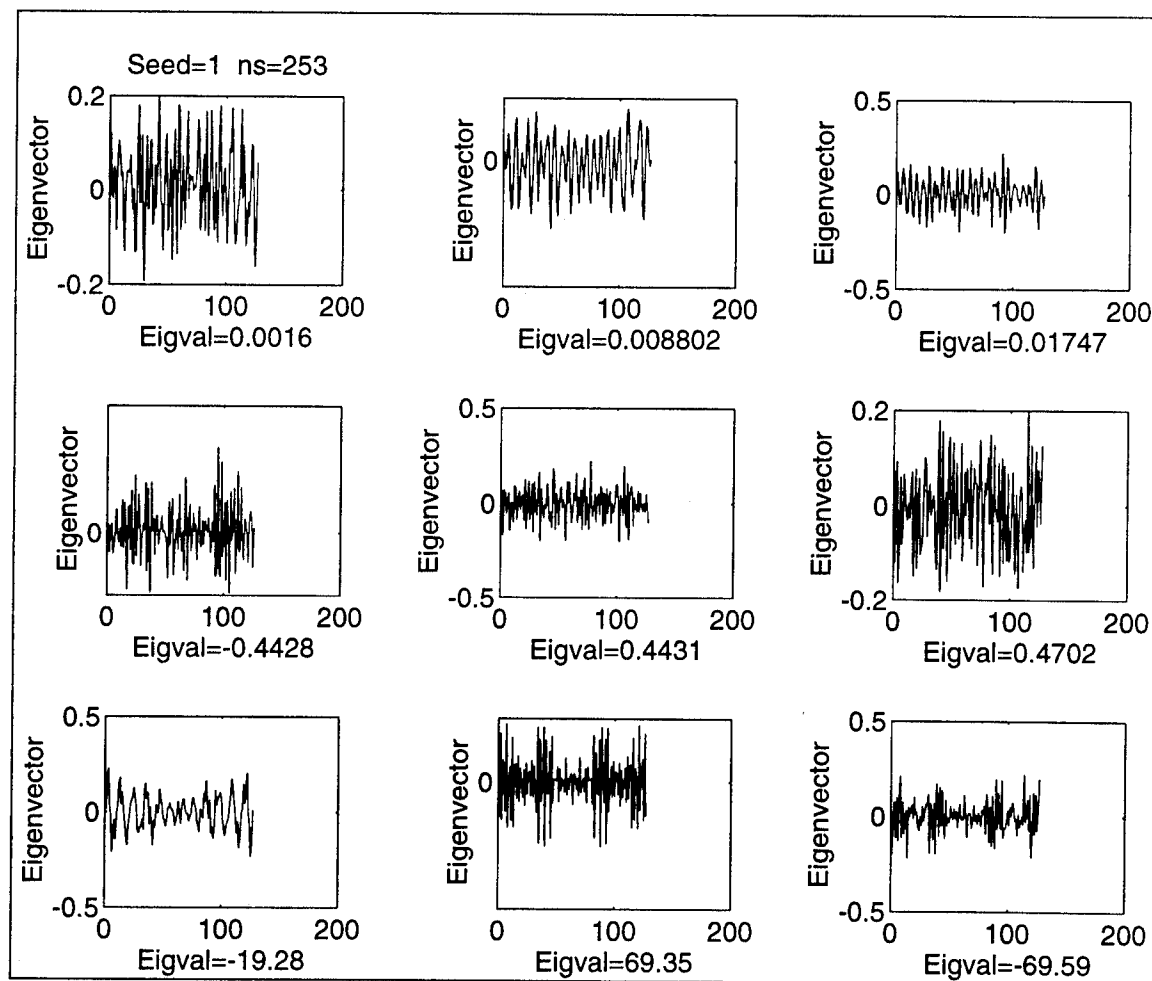


Figure 4.5: Iteration Matrix Eigenvectors for 253 Point Signal

to high frequency modes. The eigenvectors corresponding to the three large eigenvalues are narrow band, but have the unusual characteristic that one contains a low frequency mode, another has a high frequency mode, and the third has low, medium, and high frequency modes. The eigenvectors of the Toeplitz iteration matrix for the 253 point input signal case do not appear to have convenient properties for multigrid to exploit, such as those of the weighted Jacobi iteration matrix.

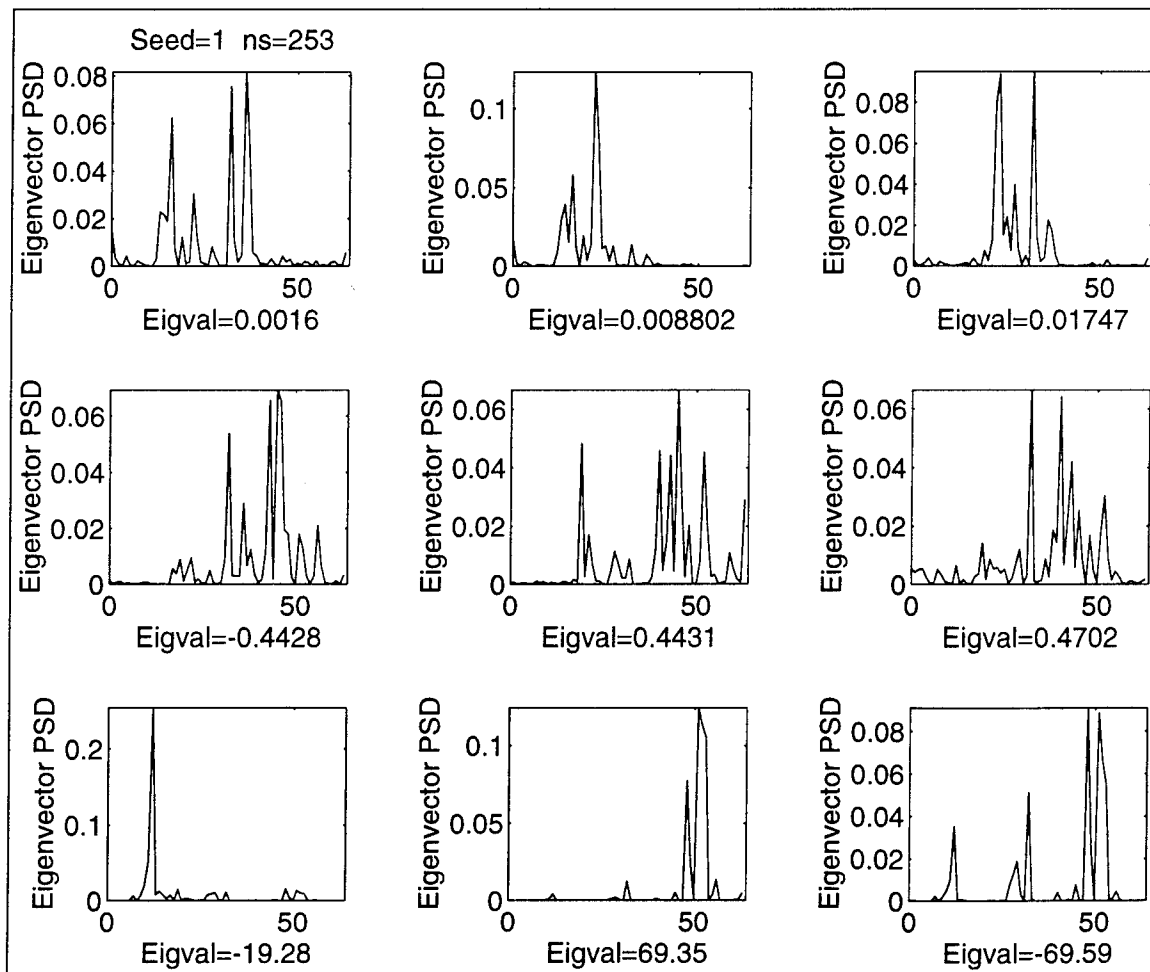


Figure 4.6: PSDs of Iteration Matrix Eigenvectors for 253 Point Signal

Figure 4.7 and Figure 4.8 show the eigenvectors of the Toeplitz iteration matrix and the PSDs of the eigenvectors for the 350 point input signal. The three small eigenvalues have eigenvectors with narrow band frequency spectrums similar to those of the eigenvectors of the three high eigenvalues for the 253 point input signal. One eigenvector contains a low frequency mode, one has a dominant high frequency mode, and the other contains both a high and a low frequency mode. The middle eigenvalues

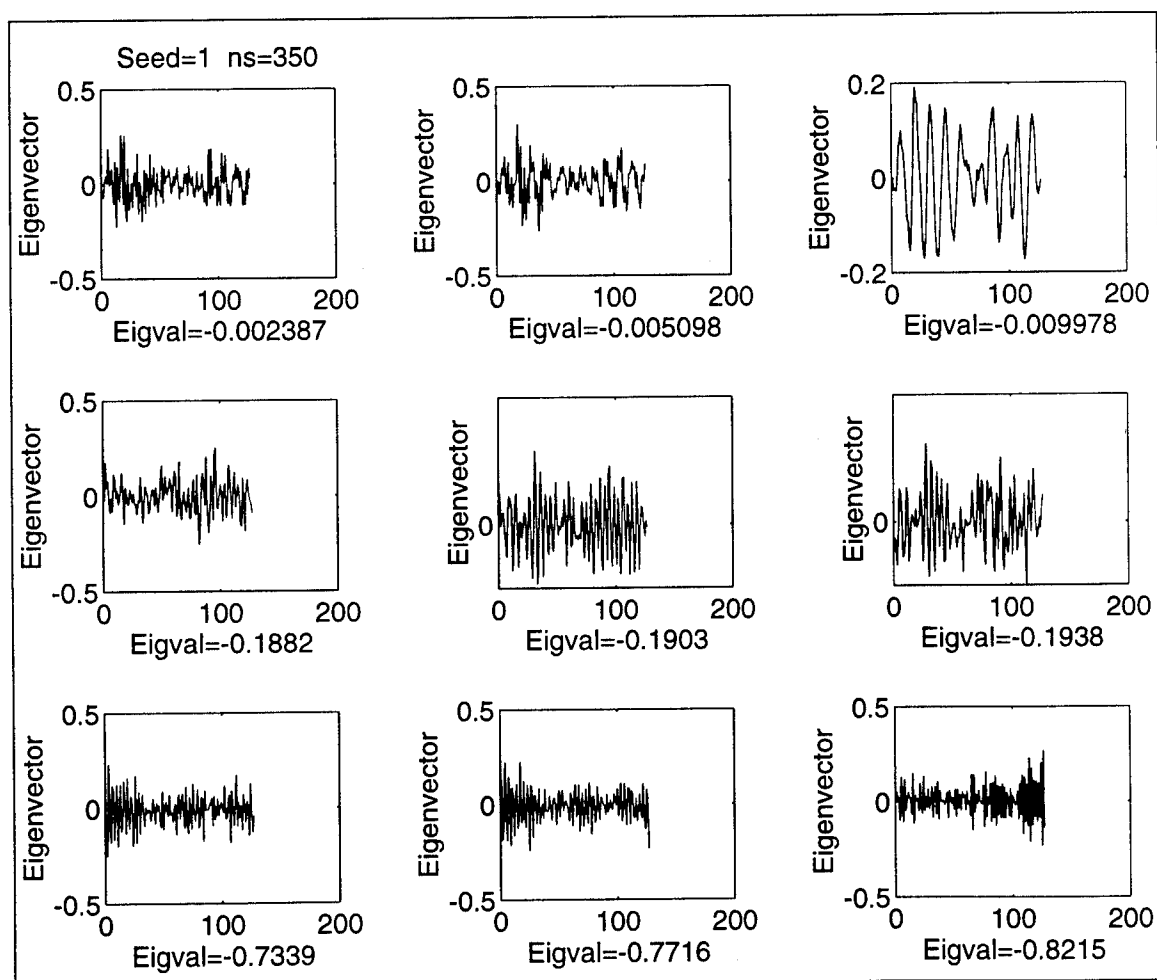


Figure 4.7: Iteration Matrix Eigenvectors for 350 Point Signal

have eigenvectors that have low to medium frequency modes with a slightly wider band than the eigenvectors of the small eigenvalues. The eigenvectors of the three large eigenvalues have narrow band, high frequency modes. As with the 253 point input signal case, the eigenvectors of the Toeplitz iteration matrix for the 350 point input signal do not seem to offer any properties that can easily be exploited by multigrid techniques.

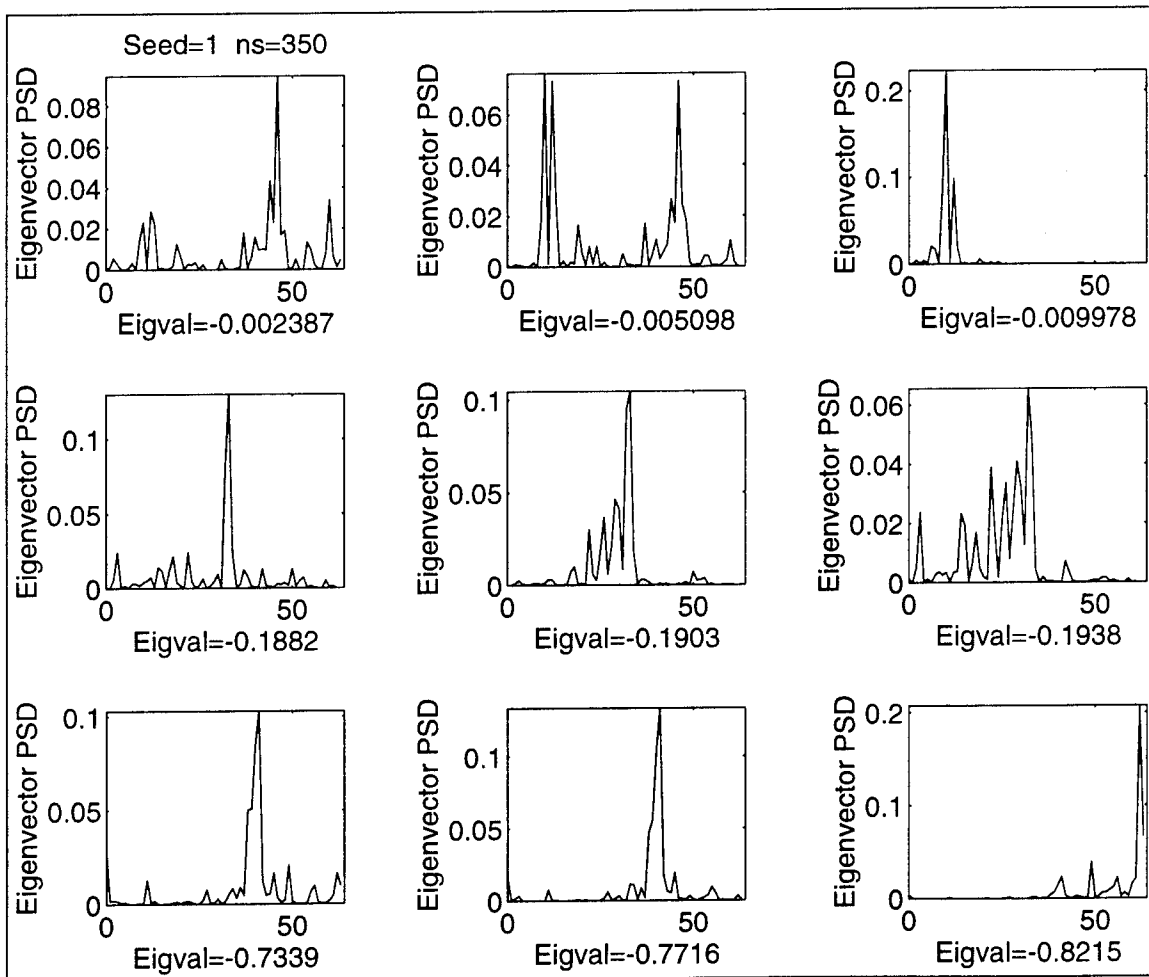


Figure 4.8: PSDs of Iteration Matrix Eigenvectors for 350 Point Signal

Figure 4.9 and Figure 4.10 contain plots of the eigenvectors of the Toeplitz iteration matrix and the PSDs of the eigenvectors for the 1000 point input signal. As with the previous two input signals, the eigenvectors of the three small eigenvalues have narrow band frequencies. Two of the eigenvectors contain a dominant high frequency mode, while the third eigenvector contains high and low frequency modes. The eigenvectors of the three middle eigenvalues are again wider band than other eigenvectors. The eigenvectors of the three middle eigenvalues are again wider band than other eigenvectors.

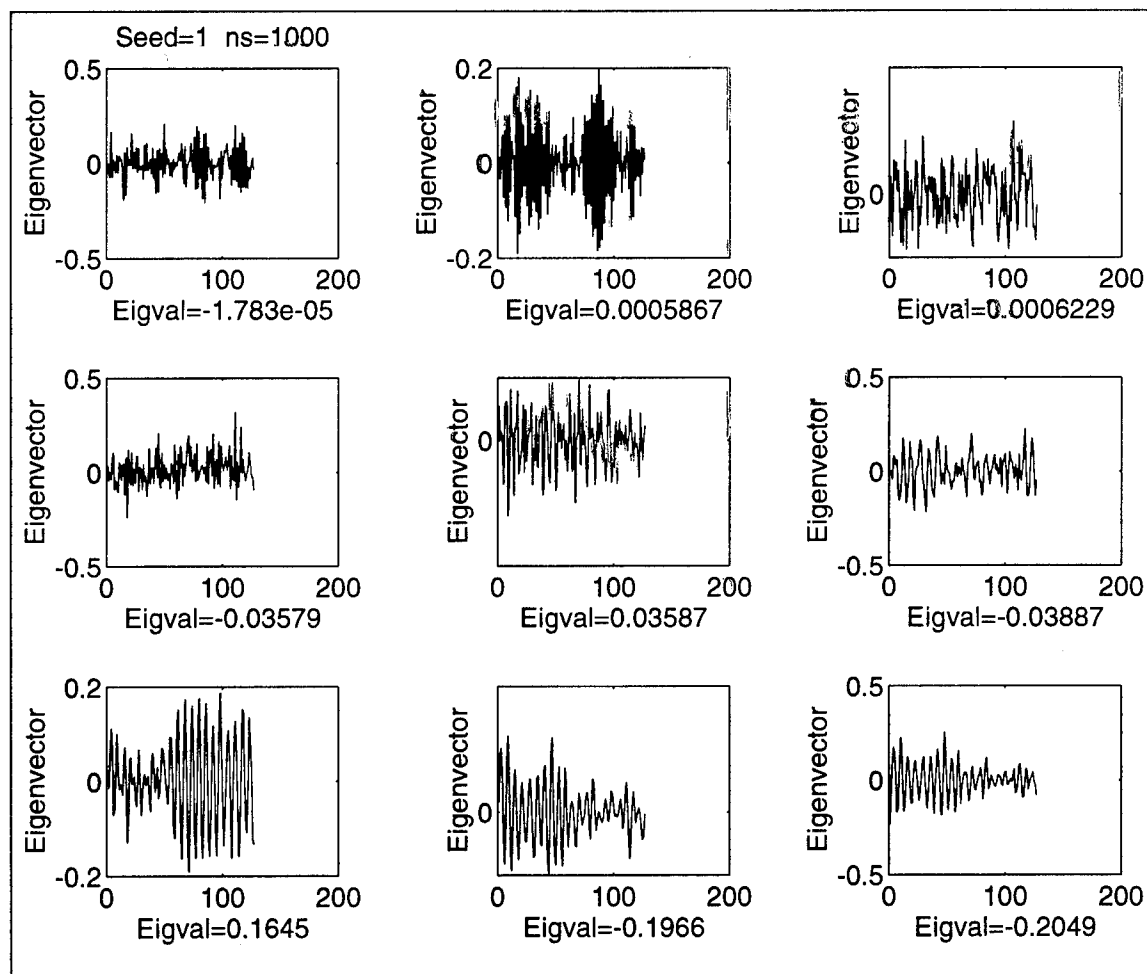


Figure 4.9: Iteration Matrix Eigenvectors for 1000 Point Signal

One eigenvector has a low frequency mode, one has a high frequency mode, and the third has medium to high frequency modes. Each of the large eigenvalues have narrow band eigenvectors with a low frequency mode. The eigenvectors of the Toeplitz iteration matrix for the 1000 point input signal do not display a discernible pattern that can benefit multigrid.

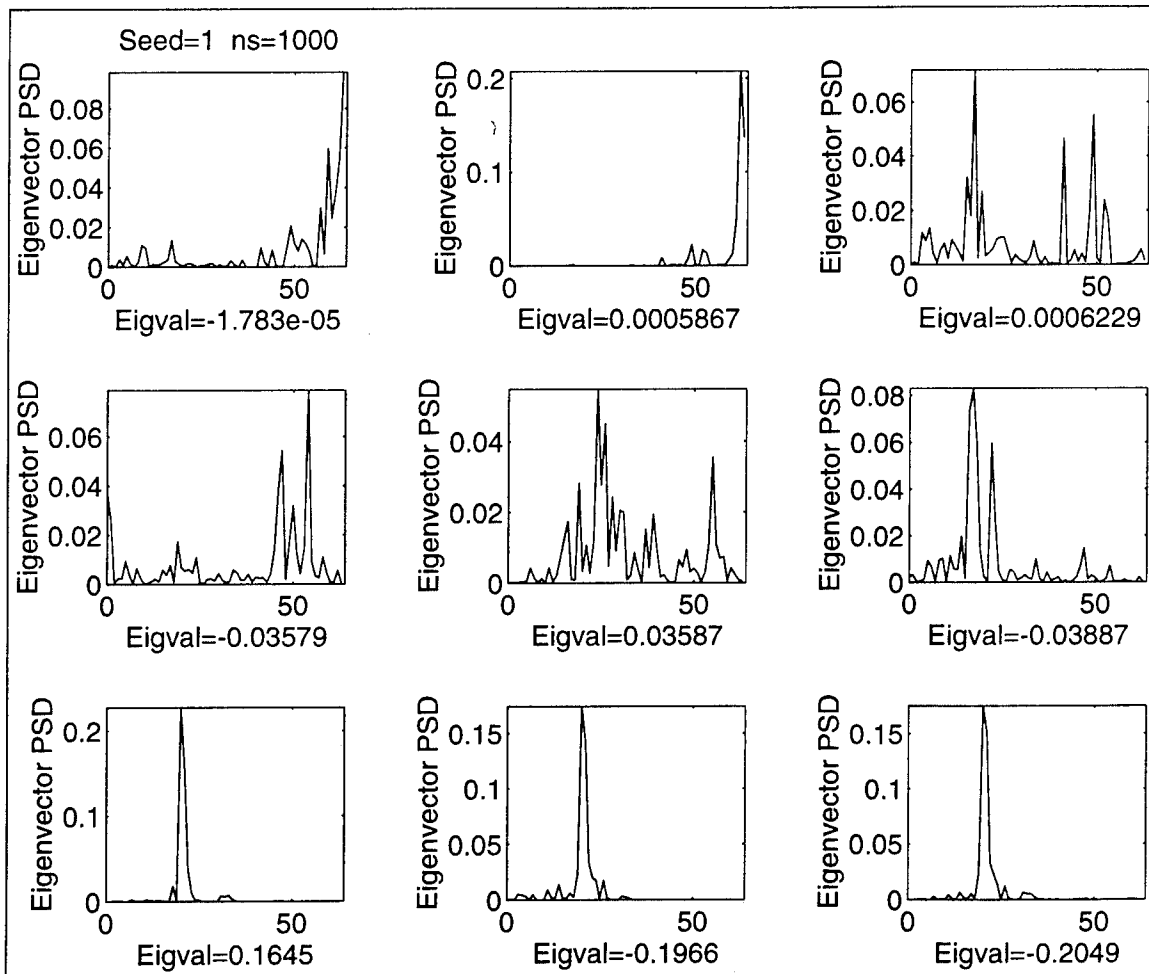


Figure 4.10: PSDs of Iteration Matrix Eigenvectors for 1000 Point Signal

Table 4.1 summarizes the frequency content trends for each of the cases. Analysis of the spectral content of the eigenvectors of the Toeplitz iteration matrix does not clearly identify any properties of the Toeplitz iteration matrix that would facilitate the use of multigrid techniques with the Toeplitz approximation algorithm. Most of the eigenvectors are narrow band, but the eigenvectors of the middle eigenvalues tend to be slightly wider band than the eigenvectors of the small and large eigenvalues. The frequency modes of each eigenvector do not appear to have any correlation to the size of

Table 4.1: Frequency Content of Toeplitz Iteration Matrix Eigenvectors

Points in Input Signal	Frequency spectrum for eigenvectors of:								
	Smallest 3 Eigenvalues			Middle 3 Eigenvalues			Largest 3 Eigenvalues		
253	low medium	low medium	medium	medium high	low medium high	medium high	low	high	low medium high
350	high	low high	low	medium	medium	low medium	high	high	high
1000	high	high	low high	high	medium high	low	low	low	low

the corresponding eigenvalue or the number of points in the input signal. The results of the Toeplitz iteration matrix eigenvector analysis do not appear to give any useful clues as to how the Toeplitz approximation algorithm might be improved with multigrid.

C. TOEPLITZ APPROXIMATION ALGORITHM ANALYSIS

Experiments described in the previous section were designed to examine the properties of the Toeplitz iteration matrix used in the Toeplitz approximation algorithm. Additional experiments were conducted to analyze the behavior of the Toeplitz approximation algorithm when applied to the Weiner-Hopf equation in various scenarios. Computer simulations were run to examine the convergence of the Toeplitz approximation algorithm to a zero solution of the Weiner-Hopf equation using eigenvectors of the Toeplitz operator (correlation) matrix as initial guesses. Also, the convergence of the Toeplitz approximation algorithm to an actual solution of an optimal FIR filter was examined. These experiments, like the experiments in the previous section, were designed to provide empirical evidence as to whether or not the Toeplitz approximation algorithm can benefit from multigrid techniques.

1. Convergence to a Zero Solution

To analyze the usefulness of a relaxation method, it is important to determine what the iterative algorithm does to the various frequency components of the error during each iteration. As discussed in the previous chapter, many relaxation methods possess a smoothing property where oscillatory components of the error are quickly removed

while smooth components of the error remain. Multigrid techniques have the capability to effectively handle the smoothing property of relaxation methods. In order to explore how multigrid might help the Toeplitz approximation algorithm, an experiment was conducted to determine the effect of the Toeplitz approximation algorithm on the various components of the error when applied to the Weiner-Hopf equation.

The experiment involved the creation of a 127×127 Toeplitz operator (correlation) matrix for the same input signals described in the previous section. Input signals of length 253 points (minimum signal length), 350 points (maximum eigenvalue of the Toeplitz iteration matrix is approximately 1.0), and 1000 points were created for a selected random number generator seed. The smallest, middle, and largest eigenvalues of the operator matrix were determined and each corresponding eigenvector was used as an initial guess for the Toeplitz approximation algorithm with the solution set to zero. Thus, the Toeplitz approximation algorithm was used to solve the equation

$$\mathbf{R}\mathbf{b} = \mathbf{0} \quad (4.2)$$

where $\mathbf{R}$ is the correlation matrix, $\mathbf{b}$ is the coefficient vector for the optimal FIR filter, and $\mathbf{0}$ is a vector of all zeros.

For each input signal, the Toeplitz approximation algorithm was run three times for 20 iterations, using eigenvectors of the operator matrix as initial guesses. The convergence or divergence of the algorithm was examined by determining the error norm

of the coefficient vector computed by the Toeplitz approximation algorithm after each iteration. The error norm of the coefficient vector is

$$\|e\| = \sqrt{\sum_{k=0}^P |b(k) - \hat{b}(k)|^2} = \sqrt{\sum_{k=0}^P |b(k)|^2} \quad (4.3)$$

where P is the FIR filter order, $b(k)$ is always zero, and $\hat{b}(k)$ is the estimate of the k^{th} coefficient of the FIR filter model. PSDs of the error vector were also computed after each iteration to determine the effect of the Toeplitz approximation algorithm on the various frequency components of the error. Plots of the results of the experiment are shown for a random number generator seed of 1.

Figure 4.11 shows a plot of all the eigenvalues of the Toeplitz operator matrix for the 253 point input signal. The figure also shows the eigenvectors of the Toeplitz operator matrix, R , that were used as initial guesses for the Toeplitz approximation algorithm. The eigenvectors correspond to the smallest, middle, and largest eigenvalues of the operator matrix. Figure 4.12 shows a plot of error norm versus iteration number when running the Toeplitz approximation algorithm for the 253 point input signal case. Note that the error at iteration 0 represents the error of the initial guess. The initial error shows how far the eigenvector is from the zero solution and is always equal to 1.0 because the eigenvectors of the operator matrix are normalized. The 'o' symbol on the plot indicates that the eigenvector corresponding to the smallest eigenvalue of the operator matrix was used as the initial guess for the Toeplitz algorithm. The 'x' and '+' symbols

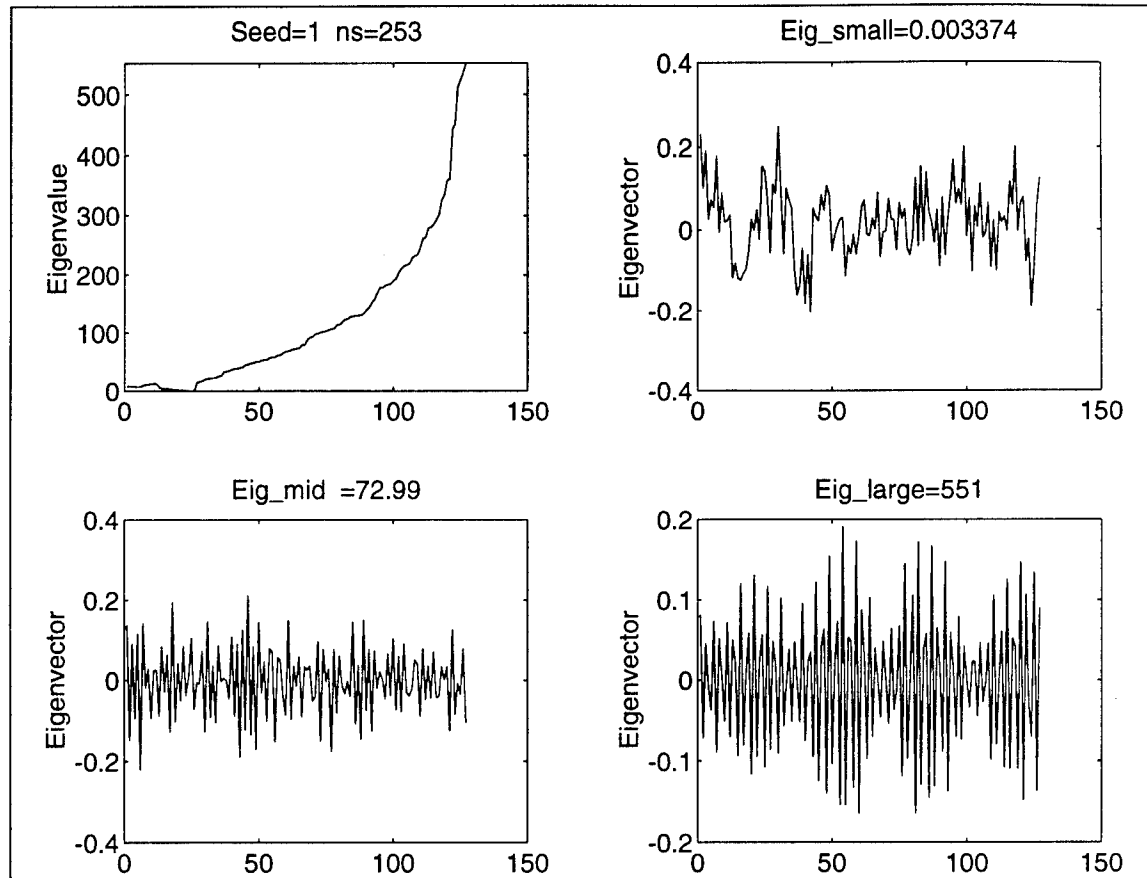


Figure 4.11: Eigenvalues and Initial Guesses for 253 Point Signal

correspond to eigenvectors of the middle and largest eigenvalues, respectively.

The Toeplitz approximation algorithm diverged for each of the three initial guesses for the 253 point input signal case. The algorithm appeared to diverge the slowest for the first three iterations using the eigenvector of the smallest eigenvalue as the initial guess. The divergence rate of the algorithm was also slower for the first iteration with the middle eigenvector initial guess than for the first iteration with the initial guess of the eigenvector of the largest eigenvalue. However, the divergence rate was the same for all three initial guesses by the fourth iteration.

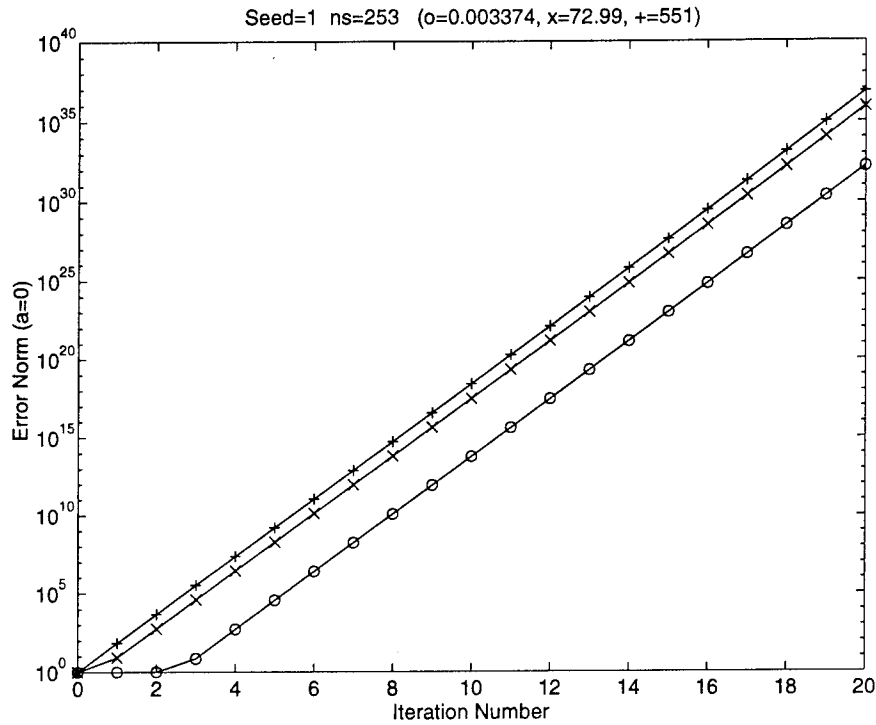


Figure 4.12: Zero Solution Error Norms for 253 Point Signal

Figure 4.13 through Figure 4.15 show the PSDs of the error vectors after the first five iterations and the 10^{th} , 15^{th} , and 20^{th} iterations of the Toeplitz approximation algorithm for different initial guesses of the 253 point input signal case. Figure 4.13 shows the PSDs when the eigenvector corresponding to the smallest eigenvalue of the operator matrix was used as the initial guess. The Toeplitz approximation algorithm appears to reduce most frequency components of the error, but excites a low, medium, and dominant high frequency component of the error. This result is contradictory to the smoothing property found in most relaxation methods.

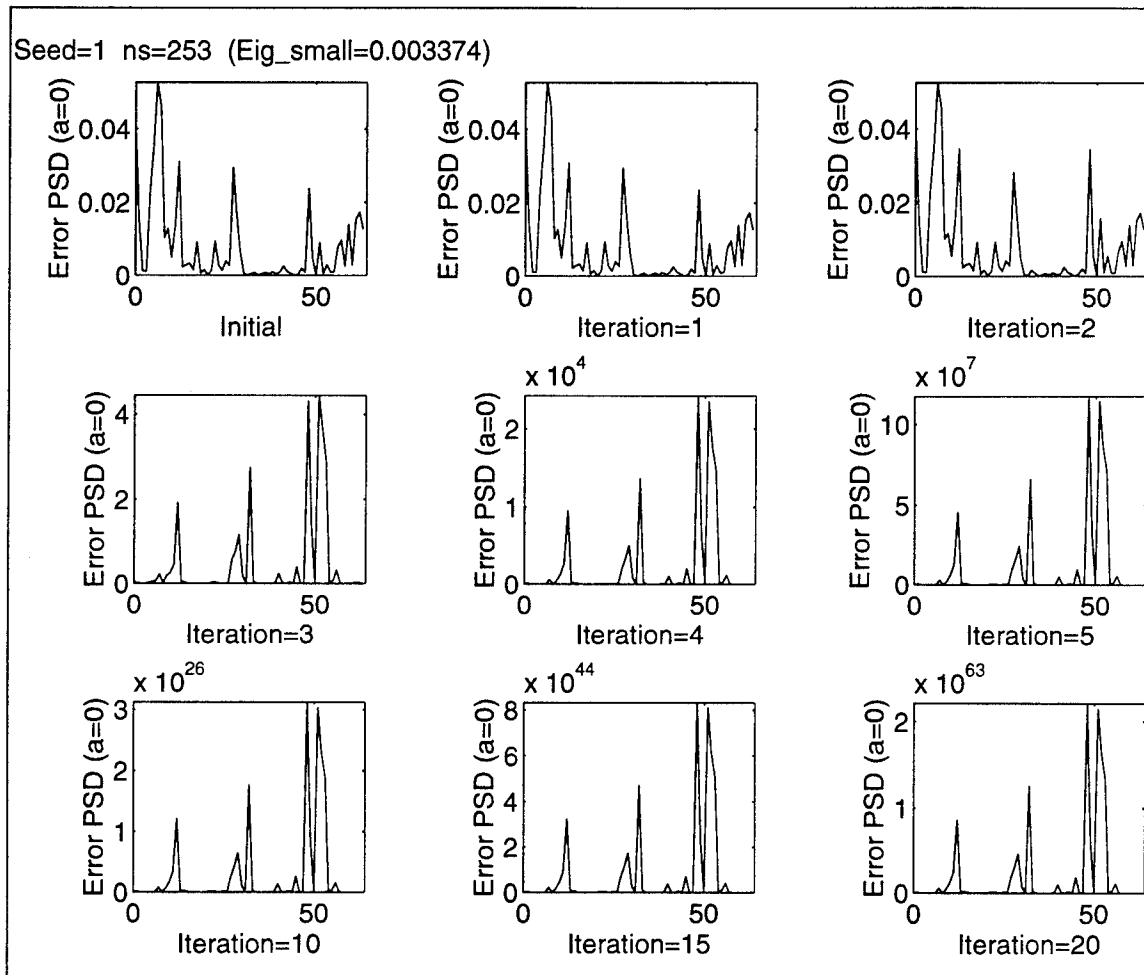


Figure 4.13: PSD of Error for 253 Point Signal, Smallest Eigenvalue

Figure 4.14 shows the PSDs for the case when the middle eigenvector is used as the initial guess for the Toeplitz approximation algorithm. The initial error contains a single dominant high frequency component which the algorithm excites in the same manner as in the previous case with the eigenvector of the smallest eigenvalue as the initial guess. However, the low and medium frequency components are not affected as much in this case.

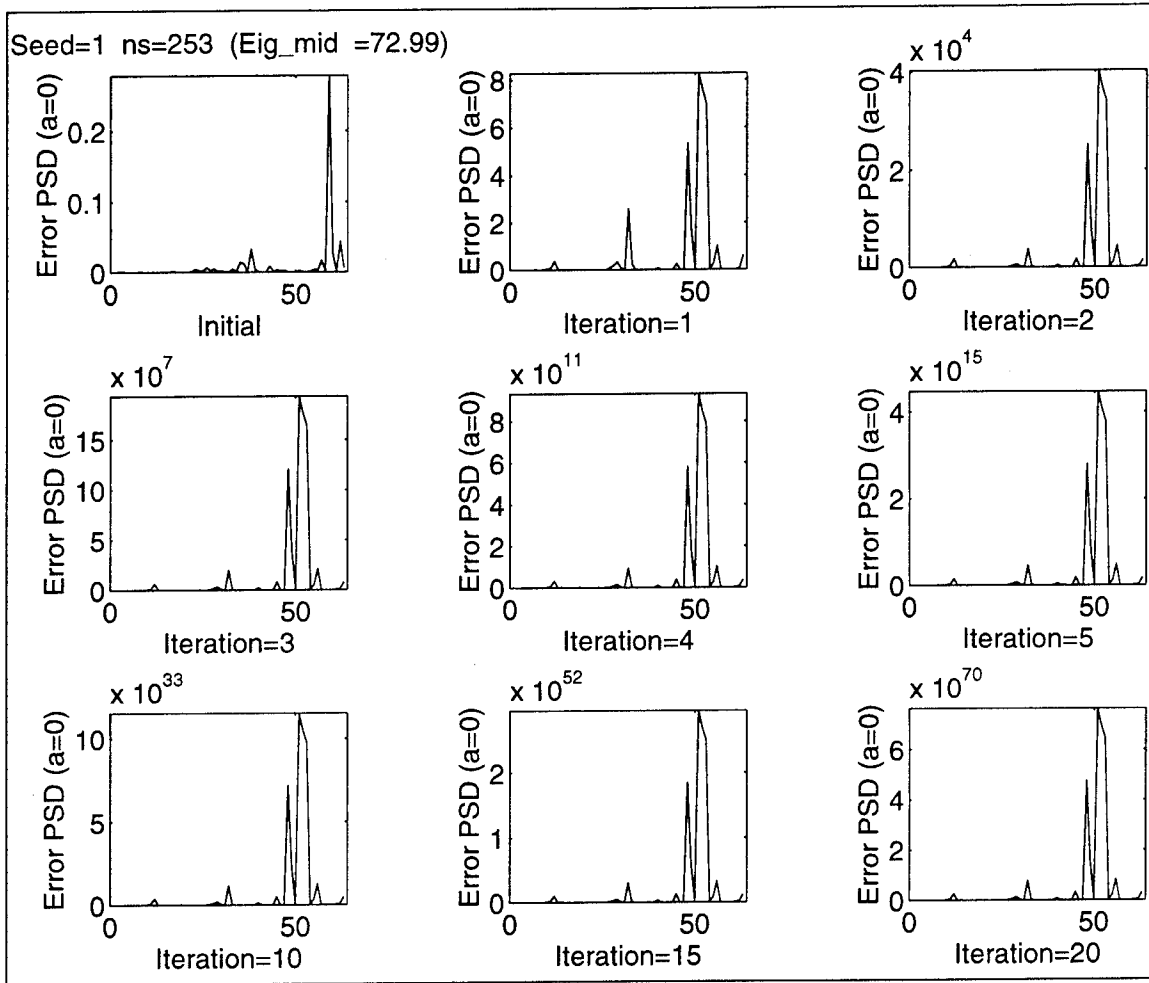


Figure 4.14: PSD of Error for 253 Point Signal, Middle Eigenvalue

Figure 4.15 shows the results when the eigenvector of the largest eigenvalue of the operator matrix is used as the initial guess. The algorithm excites the high frequency error components contained in the initial guess and introduces the same low and medium frequency components into the error that are prevalent in the case with the eigenvector of the smallest eigenvalue as the initial guess. The common result for the three initial guesses for the 253 point input signal case is that the high frequency error component is

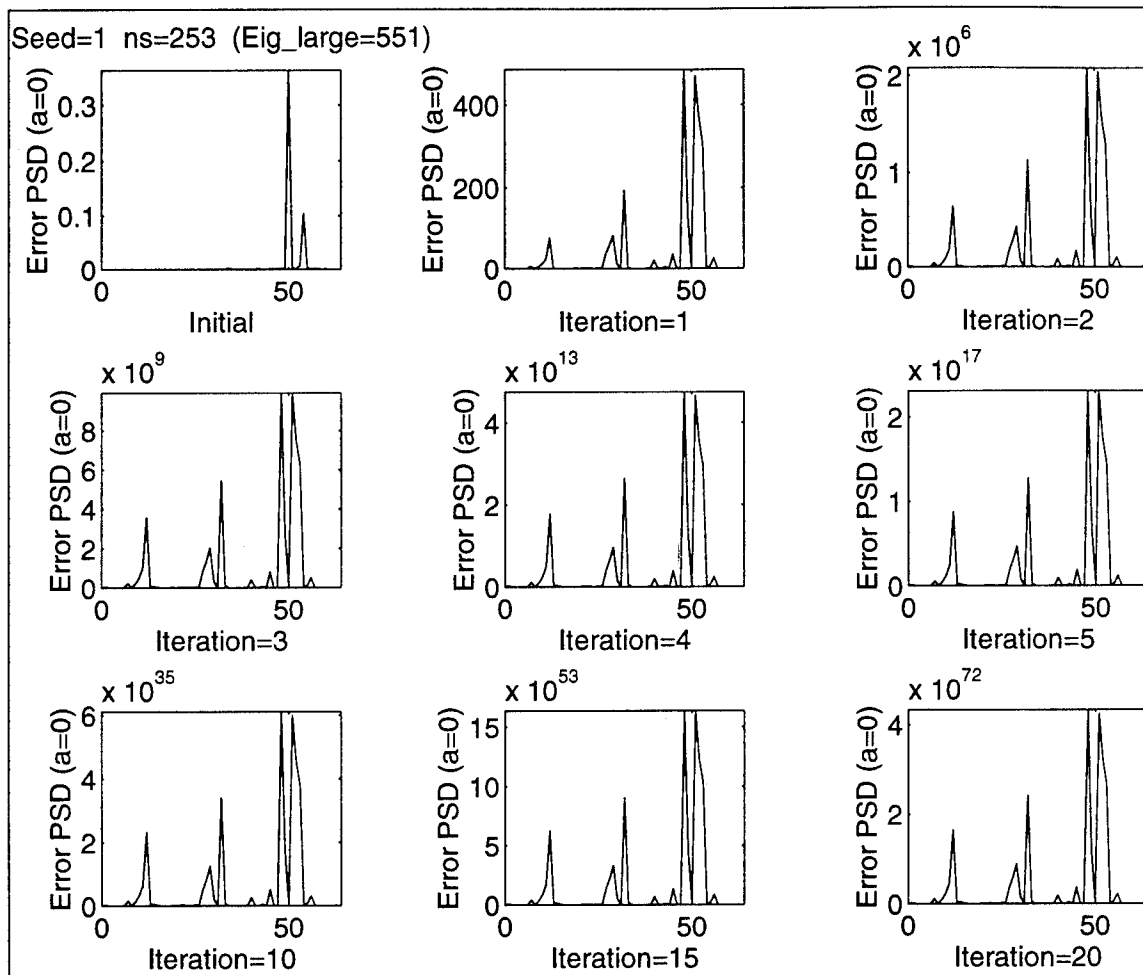


Figure 4.15: PSD of Error for 253 Point Signal, Largest Eigenvalue

excited after each iteration. Low and medium frequency components of the error may also be excited.

Figure 4.16 shows a plot of all the eigenvalues of the Toeplitz operator matrix for the 350 point input signal and the eigenvectors of the Toeplitz operator matrix that were used as initial guesses for the Toeplitz approximation algorithm. Figure 4.17 shows a plot of error norm versus iteration number when running the Toeplitz approximation algorithm for the 350 point input signal case. In this case, the algorithm actually

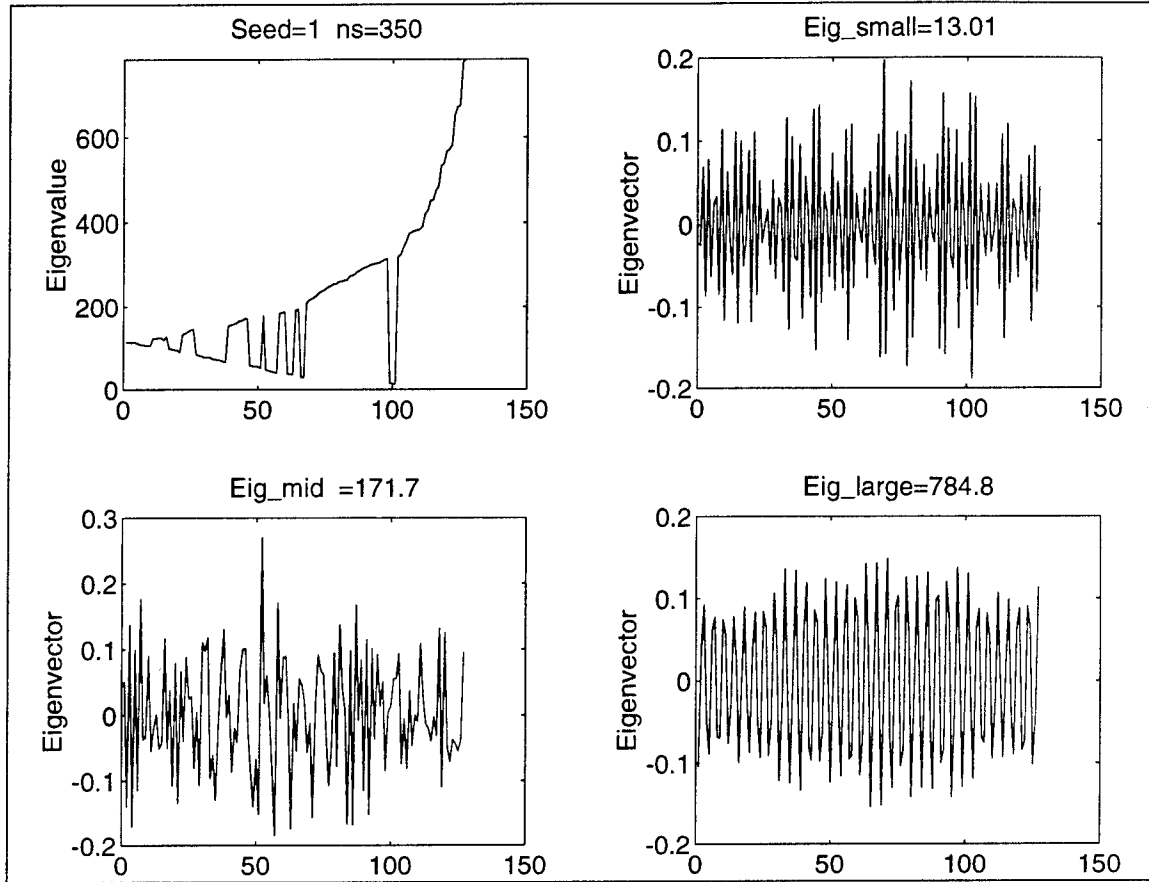


Figure 4.16: Eigenvalues and Initial Guesses for 350 Point Signal

converged for four iterations before diverging when the eigenvector of the largest eigenvalue was used as the initial guess. The eigenvector of the smallest eigenvalue reduced the error by about the same amount after five iterations. However, the error alternately converged and diverged for five iterations before the divergence rate appeared to become constant. The result was similar when the eigenvector of the middle eigenvalue was used as the initial guess, with convergence for two iterations followed by an alternating divergence and convergence before a constant divergence. For all three initial guesses in the 350 point input signal case, the algorithm diverged at a constant rate

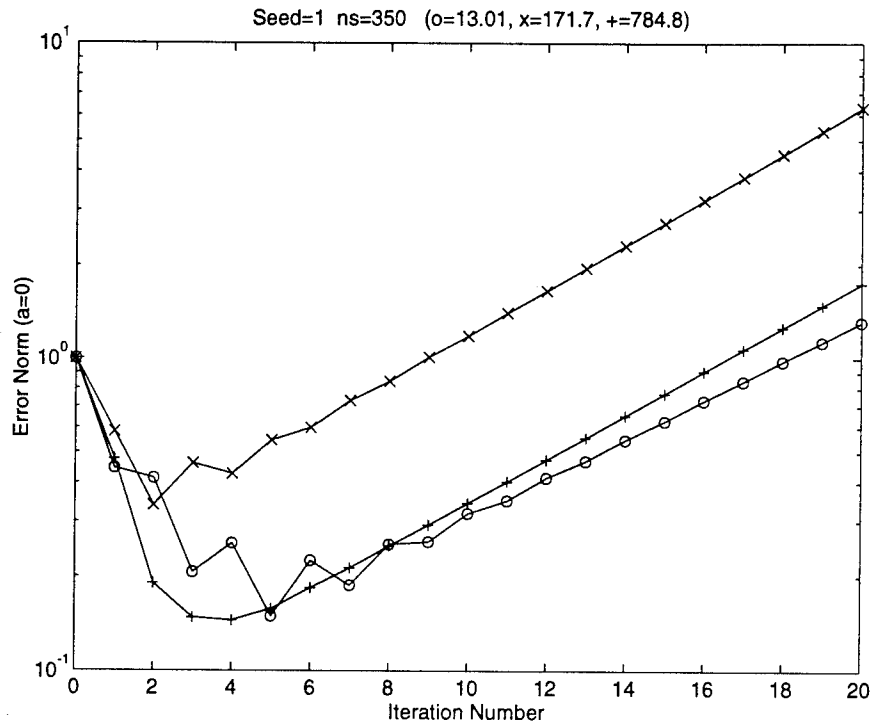


Figure 4.17: Zero Solution Error Norms for 350 Point Signal

after a few iterations. The rate of divergence was slowest for the eigenvector of the smallest eigenvalue and fastest for the eigenvector of the middle eigenvalue initial guess.

Figure 4.18 through Figure 4.20 show the PSDs of the error vectors after some of the iterations of the Toeplitz approximation algorithm for different initial guesses of the 350 point input signal case. Figure 4.18 shows the PSDs when the eigenvector of the smallest eigenvalue of the operator matrix was used as the initial guess. As with the 253 point signal case, the Toeplitz approximation algorithm appears to excite the dominant high frequency component of the error.

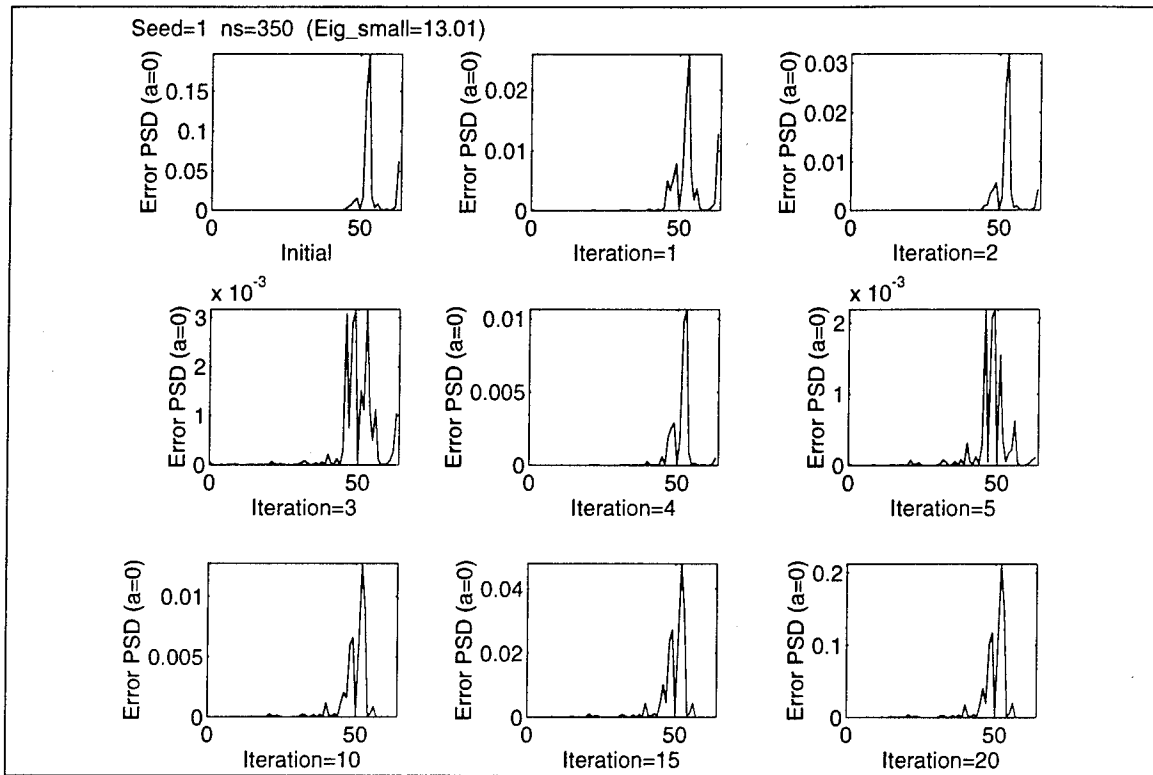


Figure 4.18: PSD of Error for 350 Point Signal, Smallest Eigenvalue

Figure 4.19 shows the PSDs for the case when the middle eigenvector is used as the initial guess for the Toeplitz approximation algorithm. The algorithm reduces the low and medium frequency components of the error but excites the high frequency error component.

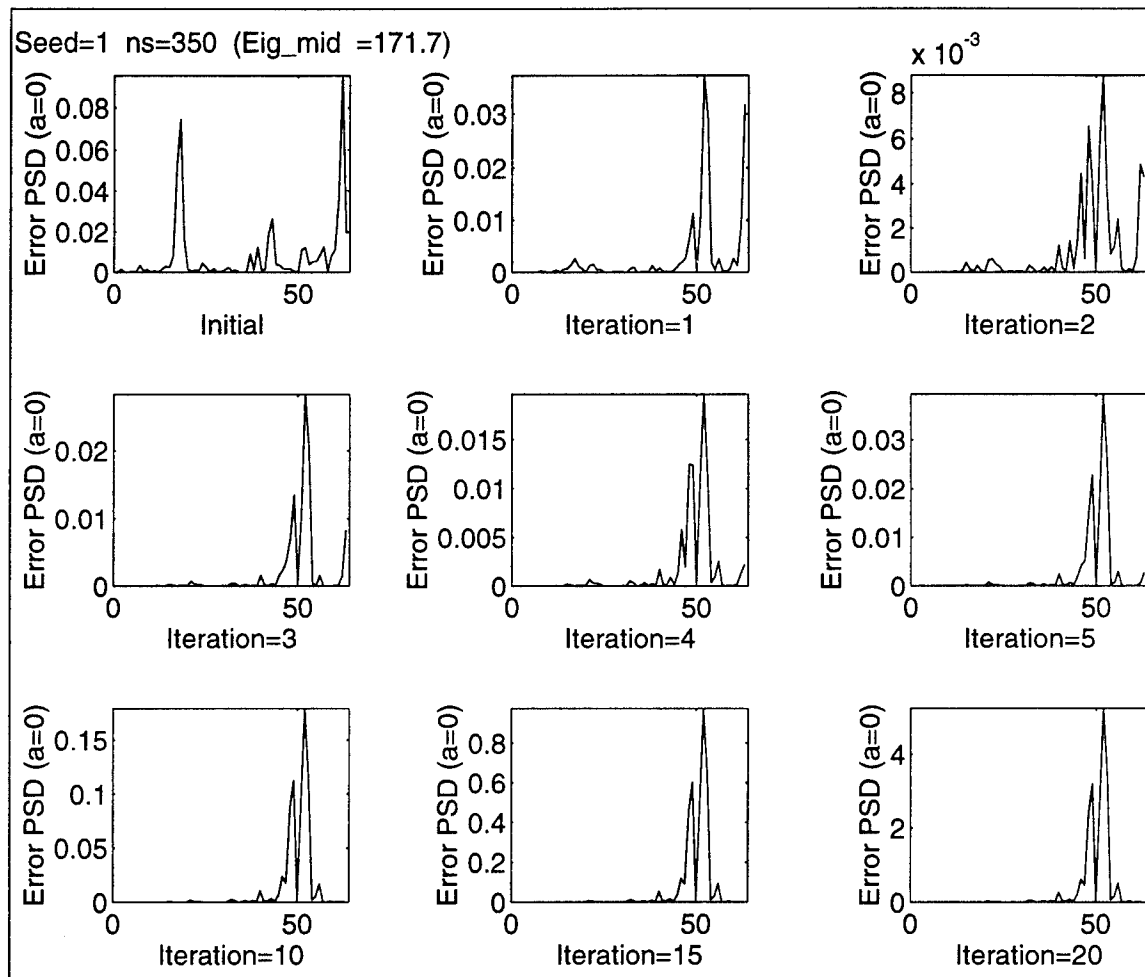


Figure 4.19: PSD of Error for 350 Point Signal, Middle Eigenvalue

Figure 4.20 shows the PSDs of the error when the eigenvector of the largest eigenvalue is used as the initial guess. The initial error consists of a single medium frequency mode that is eliminated after a few iterations while a high frequency component is introduced and excited by the algorithm. Thus, as with the 253 point input signal case, the Toeplitz approximation algorithm appears to have an opposite effect than the smoothing property.

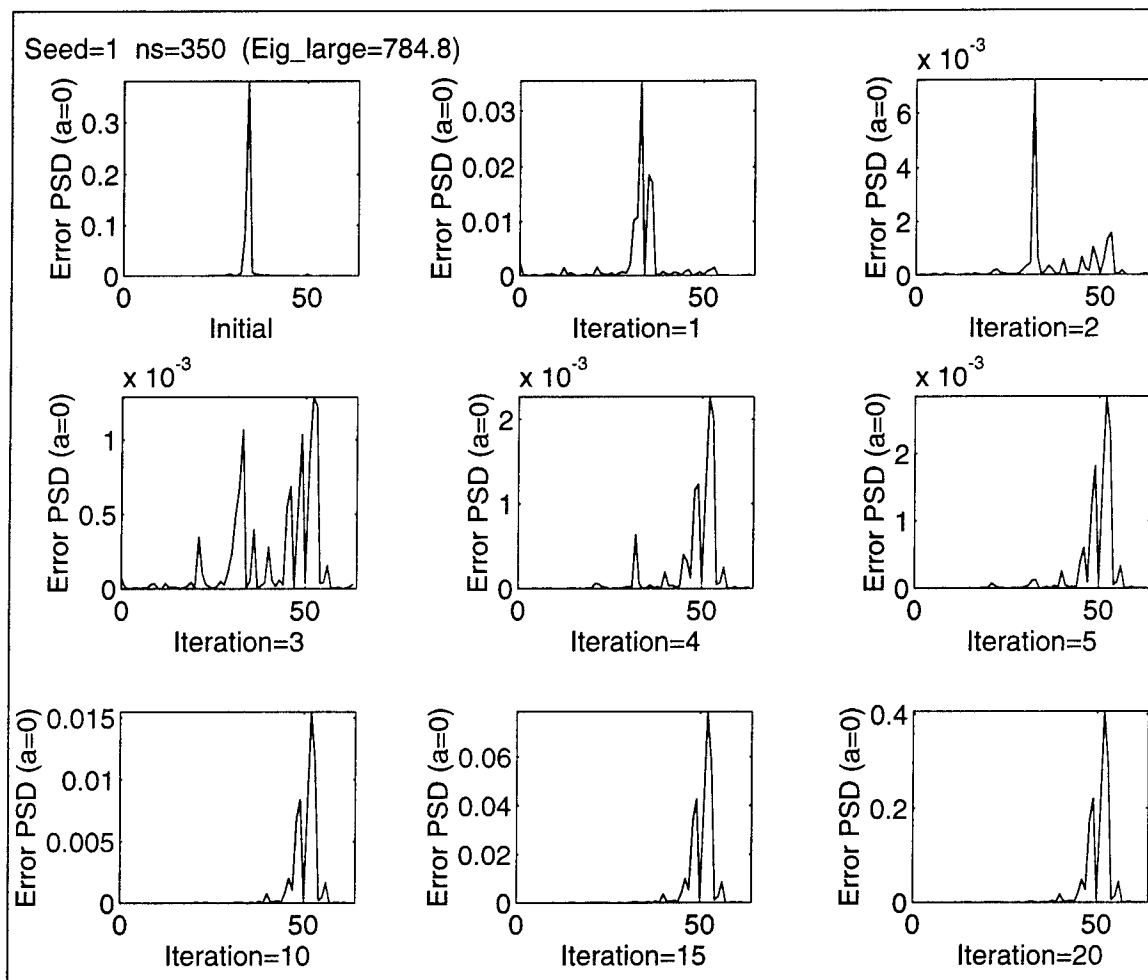


Figure 4.20: PSD of Error for 350 Point Signal, Largest Eigenvalue

Figure 4.21 shows a plot of all the eigenvalues of the Toeplitz operator matrix for the 1000 point input signal and the eigenvectors of the Toeplitz operator matrix that were used as initial guesses for the Toeplitz approximation algorithm. The Toeplitz approximation algorithm results were much more favorable for the 1000 point input signal case than for the previous two cases.

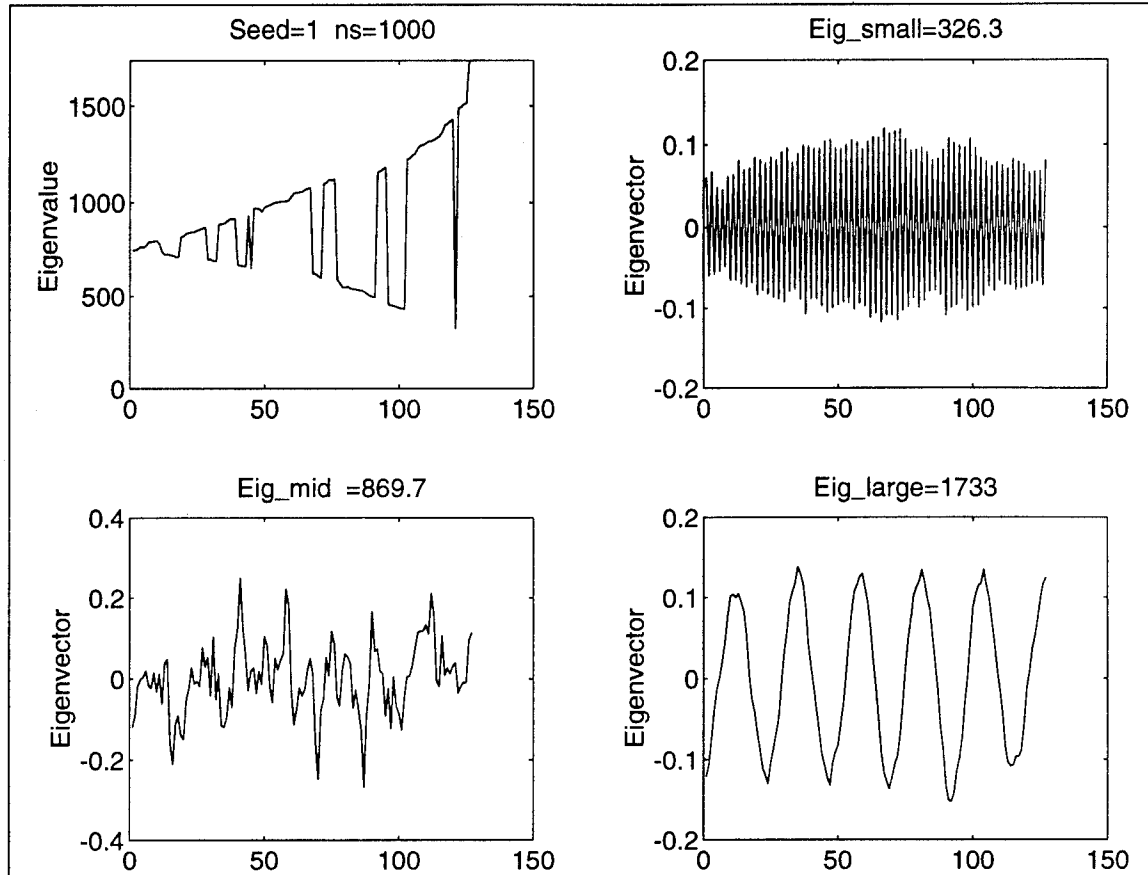


Figure 4.21: Eigenvalues and Initial Guesses for 1000 Point Signal

As shown in Figure 4.22, the Toeplitz approximation algorithm converges quickly for all three initial guesses with the 1000 point input signal. For each initial guess, the error decreases by an order of magnitude after each iteration. Figure 4.23 through Figure 4.25 show the PSDs of the error vectors after various iterations of the Toeplitz approximation algorithm for different initial guesses of the 1000 point input signal case.

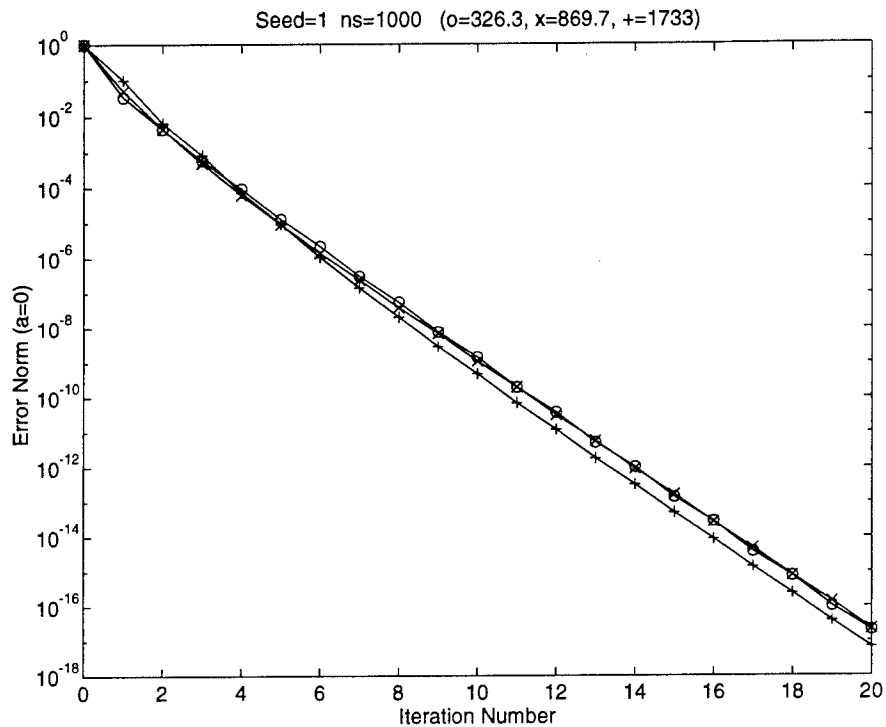


Figure 4.22: Zero Solution Error Norms for 1000 Point Signal

Figure 4.23 shows the PSDs when the eigenvector of the smallest eigenvalue of the operator matrix was used as the initial guess. As with the 253 point signal and 350 point signal cases, the Toeplitz approximation algorithm appears to reduce all errors to a high frequency component. A single high frequency mode is replaced with another high frequency error component that is slightly lower in frequency.

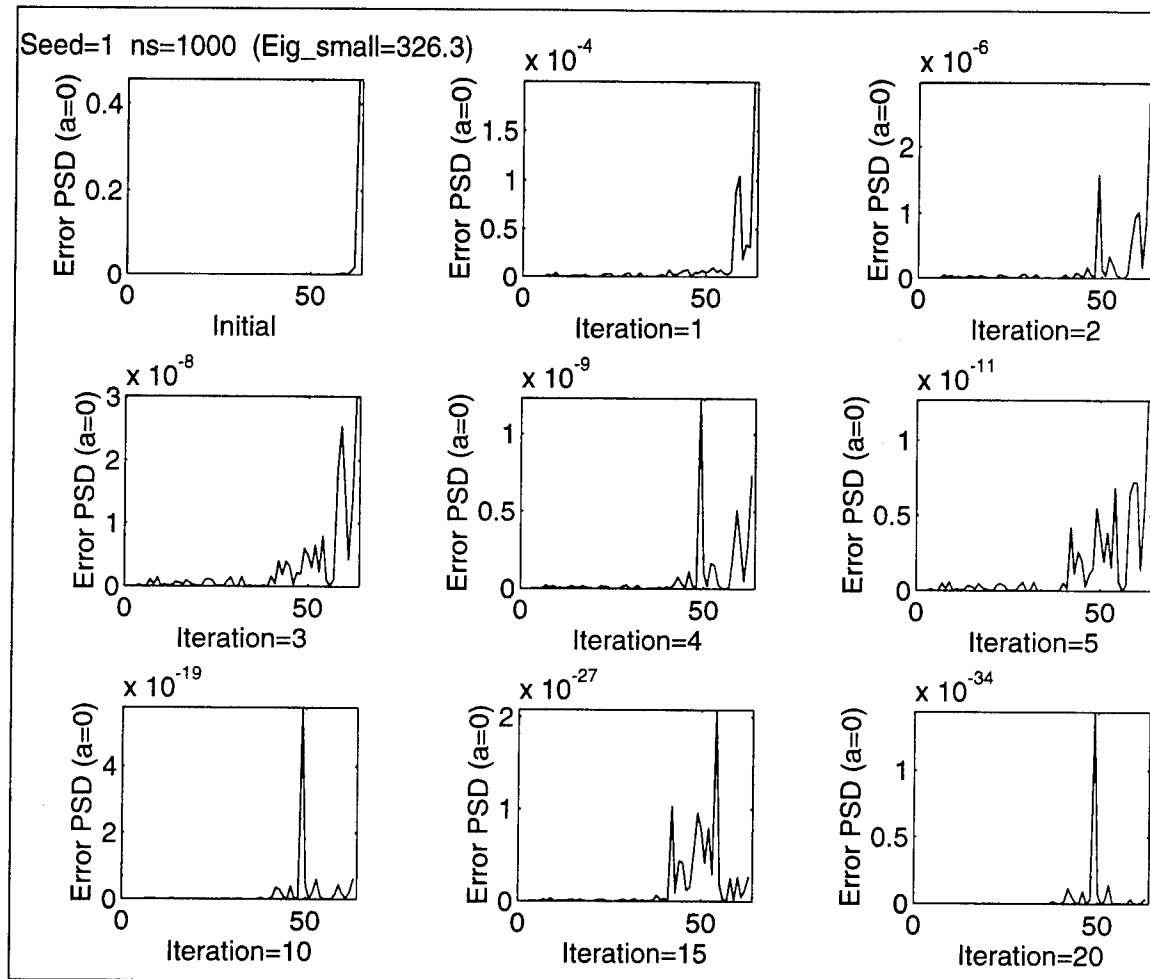


Figure 4.23: PSD of Error for 1000 Point Signal, Smallest Eigenvalue

Figure 4.24 shows the PSDs for the case when the middle eigenvector is used as the initial guess for the Toeplitz approximation algorithm. Several low and medium frequency components in the error are eliminated but a high frequency component is introduced.

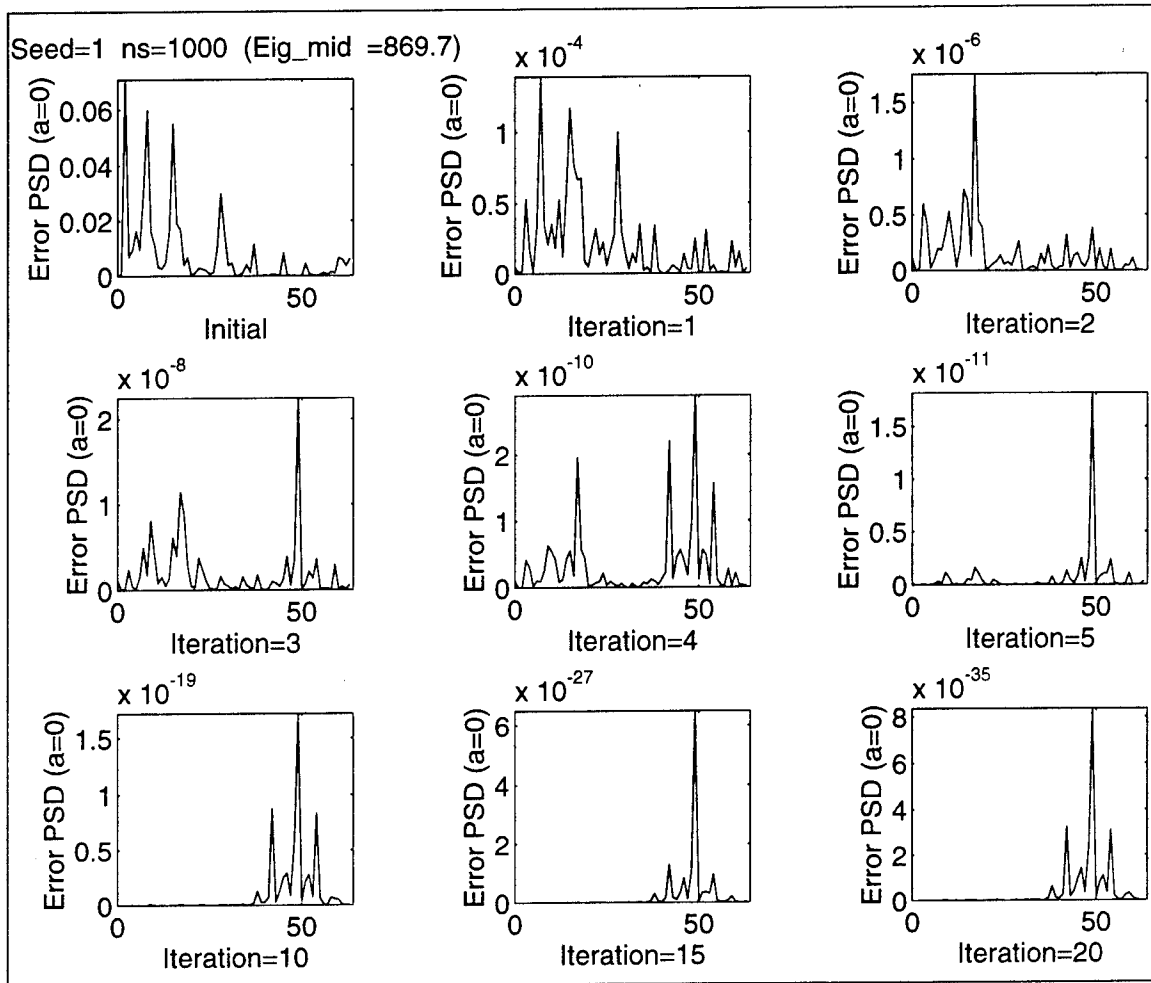


Figure 4.24: PSD of Error for 1000 Point Signal, Middle Eigenvalue

Figure 4.25 shows the PSDs of the error when the eigenvector of the largest eigenvalue is used as the initial guess. The initial error consists of a single low frequency mode that is eliminated while the same high frequency error component that appeared in the previous two cases is introduced by the algorithm.

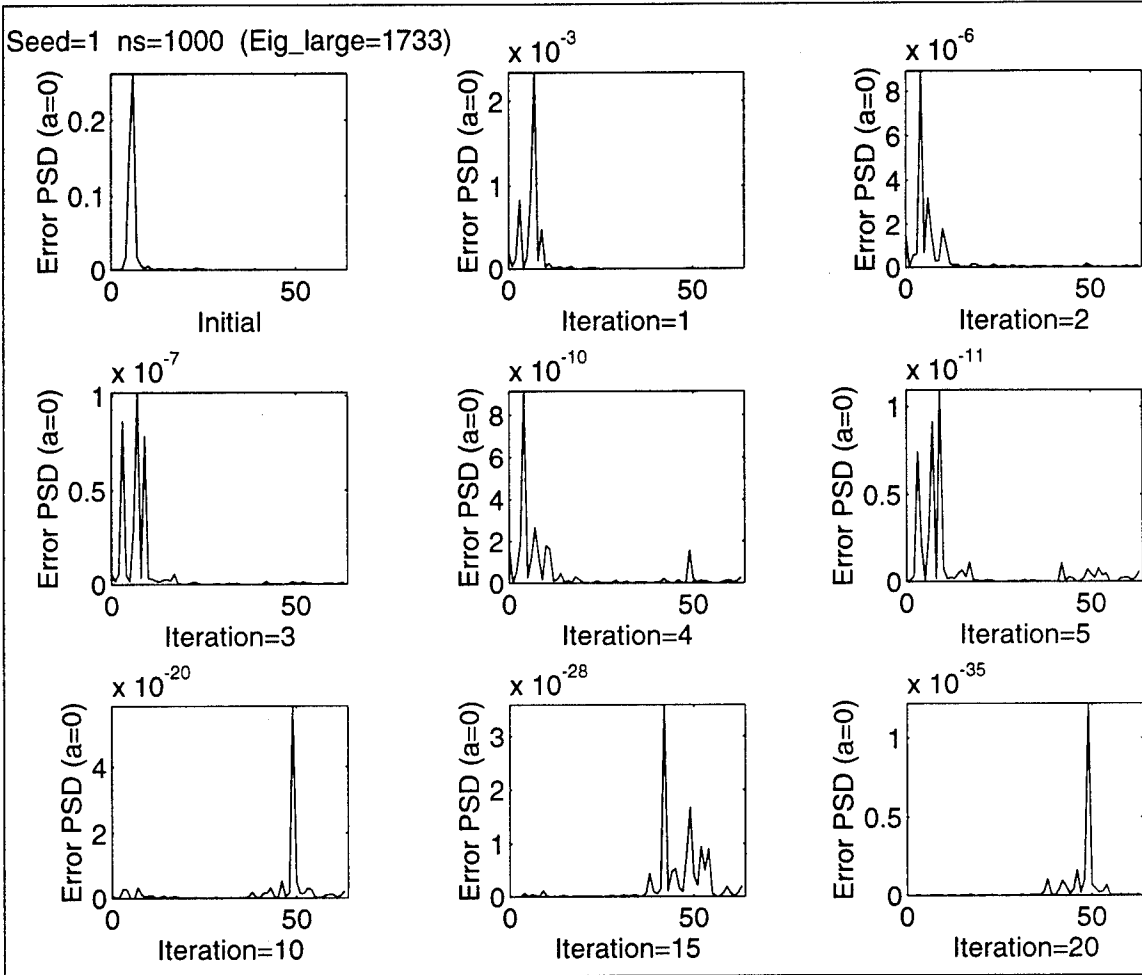


Figure 4.25: PSD of Error for 1000 Point Signal, High Eigenvalue

The results of this experiment demonstrate that the properties of the Toeplitz approximation algorithm are much different from those of many other relaxation methods, such as the Jacobi method. The Toeplitz approximation algorithm does not appear to possess the smoothing property that multigrid exploits so well. On the contrary, the Toeplitz approximation algorithm removed the low and medium frequency modes of the error, but not the high frequency modes for most initial guesses. In fact, a high frequency

error was introduced or excited in most of the test cases. Table 4.2 compares the frequency spectrums of the error for the various cases.

The choice of eigenvector for the initial guess provided different results for each of the three input signals. For the 253 point input signal, the eigenvector of the largest eigenvalue provided the worst initial guess while the eigenvector of the smallest eigenvalue provided the best initial guess. The eigenvector of the smallest eigenvalue also provided the best initial guess for the 350 point input signal case but the eigenvector of the middle eigenvalue provided the worst initial guess. The three initial guesses for the

Table 4.2: Comparison of Error PSDs for Zero Solution

Points in Input Signal	Error Frequency Components	Initial guess is eigenvector of:		
		Smallest Eigenvalue	Middle Eigenvalue	Largest Eigenvalue
253	Initial	low medium high	high	high
	After Toeplitz	low medium high	high	low medium high
350	Initial	high	low high	medium
	After Toeplitz	high	high	high
1000	Initial	high	low medium	high
	After Toeplitz	high	high	low high

1000 point input signal case provided nearly identical results with the eigenvector of the largest eigenvalue providing the best initial guess by a slight margin.

The factor which appeared to most greatly affect the convergence of the Toeplitz approximation algorithm was the length of the input signal used to create the Toeplitz operator matrix. The 1000 point input signal produced a well-conditioned Toeplitz operator matrix that allowed the algorithm to converge for all three initial guesses. The 253 point input signal caused the algorithm to diverge for all three initial guesses while the 350 point input signal cases converged for a few iterations and then diverged.

The results of the three test cases indicate that the Toeplitz approximation algorithm is most effective when the input signal length is as large as possible. Which eigenvector of the operator matrix that is used as the initial guess has little effect on the rate of convergence of the algorithm. Also, the algorithm appears to have the opposite effect of the smoothing property that is common to many relaxation methods. The results do not offer any indications that multigrid could significantly improve the convergence of Toeplitz approximation algorithm.

2. Solving the Weiner-Hopf Equation

The previous experiments were designed to analyze some of the properties of the Toeplitz approximation algorithm and its iteration matrix to determine the empirical implications of applying multigrid with the Toeplitz approximation algorithm. Actually applying the algorithm to a particular problem provides further insight about the effectiveness of the algorithm. Therefore, an experiment similar to those run on the

Toeplitz approximation algorithm for a zero solution was conducted to examine the usefulness of the Toeplitz approximation algorithm in modeling an IIR system with a FIR model.

The experiment involved the creation of a 22^{nd} order elliptic IIR bandpass filter system using the MATLAB filter generation function, The IIR filter was then modeled with a 126^{th} order FIR filter by solving the Wiener-Hopf equation

$$\mathbf{R}\mathbf{b} = \mathbf{r} \quad (4.4)$$

with the Toeplitz approximation algorithm to determine the FIR filter coefficients $\mathbf{b}$.

White noise signals of length 253, 350, and 1000 points were input to the 22^{nd} order IIR filter to determine the actual output of the system. The 127×127 correlation matrix $\mathbf{R}$, its corresponding Toeplitz matrix $\mathbf{T}$, and the 127×1 cross-correlation vector $\mathbf{r}$ were then computed and used as inputs to the Toeplitz approximation algorithm. The initial guess for the Toeplitz approximation algorithm was chosen as

$$\mathbf{b}_0 = \mathbf{T}^{-1}\mathbf{r}. \quad (4.5)$$

The actual solution of the optimal FIR filter coefficients was assumed to be

$$\mathbf{b} = \mathbf{R}^{-1}\mathbf{r} \quad (4.6)$$

where the inverse of the correlation matrix was obtained from the MATLAB 'inv' function. This led to an initial error vector of

$$\mathbf{e}_0 = \mathbf{R}^{-1}\mathbf{r} - \mathbf{T}^{-1}\mathbf{r}. \quad (4.7)$$

The Toeplitz approximation algorithm was run for 20 iterations and the error norm of the coefficient vector

$$e(k) = \|\mathbf{e}_k\| = \|\mathbf{R}^{-1}\mathbf{r} - \hat{\mathbf{b}}_k\| \quad (4.8)$$

was computed after each iteration, where $\hat{\mathbf{b}}_k$ is the coefficient vector estimate at the k^{th} iteration of the Toeplitz approximation algorithm. As in the previous experiment, PSDs of the error vector were computed after the first five iterations and the 10^{th} , 15^{th} , and 20^{th} iterations of the Toeplitz approximation algorithm. Plots of the error norm versus iteration number and plots of the error vector PSDs after selected iterations are shown for a random number generator seed of I .

Figure 4.26 shows a plot of error norm versus iteration number when trying to solve the Wiener-Hopf equation with the Toeplitz approximation algorithm and a 253 point input signal. Note that the algorithm steadily diverges from the onset for this minimum length input signal case. This result concurs with the findings from the previous experiment for the 253 point input signal, which also showed poor performance of the Toeplitz approximation algorithm.

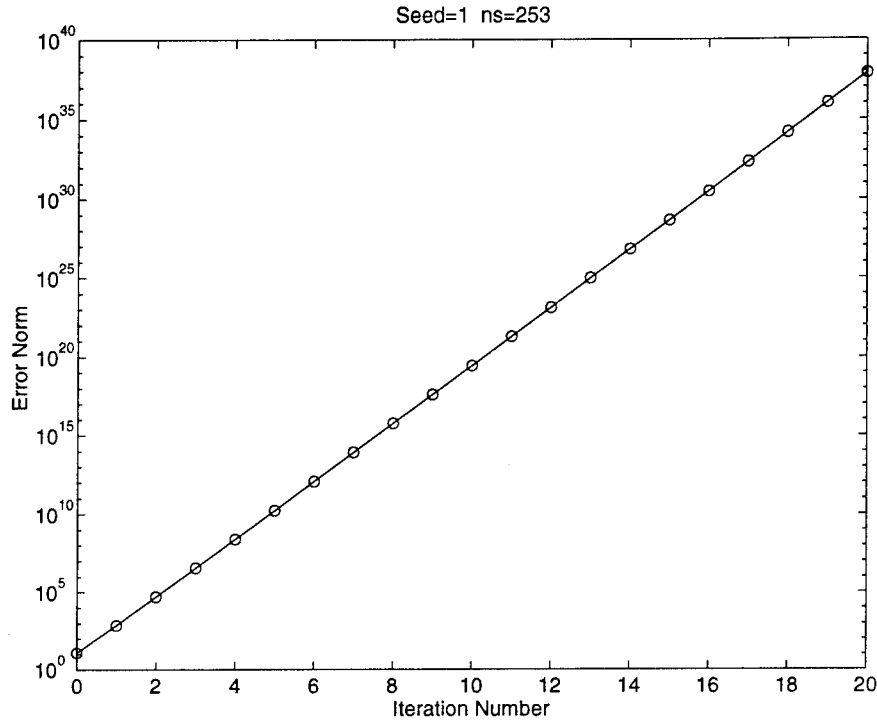


Figure 4.26: Wiener-Hopf Solution Error Norms for 253 Point Signal

Figure 4.27 shows the PSDs of the error after various iterations of the Toeplitz approximation algorithm for the 253 point input signal. The spectral content of the error appeared to be unaffected by iterations of the Toeplitz approximation algorithm, with the error simply increasing. The error was dominated by a narrow band high frequency component with low and medium frequency components also present. The Toeplitz approximation algorithm is virtually ineffective for the 253 point input signal case.

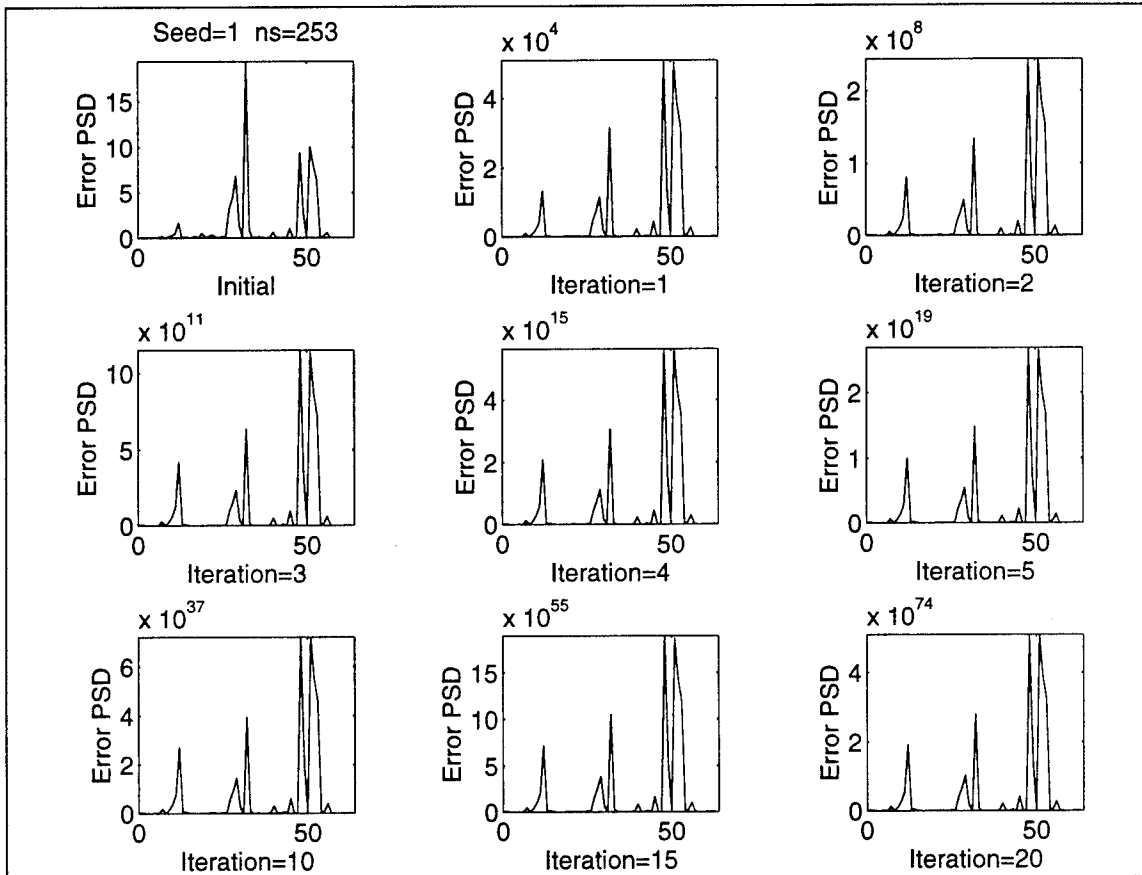


Figure 4.27: Weiner-Hopf Solution PSD of Error for 253 Point Signal

Figure 4.28 contains the error norm versus iteration number plot when applying the 350 point input signal to the Toeplitz approximation algorithm. The algorithm converged slightly for the first two iterations and then began to slowly diverge. The algorithm diverged much slower for the 350 point input signal than for the 253 point input signal case.

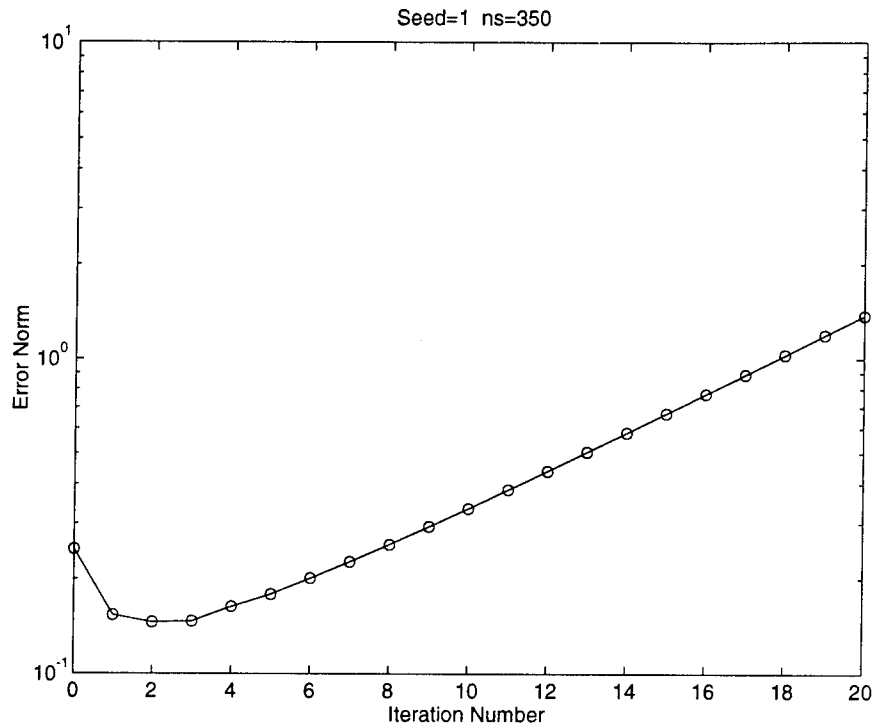


Figure 4.28: Weiner-Hopf Solution Error Norms for 350 Point Signal

Figure 4.29 contains the PSDs of the error after iterations of the Toeplitz approximation algorithm for the 350 point input signal. The initial error, which contained wide band medium frequency components, was reduced to a narrow band high frequency component by the algorithm. The behavior of the Toeplitz approximation algorithm to reduce errors to a high frequency component was also seen in previous experiments for input signals of varying lengths.

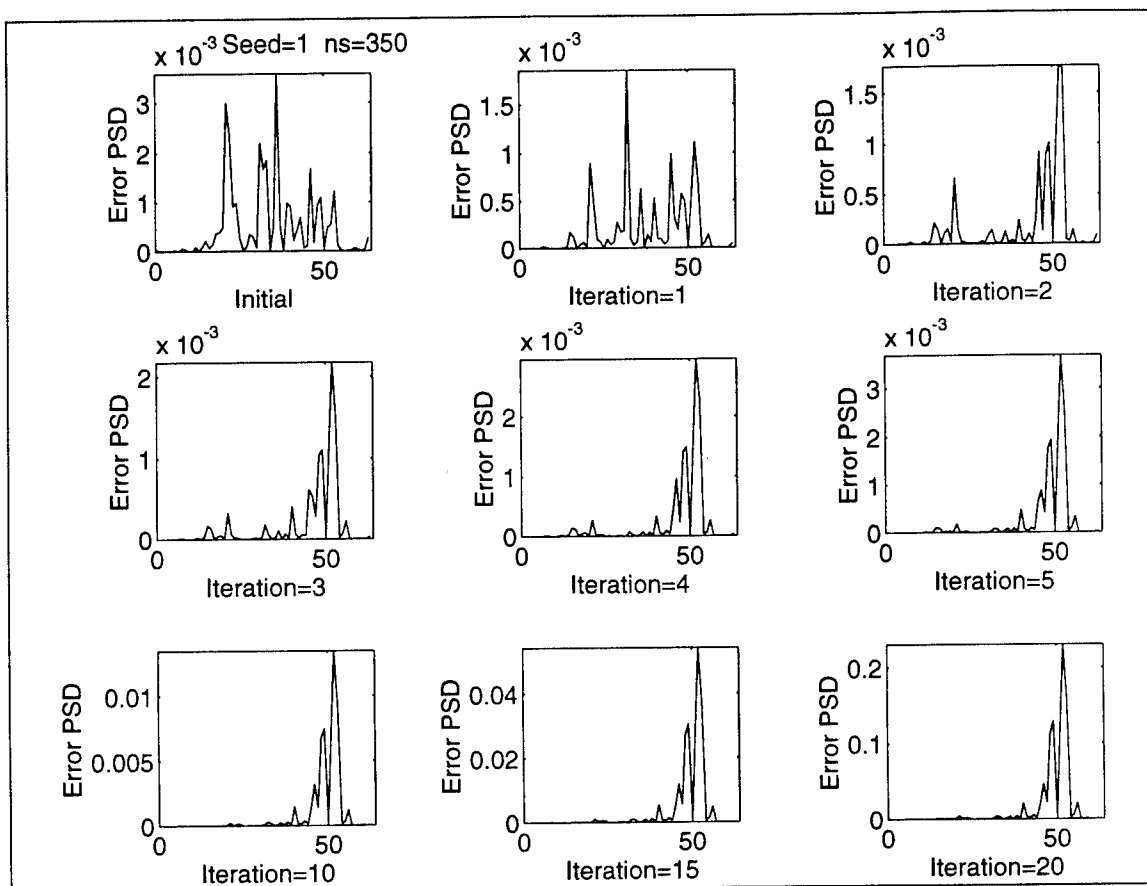


Figure 4.29: Wiener-Hopf Solution PSD of Error for 350 Point Signal

Figure 4.30 shows the error norm versus iteration number plot when applying the 1000 point input signal to the Toeplitz approximation algorithm. As expected, the 1000 point input signal provides the best solution of the Wiener-Hopf equation with a steady convergence of the Toeplitz approximation algorithm for 17 iterations before the error norm leveled off.

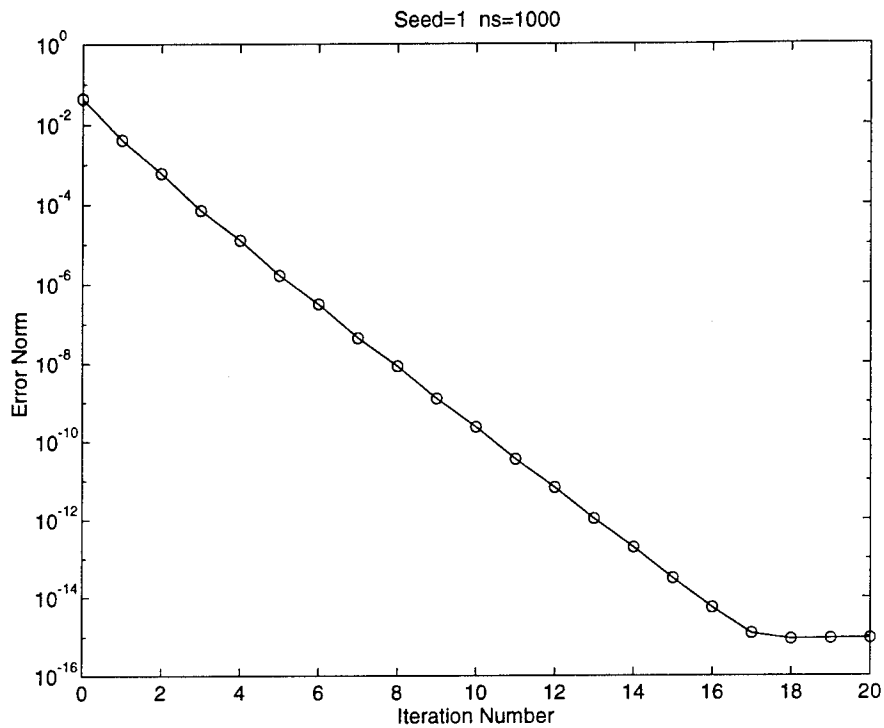


Figure 4.30: Weiner-Hopf Solution Error Norms for 1000 Point Signal

Figure 4.31 contains the PSDs of the error after various iterations of the Toeplitz approximation algorithm for the 1000 point input signal. Unlike the 253 and 350 point cases, the 1000 point input signal case resulted in a continuous decrease in the error. However, the treatment of the frequency components of the error appears to be much different compared to the previous results. The amplitude of the high frequency error component was reduced after each iteration, but the error after 20 iterations contained both high and low frequency components.

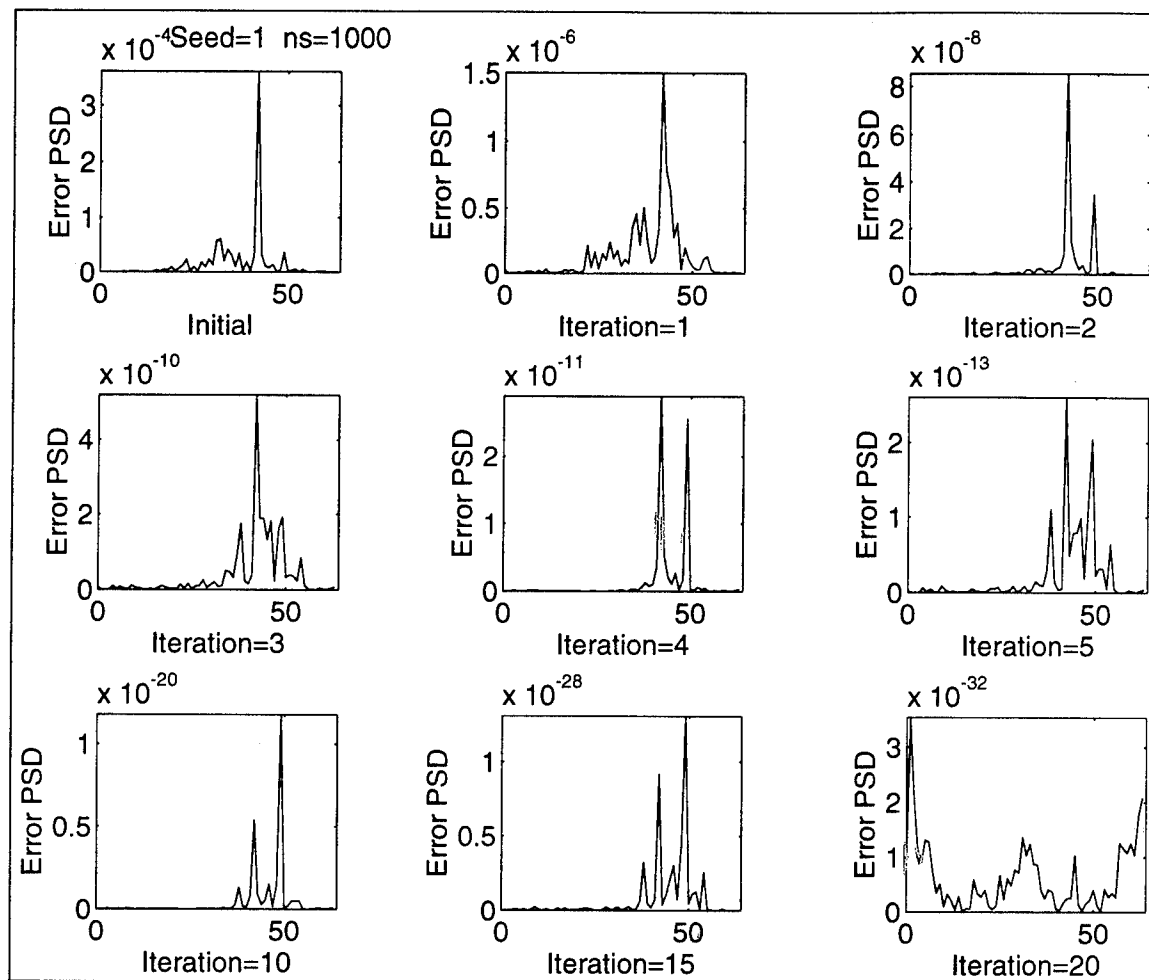


Figure 4.31: Wiener-Hopf Solution PSD of Error for 1000 Point Signal

As with the experiments in the previous section, the Toeplitz approximation algorithm appears to be most effective for solving the Wiener-Hopf equation when a larger sample of the input signal is used to generate the Toeplitz iteration matrix. Also, as shown in the previous experiments, the Toeplitz approximation algorithm seems to have trouble eliminating the high frequency components of the error. The empirical

analysis does not seem to support the concept of using the Toeplitz approximation algorithm with multigrid techniques.

D. MULTIGRID ANALYSIS

Although theoretical analysis does not seem to support the concept of using the Toeplitz approximation algorithm with multigrid to solve the Weiner-Hopf equation, computer simulations can be used to easily and inexpensively test the full effectiveness of multigrid on the system modeling problem. Previous research has used this approach to demonstrate that multigrid provides some benefit with the Toeplitz approximation algorithm in modeling even order FIR filters. Experiments described in this section were conducted to determine if nested multigrid techniques can provide a better initial guess for the Toeplitz approximation algorithm. Other experiments were run to determine the utility of various multigrid restriction operators on the system modeling problem. Multigrid techniques were applied with the Toeplitz approximation algorithm and the results were compared to those from direct matrix inversion and from running the Toeplitz approximation algorithm without multigrid. These experiments were designed to fully test the practicality of applying the Toeplitz approximation algorithm with multigrid to the system modeling problem.

1. Nested Multigrid Evaluation

One way to improve the convergence of iterative algorithms is to start with a better initial guess. Experiments were conducted to determine if nested multigrid

techniques could be employed to obtain a better initial guess for the Toeplitz approximation algorithm. The previous section described experiments applying the Toeplitz approximation algorithm to solve the Weiner-Hopf equation for a 126^{th} order FIR filter. These experiments were repeated using an initial guess obtained from nested multigrid techniques rather than the initial guess in (4.5), to determine if the Toeplitz approximation algorithm can benefit from nested multigrid.

In the experiment, initial guesses were obtained by a recursive nested multigrid Toeplitz approximation algorithm which transferred the 127×127 correlation matrix to coarser grids, using the full weighting multigrid operator, until a single element resulted on a 1×1 grid. The Weiner-Hopf equation was then solved on successive grids of 1×1 , 3×3 , 7×7 , 15×15 , 31×31 , and 63×63 using the Toeplitz approximation algorithm. The linear interpolation operator was used to transfer the coefficient vector computed at each coarse grid back to the next finer grid. The transferred coefficient vector then became the initial guess for the Toeplitz approximation algorithm on the finer grid. The coefficient vector computed on the 63×63 grid was transferred to the finest grid and became the initial guess for the Toeplitz approximation algorithm. As with previous experiments, the algorithm was run for 20 iterations using input signals of length 253, 350, and 1000 points.

The results of using nested multigrid techniques to obtain a better initial guess for the Toeplitz approximation algorithm are shown for signals created with a random number generator seed of 1. Figure 4.32 shows a plot of error norm versus iteration

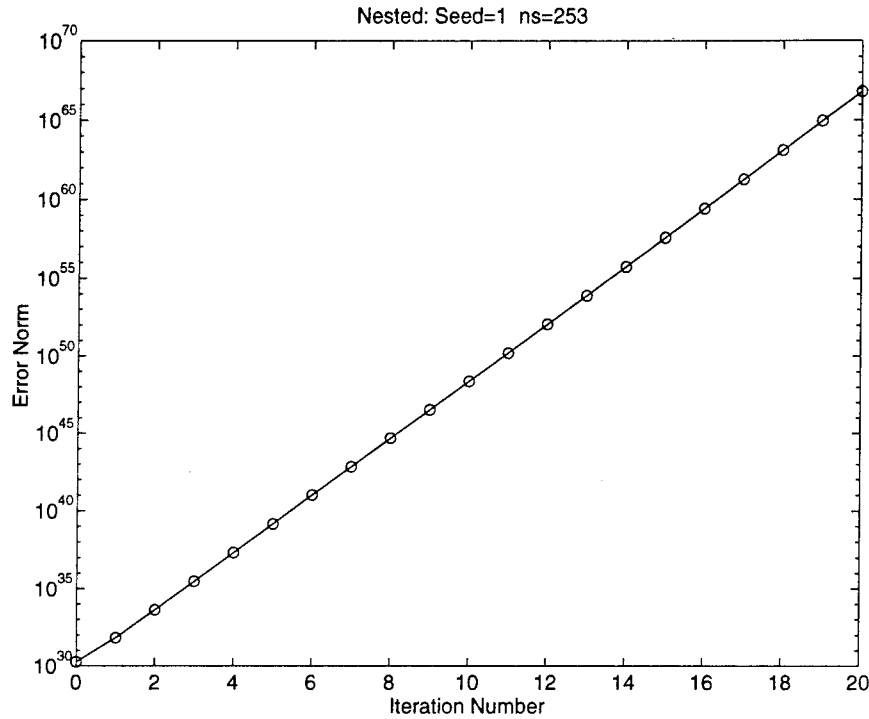


Figure 4.32: Nested Multigrid Error Norms for 253 Point Signal

number for the 253 point input signal case. Note that the initial guess produced by nested multigrid techniques was extremely poor. In fact, it took 16 iterations of the Toeplitz approximation algorithm with an initial guess of $T^1 r$ to diverge to the same error as the initial guess using nested multigrid.

Figure 4.33 shows plots of the PSDs of the error for the 253 point input signal case. The initial guess resulted in an error with a low frequency component, but the Toeplitz approximation algorithm changed the frequency spectrum of the error to a dominant high frequency error component plus a low and medium frequency component.

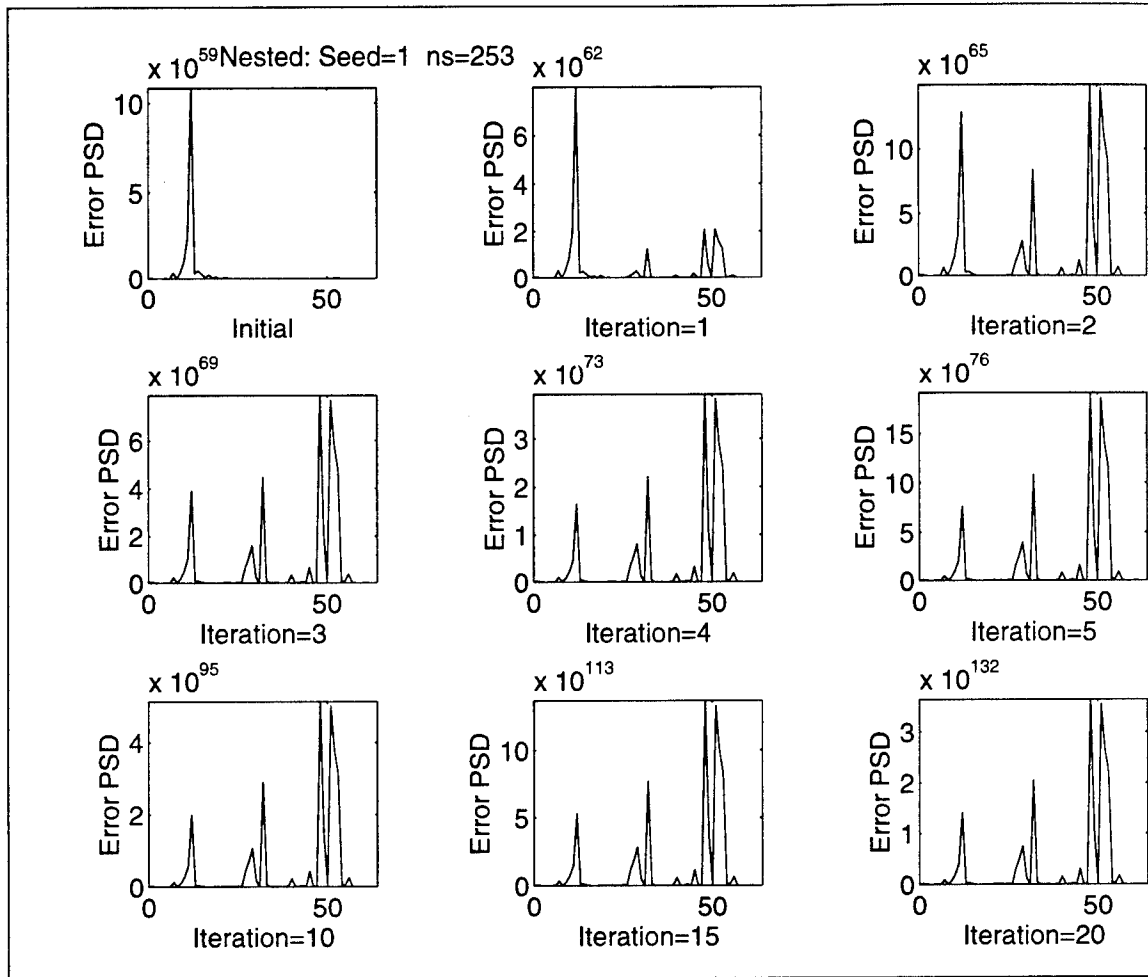


Figure 4.33: Nested Multigrid PSD of Error for 253 Point Signal

Figure 4.34 shows a plot of error norm versus iteration number for the 350 point input signal. In this case, the initial guess was almost as good as the initial guess of $T^I r$ from the experiments in the previous section. However, the Toeplitz approximation algorithm only converged for three iterations before slowly diverging.

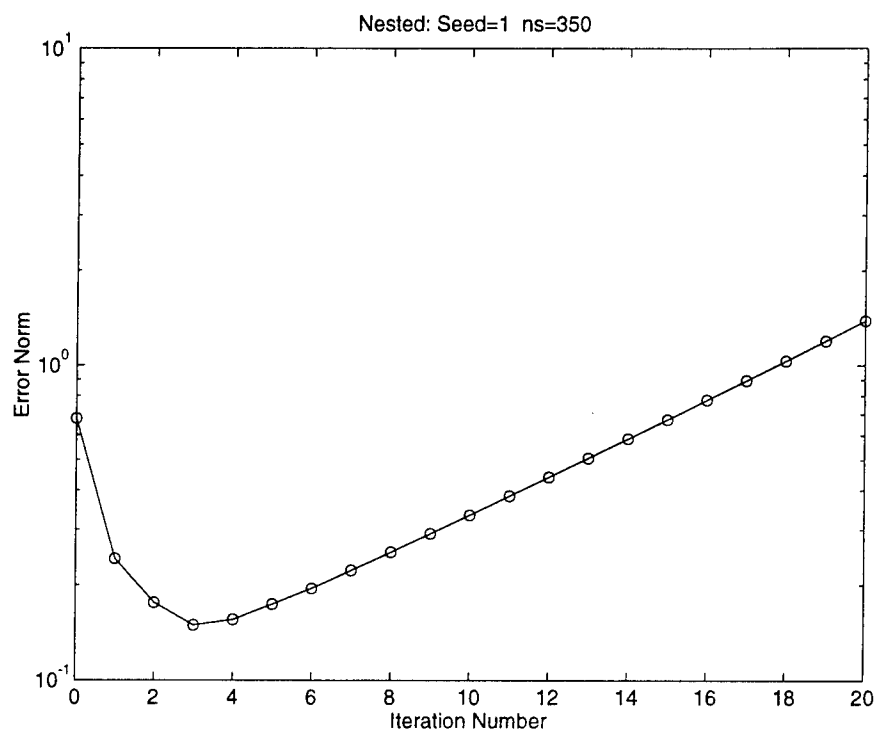


Figure 4.34: Nested Multigrid Error Norms for 350 Point Signal

Figure 4.35 shows plots of the PSDs of the error for the 350 point input signal case. The initial guess obtained from nested multigrid resulted in a wide band, medium frequency error. The Toeplitz approximation algorithm again reduced the error to a high frequency component as seen in many of the experiments that were conducted.

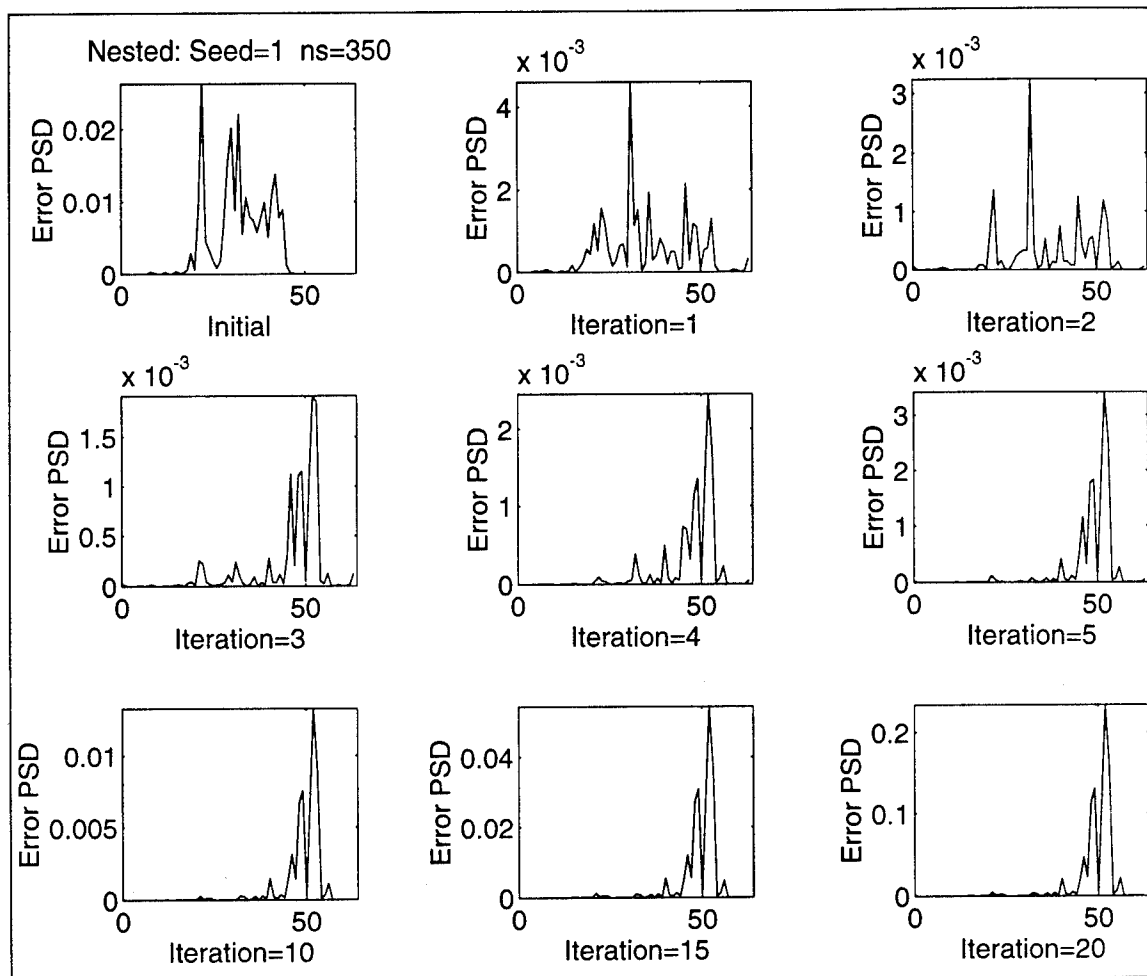


Figure 4.35: Nested Multigrid PSD of Error for 350 Point Signal

Figure 4.36 shows a plot of error norm versus iteration number for the *1000* point input signal. The initial guess was again almost as good as the initial guess of $T^I r$ used in previous experiments. In fact, the convergence of the Toeplitz approximation algorithm for the *1000* point input signal case with nested multigrid appears to be just one iteration behind the convergence without nested multigrid.

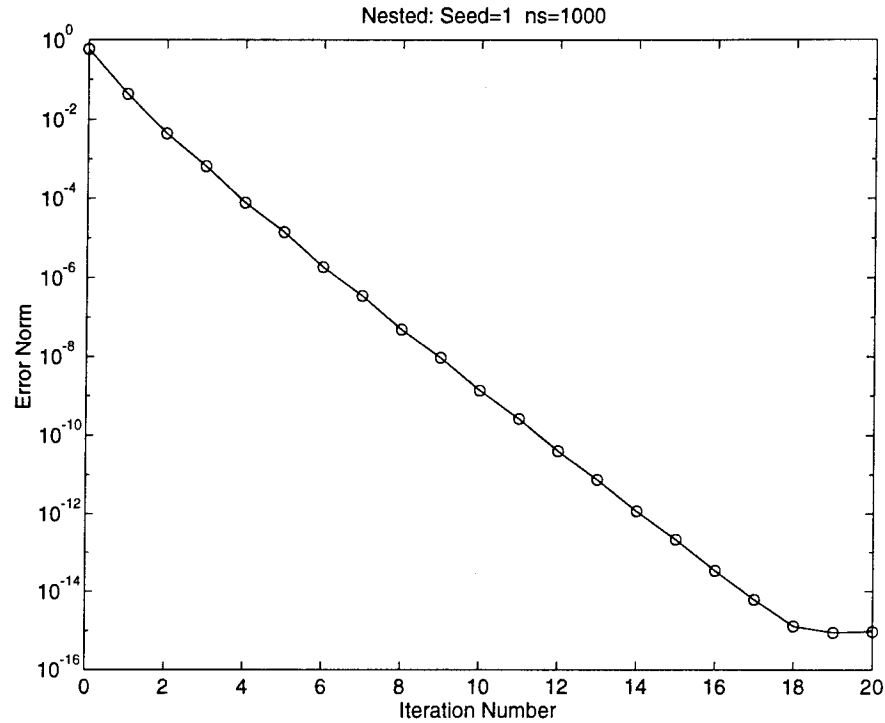


Figure 4.36: Nested Multigrid Error Norms for 1000 Point Signal

Figure 4.37 shows plots of the PSDs of the error for the *1000* point input signal case. As with the *350* point input signal case, the initial guess from nested multigrid produced a wide band, medium frequency error. The error was first reduced to a high frequency error component for several iterations and then resulted in the same strange high and low frequency components seen in the previous section for the *1000* point input signal case.

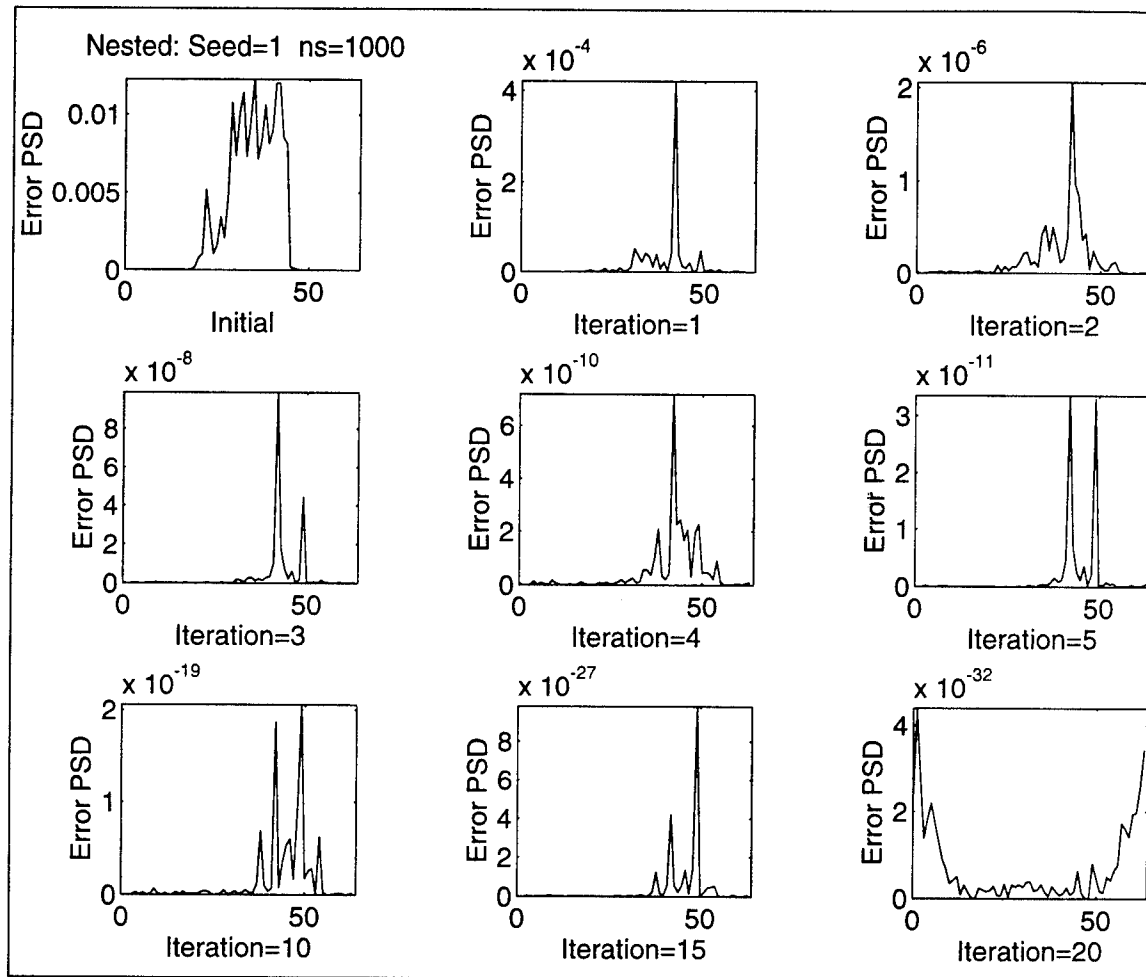


Figure 4.37: Nested Multigrid PSD of Error for 1000 Point Signal

The experiments applying nested multigrid techniques in an attempt to obtain a better initial guess for the Toeplitz approximation algorithm did not offer very promising results. The initial guesses provided by the nested multigrid techniques were not as good as the initial guess of $T^I r$ used in previous experiments. The convergence or divergence of the Toeplitz approximation algorithm seemed to be the same for the nested multigrid initial guesses as they were for the $T^I r$ initial guess. The algorithm converged to a

solution of the Weiner-Hopf equation for the 1000 point input signal case, diverged for the 253 point input signal, and converged for 3 iterations of the 350 point input signal case before diverging. The PSDs of the error also showed a tendency for the Toeplitz approximation algorithm with nested multigrid initial guesses to reduce the error frequency spectrum to a high frequency component. Overall, there appeared to be no clear advantage of using the nested multigrid techniques to obtain an initial guess for the Toeplitz approximation algorithm rather than using an initial guess of $\mathbf{T}^{-1}\mathbf{r}$.

2. Applying Multigrid for Even Order FIR Filters

Previous research has demonstrated that multigrid techniques can be applied with the Toeplitz approximation algorithm to solve the Weiner-Hopf equation for optimal FIR filters of even order. To obtain a clear picture of how well multigrid works for modeling even order filters, computer simulations were developed to run multigrid techniques with the Toeplitz approximation algorithm by applying different cycling schemes and restriction operators. Both the multigrid V-cycle and FMV-cycle schemes were examined using the full weighting and injection restriction operators with the linear interpolation prolongation operator. Direct matrix inversion and the Toeplitz approximation algorithm without multigrid were also used to solve the Weiner-Hopf equation to establish a basis for comparison to the multigrid techniques. This experiment was designed to determine if there is any benefit to using multigrid with the Toeplitz approximation algorithm on the system modeling problem for even order FIR filters.

The experiments attempted to model the same 22nd order elliptic IIR bandpass system that was modeled in previous experiments. The system was modeled with a 126th order FIR filter by using various algorithms to determine the filter coefficients. The same white noise input signals of length 253, 350, and 1000 points that were used in several of the previous simulations were used to compute the correlation matrix $\mathbf{R}$ and the cross correlation vector $\mathbf{r}$. The model FIR filter coefficients were then obtained by solving the Wiener-Hopf equation using direct matrix inversion, the Toeplitz approximation algorithm, multigrid V-cycle, and multigrid FMV-cycle techniques. The 255, 350, and 1000 point signals were input to both the IIR system and the FIR model to obtain the actual system and estimated model output signals. The norm of the error between the two output signals

$$e = \|\mathbf{y} - \hat{\mathbf{y}}\| \quad (4.9)$$

was computed after each iteration of the selected algorithm to determine how well the output of the FIR filter model matches the output of the IIR system. Note that the error norms computed in this experiment are different from the coefficient vector error norms described in previous experiments.

Direct matrix inversion was used as the benchmark for determining how well the various algorithms performed in determining the coefficients of the FIR model. Solution by direct matrix inversion was obtained by inverting the correlation matrix using the MATLAB 'inv' function and then solving

$$\hat{\mathbf{b}} = \mathbf{R}^{-1}\mathbf{r} \quad (4.10)$$

for the optimal FIR filter coefficients. The Toeplitz approximation algorithm was then run without multigrid to provide a comparison for the test cases with multigrid techniques. Multigrid with the Toeplitz approximation algorithm was run using the multigrid V-cycle with full weighting and injection restriction operators and the multigrid FMV-cycle with the same operators. The results, shown in Table 4.3 for a random number generator seed of 1, indicated that direct matrix inversion provided the best solution for all three input signals.

The error norm for the direct matrix inversion method was significantly less than the error norms for each of the other methods in the 253 point input signal case. The other five test cases all diverged from the first iteration, with the Toeplitz approximation algorithm having the next lowest error norm after one iteration (21 times the error norm of the direct matrix inversion). Figure 4.38 shows the PSDs of the error after one iteration of each of the multigrid simulations for the 253 point input signal.

Table 4.3: Comparison of Multigrid Techniques for Modeling 126th Order Filter
 *** indicates methods were diverging for given number of iterations

Method	Number of Iterations	253 Points in Input Signal	350 Points in Input Signal	1000 Points in Input Signal
Direct Inversion		4.4328	0.6550	2.05850846
Toeplitz Approximation Algorithm	1	94.581	3.3370	2.4045
	2	***	1.9872	2.0623
	3	***	1.8718	2.05855131
	4	***	***	2.05851292
	5	***	***	2.05850788
Multigrid V-cycle (Full Weighting)	1	773316	1.5740	2.0612
	2	***	1.5751	2.05851166
	3	***	1.9876	2.05850857
	4	***	***	2.05850846
Multigrid V-cycle (Injection)	1	17918	1.4464	2.0611
	2	***	1.3892	2.05851075
	3	***	1.7160	2.05850857
	4	***	***	2.05850846
Multigrid FMV-cycle (Full Weighting)	1	444073079	1.5941	2.0617
	2	***	1.6893	2.05851165
	3	***	2.1666	2.05850856
	4	***	***	2.05850846
Multigrid FMV-cycle (Injection)	1	1695	1.2011	2.0607
	2	***	1.1585	2.05850999
	3	***	1.3645	2.05850854
	4	***	***	2.05850846

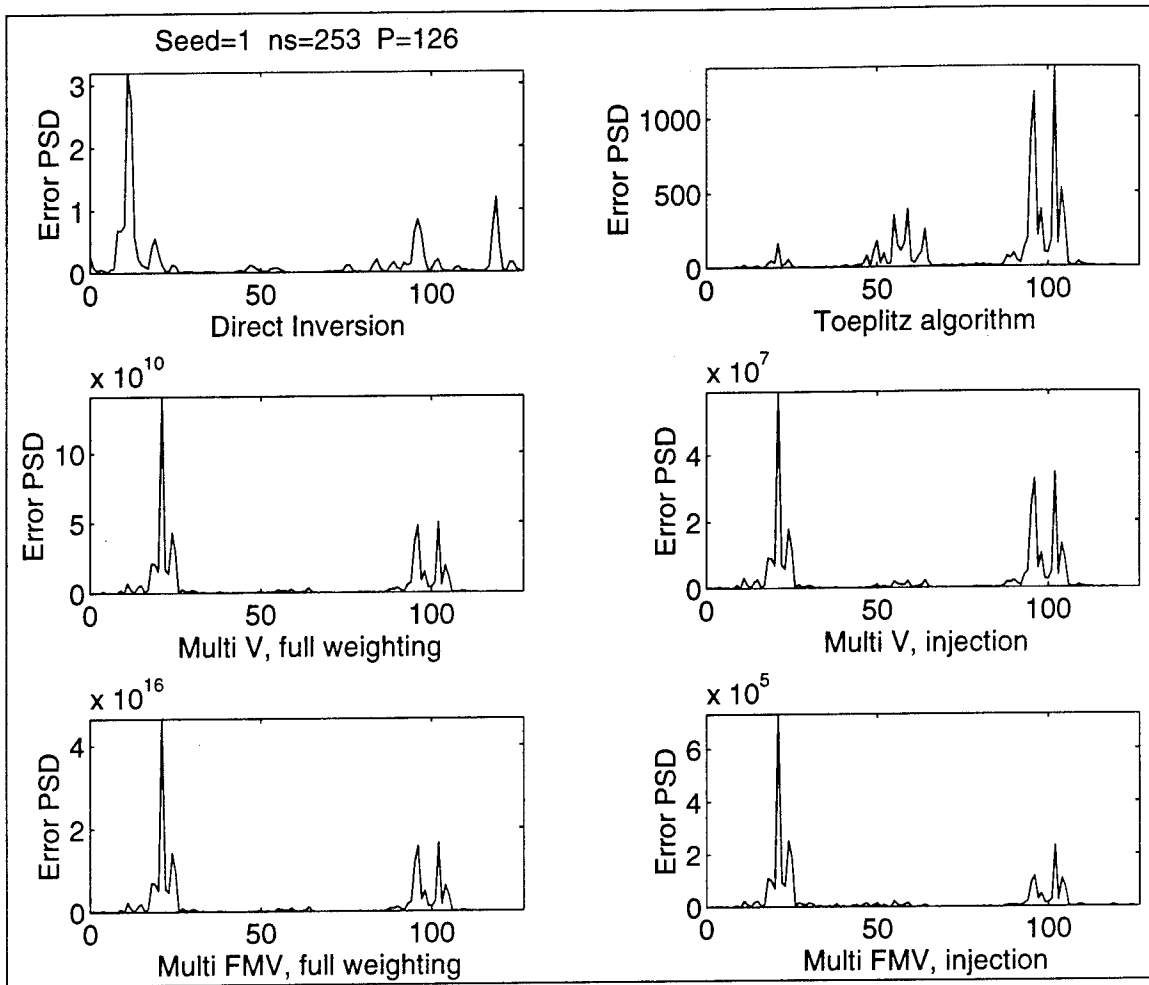


Figure 4.38: 126th Order Multigrid Solution PSD of Error for 253 Point Signal

The methods performed somewhat better for the 350 point input signal case. The direct matrix inversion method again provided the best solution to the Weiner-Hopf equation with the next best algorithm having an error norm more than 1.7 times that of direct matrix inversion. The multigrid FMV-cycle with the injection operator provided the next best solution, followed closely by the multigrid V-cycle with injection. The multigrid V-cycle and FMV-cycle techniques with the full-weighting operator performed slightly worse than the multigrid techniques using injection. The Toeplitz approximation

algorithm by itself, with an error norm of 1.6 times that of the best multigrid method, was outperformed by each of the multigrid techniques for the 350 point signal case. All the multigrid methods began to diverge after one or two iterations and the Toeplitz approximation algorithm began to diverge after three iterations. Figure 4.39 shows the PSDs of the error for the multigrid methods after one iteration for the 350 point input signal.

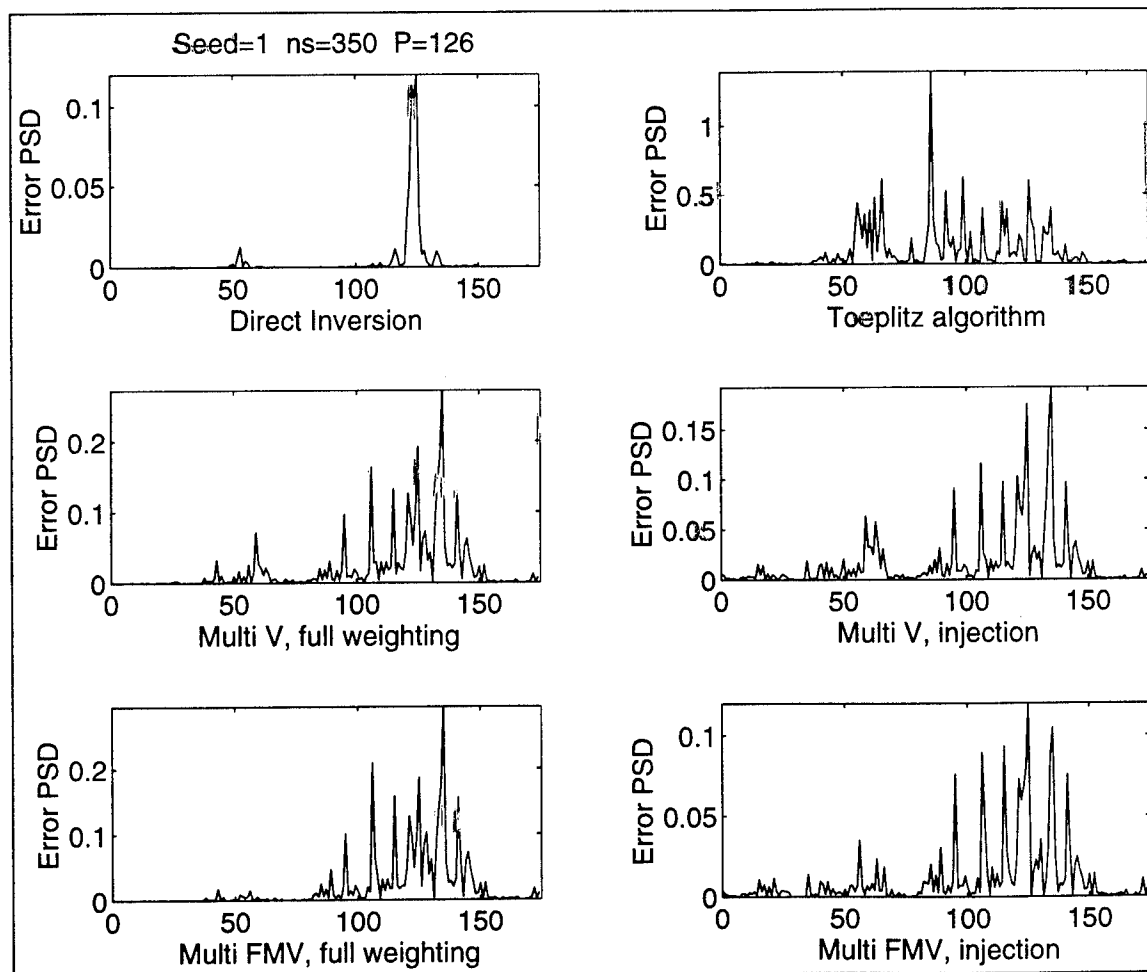


Figure 4.39: 126th Order Multigrid Solution PSD of Error for 350 Point Signal

The multigrid methods were essentially as good as direct matrix inversion for the 1000 point input signal case. The error norm after one iteration of each of the multigrid methods was within 0.003 of the error norm for the direct matrix inversion. After just four iterations, each of the multigrid methods converged to a solution with the same error norm as the direct matrix inversion method. After five iterations, the Toeplitz approximation algorithm converged to a solution with a slightly better error norm than the other methods. However, after a few more iterations, the error norm increased back to the same error norm obtained by the other methods. The multigrid methods seemed to perform equally as well, with the injection operator cases providing a slightly better solution after one iteration than the full weighting operator cases. All of the multigrid methods provided better solutions after one iteration than the Toeplitz approximation algorithm alone. Figure 4.40 shows the PSDs of the error for the multigrid methods after one iteration for the 1000 point input signal.

The multigrid methods with the Toeplitz approximation algorithm appeared to offer an improvement over the Toeplitz approximation algorithm by itself in solving the Wiener-Hopf equation for an even order FIR filter. However, as shown in previous experiments, a significant sample of the input signal was necessary for the multigrid methods to converge to a solution. The multigrid method was useless for the 253 point input signal case, somewhat beneficial for the 350 point input signal, and very effective for the 1000 point input signal. Direct matrix inversion seemed to provide better results than the other algorithms when the correlation matrix was not well conditioned, as in the

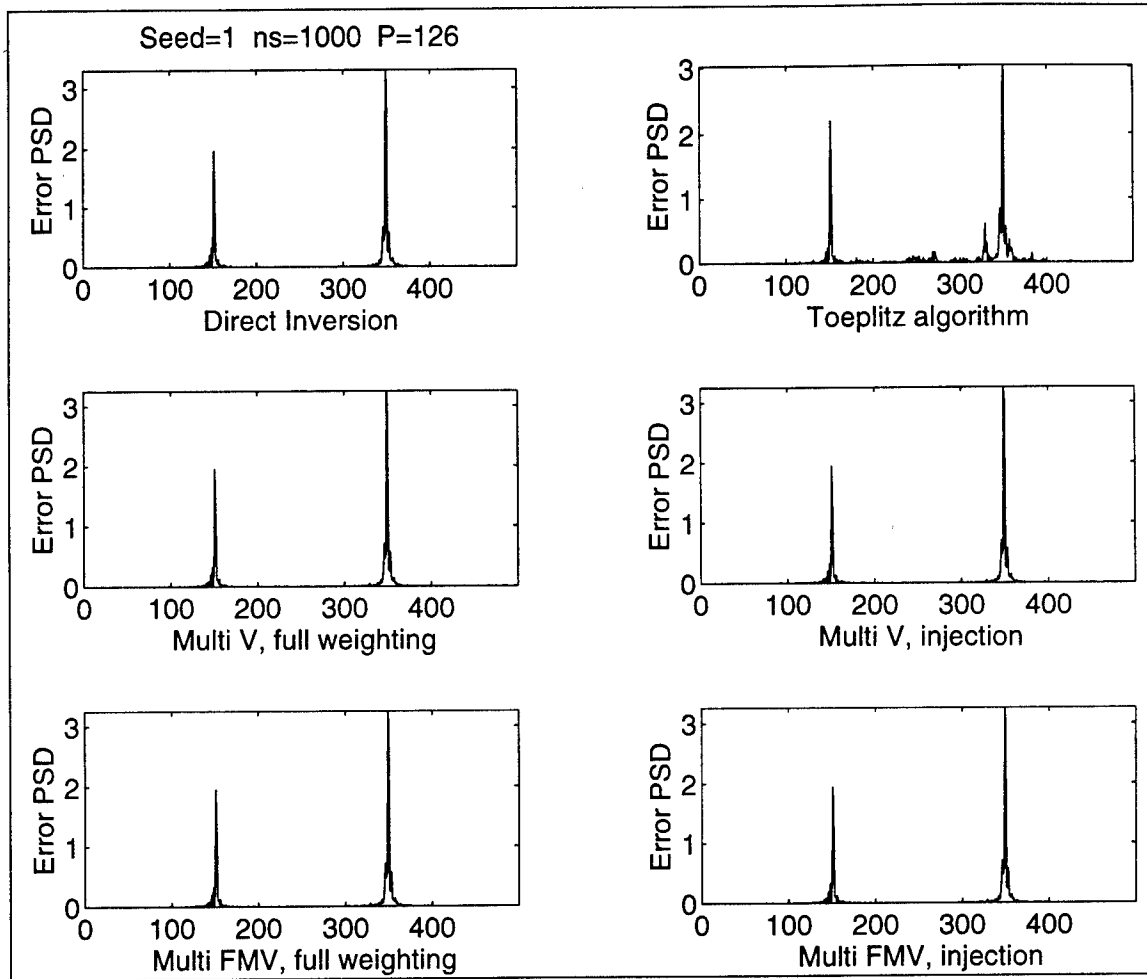


Figure 4.40: 126th Order Multigrid Solution PSD of Error for 1000 Point Signal

253 point and 350 point input signal cases. However, the multigrid methods were equally as successful as direct matrix inversion for the 1000 point input signal case, obtaining the same error norm in four iterations.

Multigrid with the injection restriction operator appeared to be somewhat better than multigrid with the full weighting operator on the even order filter problem. The multigrid V-cycle and the multigrid FMV-cycle had very similar performance

characteristics. The overall results seemed to indicate that multigrid can be used with the Toeplitz approximation algorithm to solve the Wiener-Hopf equation for an even order FIR filter, provided that a sufficient sample of the input signal is available to form the correlation matrix. Multigrid methods can enhance the performance of the Toeplitz approximation algorithm to provide results as good as those obtained through direct matrix inversion.

3. Applying Multigrid for Odd Order FIR Filters

The previous section demonstrated the use of multigrid for modeling even order FIR filters. The modeling of odd order FIR filters is another problem to which multigrid may be applied. Solving the Wiener-Hopf equation for the optimal coefficients of an odd order filter requires a restriction operator other than full weighting. The row lumping operator has been proposed as a multigrid restriction operator for modeling odd order filters. The transpose of the row lumping operator times the constant $1/2$ can be used as the multigrid prolongation operator. In order to determine how well multigrid with the row lumping operator works on the Wiener-Hopf equation, the experiments described in the previous section were repeated using a 127^{th} order FIR filter to model the 22^{nd} order IIR system. The Wiener-Hopf equation was again solved using direct matrix inversion, the Toeplitz approximation algorithm, and the multigrid V-cycle and FMV-cycle with the Toeplitz approximation algorithm and the row lumping operator. This experiment was designed to determine if multigrid techniques can be useful in solving the Wiener-Hopf equation for an odd order FIR filter.

Table 4.4 shows the results of the experiment for the four methods that were used to determine the coefficients of the optimal odd order FIR filter using a random number generator seed of 1. As with the even order filter experiments, the direct matrix inversion method provided the best solution of the Weiner-Hopf equation for the odd order FIR filter. In fact, the results for the odd order FIR filter were nearly identical to the results from the previous experiments with multigrid even order FIR filters.

Table 4.4: Comparison of Multigrid Methods for Modeling 127th Order Filter
 *** indicates methods were diverging for given number of iterations

Method	Number of Iterations	253 Points in Input Signal	350 Points in Input Signal	1000 Points in Input Signal
Direct Inversion		1.2552	0.6678	2.05737074
Toeplitz Approximation Algorithm	1	48.483	3.4897	2.3825
	2	***	2.2724	2.0609
	3	***	2.2581	2.05743397
	4	***	***	2.05737570
	5	***	***	2.05737084
Multigrid V-cycle (Row Lumping)	1	1762	1.5971	2.0595
	2	***	1.6134	2.05737279
	3	***	2.1878	2.05737083
	4	***	***	2.05737074
Multigrid FMV-cycle (Row Lumping)	1	2306	1.0133	2.0584
	2	***	1.1145	2.05736778
	3	***	1.2944	2.05737071
	4	***	***	2.05737074

For the 253 point input signal case, the direct matrix inversion method provided a solution with an error norm 38 times less than that of the Toeplitz approximation algorithm. The Toeplitz approximation algorithm, the multigrid V-cycle, and multigrid FMV-cycles diverged from the onset and were ineffective for the 253 point input signal case. Figure 4.41 shows the PSDs of the error after one iteration of each of the multigrid simulations for the 253 point input signal.

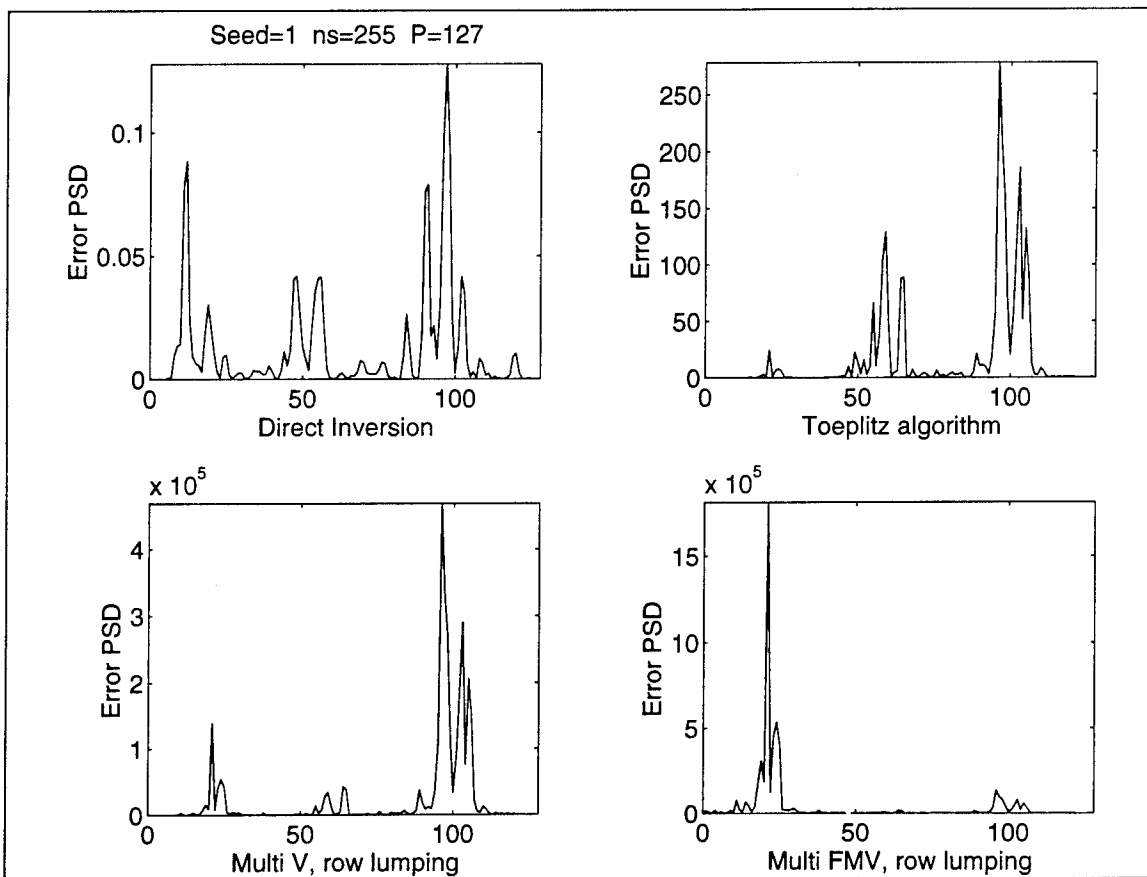


Figure 4.41: 127th Order Multigrid Solution PSD of Error for 253 Point Signal

Multigrid with the row lumping operator was more successful for the 350 point input signal case, with the multigrid FMV-cycle providing a solution with an error norm only 1.5 times that obtained from the direct matrix inversion method. The solution was 2.2 times better than the solution obtained by the Toeplitz approximation algorithm alone. The multigrid V-cycle solution was 1.4 times better than the solution obtained from the Toeplitz approximation algorithm. Both multigrid methods diverged after the first iteration, while the Toeplitz approximation algorithm diverged after three iterations. Figure 4.42 shows the PSDs of the error after one iteration of each of the multigrid simulations for the 350 point input signal.

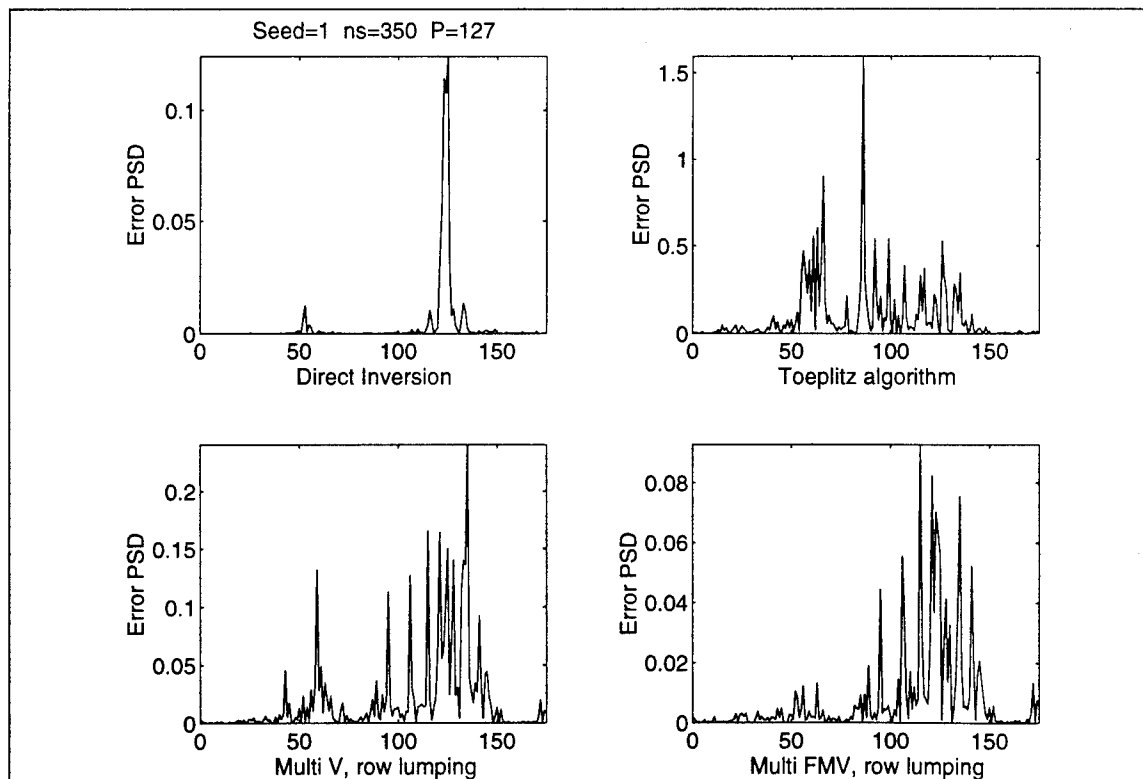


Figure 4.42: 127th Order Multigrid Solution PSD of Error for 350 Point Signal

As in the experiments modeling the even order filter, multigrid was most effective for modeling the odd order FIR filter for the *1000* point input signal case. Both multigrid methods and the Toeplitz approximation algorithm performed as well as the direct matrix inversion. The multigrid V-cycle and the multigrid FMV-cycle each converged after four iterations to a solution with the same error norm as that obtained by direct matrix inversion. The error norm of the solution after the first iteration of the multigrid FMV-cycle was only 0.001 from the direct matrix inversion error norm. The Toeplitz approximation algorithm by itself converged to the same solution after six iterations. Figure 4.43 shows the PSDs of the error after one iteration of each of the multigrid simulations for the *1000* point input signal.

The multigrid methods with the Toeplitz approximation algorithm were able to successfully solve the Wiener-Hopf equation for an odd order FIR filter. The multigrid methods did improve the Toeplitz approximation algorithm and provided solutions as good as the direct inversion method for the *1000* point input signal case. For the *350* point input signal case, multigrid was better than the Toeplitz approximation algorithm but not as good as direct matrix inversion. Multigrid and the Toeplitz approximation algorithm were useless for the *253* point input signal case. These results agreed with the results from the even order filter experiments and all the other experiments which showed that the sample size of the input signal is the most important factor in determining the effectiveness of the Toeplitz approximation algorithm in solving the Wiener-Hopf equation.

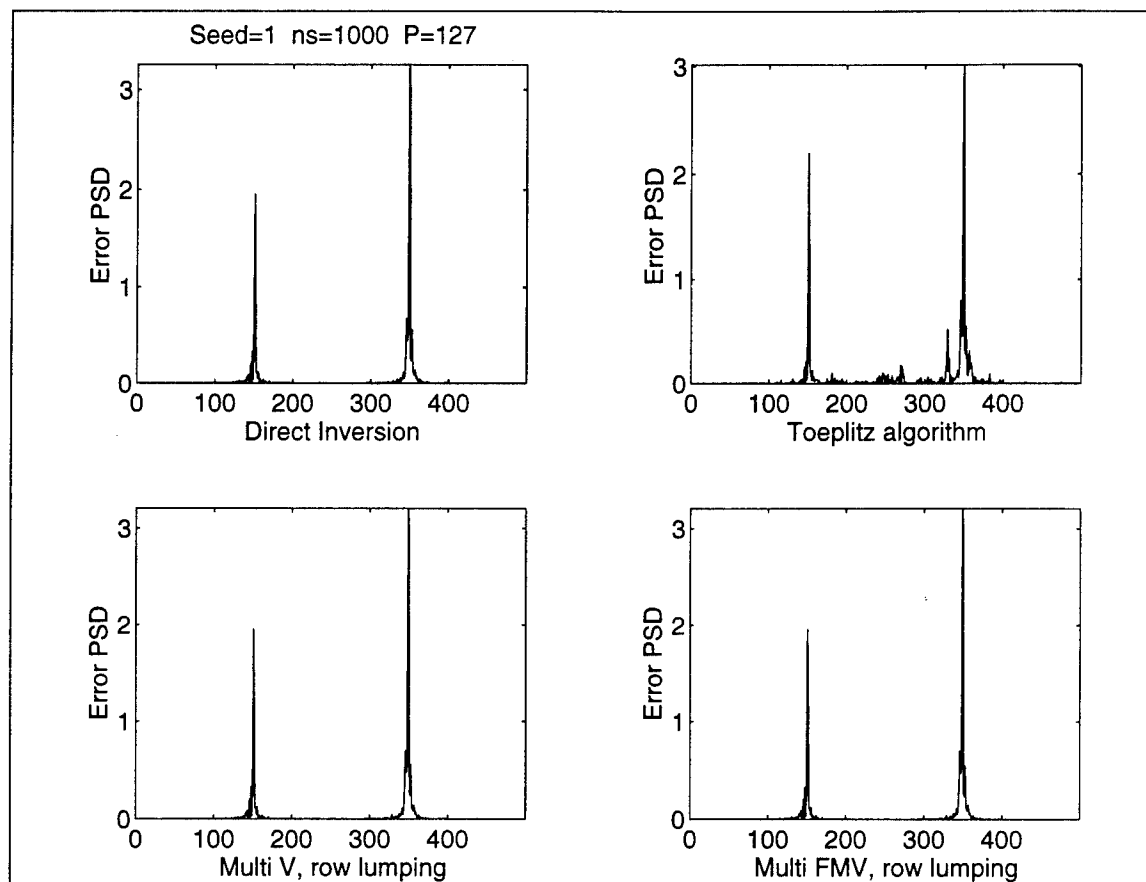


Figure 4.43: 127th Order Multigrid Solution PSD of Error for 1000 Point Signal

Multigrid with the row lumping restriction operator was shown to be effective for modeling odd order FIR filters. The choice of multigrid V-cycle or multigrid FMV-cycle did not appear to make much of a difference. Multigrid can be used to enhance the Toeplitz approximation algorithm when solving the Weiner-Hopf equation for even odd FIR filters while direct matrix inversion appears to provide as good or better results than multigrid.

V. CONCLUSIONS

A. SUMMARY OF RESULTS

This research provided several important results concerning the use of multigrid with the Toeplitz approximation algorithm for solving the Weiner-Hopf equation for optimal FIR filter coefficients. Since the properties of the Toeplitz iteration matrix did not appear to be conducive to theoretical analysis, empirical analysis was conducted to gain insight into the application of multigrid methods to system modeling problems. It was determined that the most significant factor in solving the Weiner-Hopf equation with the Toeplitz approximation algorithm and multigrid is the sequence length of the input signal used to create the correlation matrix. A sequence length of at least four times the FIR filter order ($N_s=4P$) is recommended to expect any reasonable results from the Toeplitz approximation algorithm. Applying multigrid with the Toeplitz approximation algorithm using longer sequence lengths for the input signal can lead to a good approximation of the system with just one iteration. Applying the Toeplitz approximation algorithm and multigrid using the minimum length input signal ($N_s=2P+1$) can lead to disastrous results, due to the ill-conditioned correlation matrix that is derived from the input data matrix.

The Toeplitz iteration matrix was found to lack the properties of the Jacobi iteration matrix that contribute to the success of multigrid on the model problem. The eigenvectors of the Toeplitz iteration matrix do not appear to offer any hints as to how

multigrid might help the Toeplitz approximation algorithm. Unlike the Jacobi iteration matrix, which is the same as the operator matrix for the model problem and whose eigenvectors are the Fourier modes, the eigenvectors of the Toeplitz iteration matrix contain several frequency modes with no apparent pattern between the eigenvectors corresponding to the largest and smallest eigenvalues of the Toeplitz iteration matrix.

The Toeplitz approximation algorithm does not possess the smoothing property which helps multigrid work on the model problem. On the contrary, the Toeplitz approximation algorithm seems to reduce the low and medium frequency components of the error while introducing or increasing oscillatory components of the error. Also, while the weighted Jacobi method does not mix modes, the Toeplitz approximation algorithm appears to generate new frequency components of the error between iterations. The Toeplitz approximation algorithm appeared to have the most success when the error contained predominantly smooth modes. The Toeplitz approximation algorithm appeared to possess an anti-smoothing property.

Various multigrid techniques were explored through computer simulations for this research. Nested multigrid was shown to be virtually ineffective for obtaining a better initial guess for the Toeplitz approximation algorithm in solving the Weiner-Hopf equation. Multigrid with the Toeplitz approximation was able to solve the Weiner-Hopf equation for both even and odd order FIR filters, provided that enough samples of the input signal were used to form the correlation matrix. Multigrid with both the V-cycle and FMV-cycle schemes improved the effect of the Toeplitz approximation algorithm on the system modeling problem. For even order FIR filters, the multigrid injection

operator appeared to be a somewhat better restriction operator for use with the linear interpolation prolongation operator than the full weighting operator. The row lumping operator was introduced and successfully demonstrated as a multigrid restriction operator for odd order filters. The multigrid FMV-cycle was slightly more effective than the multigrid V-cycle for both the even and odd order filter experiments. It is important to note that, while multigrid with the Toeplitz approximation algorithm works on the system modeling problem, the solutions were no better than the solutions obtained by direct matrix inversion methods.

Multigrid with the Toeplitz approximation algorithm is a possible technique for solving the Weiner-Hopf equation for both even and odd order FIR filters. Whether or not multigrid offers a distinct advantage over direct matrix inversion or relaxation methods is problem dependent. The system modeling problem and the Toeplitz approximation algorithm do not appear to possess the nice features of the multigrid model problem and the weighted Jacobi algorithm that make multigrid work so well on the multigrid model problem. However, given that the typical cost of inverting an $N \times N$ matrix with direct methods is $O(N^3)$ compared to the cost for the Toeplitz approximation algorithm of about $O(N^2)$ per sweep, the Toeplitz approximation algorithm does offer some improvement over direct methods of matrix inversion. Also, the partial solution obtained after each sweep of the Toeplitz approximation algorithm with multigrid generally improves after each iteration. Therefore, the desired accuracy for a given system modeling problem solution can often be obtained by adjusting the number of iterations of multigrid to perform. Multigrid should be considered as an alternative

method for solving the Weiner-Hopf equation for system modeling problems with very large matrices (high order filters). Depending on the nature of the particular problem, multigrid may result in a significant improvement in compute time when solving the Weiner-Hopf equation.

System modeling is just one of the many areas of analysis that can potentially be improved with multigrid techniques. This thesis was designed to provide a complete and comprehensive analysis of multigrid and the Toeplitz approximation algorithm so that other researchers can easily design and develop experiments for applying multigrid to their particular problem. All the MATLAB code used to run the computer simulations has been provided to facilitate further analysis. The following section provides several suggestions for future research on the application of multigrid to the system modeling problem.

B. RECOMMENDATIONS FOR FUTURE WORK

The possibilities for future applications of multigrid are endless. Just in the area of digital signal processing, there are several more experiments that could be conducted on applying multigrid to the system modeling problem. Typically, multigrid applies a single relaxation method to a problem. It would be interesting to examine the effects of applying a mix of relaxation methods with multigrid to the system modeling problem. A combination of the Toeplitz approximation algorithm for removing smooth components of the error and the weighted Jacobi method for removing oscillatory components of the error may improve the effect of multigrid in solving the Weiner-Hopf equation.

Combining multigrid restriction operators could be done to improve the effect of multigrid on modeling filters of varying lengths. Row lumping has been shown to work well with odd order FIR filters, but it would be interesting to compare the effectiveness of row lumping on even order filters. Conversely, injection has been demonstrated to work on even order filters and could easily be applied to odd order filters. Alternative prolongation operators could also be explored. The effect of non-linear prolongation operators, such as quadratic or cubic interpolation, could be compared to the effect of linear prolongation operators.

The effects of applying multigrid to the system modeling problem using adaptive filtering techniques instead of system identification could be studied. Adaptive filtering offers a whole new approach to the system modeling problem. The success of multigrid will be measured by the speed by which multigrid can provide an accurate solution to the problem. Therefore, it would be beneficial to investigate methods for improving the speed of the algorithms by developing efficient code to exploit the power of today's computers. Parallel processors could be used to further increase the speed of multigrid techniques. There are many more problems in engineering and mathematics that might benefit from multigrid. It is hoped that this paper might spark some interest to further expand the realm of multigrid analysis.

APPENDIX: MATLAB CODE

This appendix contains the MATLAB source code of the programs that provided the experimental results for this thesis. The source code is included as part of the thesis so that the experimental results can easily be reproduced. In addition to the experiment specific programs, there are some generic programs that can be used for other projects. Below is a brief description of each of the programs. More detail about each of the programs can be found in the detailed comments contained in the source code.

Signal Routines

`get_signal.m` - generates a random input white noise signal and the corresponding output for an elliptic IIR bandpass filter system

`get_correlation.m` - computes a data matrix, correlation matrix, and cross correlation vector for a given input signal, output signal, and FIR filter order

`get_toeplitz.m` - computes the Toeplitz and residual portions of a correlation matrix and generates the Toeplitz iteration matrix used in the Toeplitz approximation algorithm

Multigrid Operator Routines

`full_weighting.m` - generates the full weighting restriction operator for a selected space

`injection.m` - generates the injection restriction operator for a selected space

`row_lumping.m` - generates the row lumping restriction operator for a selected space

`linear_interp.m` - generates the linear interpolation prolongation operator for a selected space

Algorithm Routines

toeplitz_algorithm.m - performs the Toeplitz approximation algorithm for finding the coefficients of an FIR filter

levinson_recursion.m - performs the Levinson recursion for inverting a Toeplitz matrix

nested_toeplitz.m - recursively runs the Toeplitz approximation algorithm on coarse grids to obtain better initial guesses of coefficient vector on finer grids

multi_vcycle.m - performs the multigrid V-cycle scheme with the Toeplitz approximation algorithm to find coefficients of a FIR filter

multi_fmvcycle.m - performs the multigrid FMV-cycle scheme with the Toeplitz approximation algorithm to find coefficients of a FIR filter

Eigenvalue Plot Routines

save_eigens - computes the eigenvalues and eigenvectors of the Toeplitz iteration matrix for 8 different input signals and saves the values in a file for use by other experiment programs

plot_eigenvectors.m - determines eigenvectors corresponding to the 3 hi, 3 med, and 3 low eigenvalues of the Toeplitz iteration matrix, makes plots of eigenvectors, and plots PSDs of eigenvectors

plot_748_max_eigens.m - computes the maximum eigenvalue of the Toeplitz iteration matrix for each of 748 different input signals and plots maximum eigenvalue versus signal length

plot_8_max_eigens.m - plots the maximum eigenvalue of the Toeplitz iteration matrix for each of 8 different input signals versus signal length

plot_8_max_eigens_12.m - plots the maximum eigenvalue of the Toeplitz iteration matrix for each of 8 different input signals versus signal length for 12 different random number seeds on one page

plot_all_eigens.m - plots all eigenvalues of the Toeplitz iteration matrix for each of 8 different input signals on one page

plot_avg_max_eigens.m - plots the average maximum eigenvalues of the Toeplitz iteration matrix for 8 different input signals and 12 different random number seeds

Experiment Routines

run1_toeplitz.m - runs the Toeplitz approximation algorithm using eigenvectors corresponding to the smallest, middle, and largest eigenvalues of the correlation matrix as initial guesses with solution set to 0 ($r=0$ is cross correlation vector) and produce plots of the error norm and error spectrum

run2_toeplitz.m - run the Toeplitz approximation algorithm using $\text{inv}(T)*r$ as initial guess with solution set to $\text{inv}(R)*r$ and produce plots of the error norm and error spectrum

run3_toeplitz.m - run the Toeplitz approximation algorithm using multigrid nested iteration techniques to obtain better initial guesses at each grid level and produce plots of the error norm and error spectrum

run1_model.m - run multigrid with the Toeplitz approximation algorithm using V-cycle and FMV-cycle schemes to solve the Wiener-Hopf equation for an even order FIR filter (uses the full weighting and injection restriction operators)

run2_model.m - run multigrid with the Toeplitz approximation algorithm using V-cycle and FMV-cycle schemes to solve the Wiener-Hopf equation for an odd order FIR filter (uses the row lumping restriction operator)

get_signal.m

```
function [B,A,x,y]=get_signal(Ps,ns,seed)
%
% Generates a random normal input (white noise) signal and the
% output signal of a Pth order elliptic IIR bandpass filter
% with cutoff frequencies of .3*pi and .7*pi, passband ripple
% of .5 dB, and stopband attenuation of -30 dB
%
% Written by: John Volk (Nov 93)
%
% Inputs:  Ps = IIR filter order for system (must be even)
%          ns = number of points in input signal (ns >= 2*Ps+1)
%          seed = seed for random number generator
%
% Outputs: B = IIR filter zeros (feedforward coefficients)
%          A = IIR filter poles (feedback coefficients)
%          x = random normal input (white noise) signal
%          y = output signal from filter
%
if (rem(Ps,2) ~= 0)
    error('*** Error *** Filter order must be even.')
end
Rp = 0.5;           % passband ripple for filter in dB
Rs = 30.0;          % stopband attenuation for filter in dB
Wn = [0.3; 0.7];    % cutoff frequencies for filter in fraction of pi
[B,A] = ellip(Ps/2,Rp,Rs,Wn); % B=filter zeros, A=filter poles
randn('seed',seed); % set random number seed
x=randn(1,ns);      % generate white noise random input signal
y=filter(B,A,x);    % generate output from IIR filter
return
```

get_correlation.m

```
function [R,r,X]=get_correlation(x,y,P)
%
%   Computes the correlation matrix and cross correlation vector for given
%   input and system output signals to design a Pth order FIR filter
%
%   Written by: John Volk (Nov 93)
%
%
%   Inputs:   x = input signal
%             y = output signal
%             P = FIR filter order
%
%   Outputs:  R = correlation matrix
%             r = cross correlation vector
%             X = input signal data matrix
%
ns=length(x); % check for enough points in input signal
if (ns < 2*P+1)
    error('*** Error *** Need at least 2P+1 points (P=filter order).')
end
N=ns-P; % number of rows in input matrix
M=P+1; % number of columns in input matrix
X=zeros(N,M); % define input signal data matrix
for k=1:N
    X(k,:)=fliplr(x(k:k+P)); % Generate N rows of input matrix X
end
R=X'*X; % compute correlation matrix
r=X'*y(M:M+N-1); % compute cross correlation vector
return
```

get_toeplitz.m

```
function [T,S,Pt]=get_toeplitz(R)
%
%   Splits input matrix into Toeplitz and residual matrices
%   and calculates its Toeplitz algorithm iteration matrix
%
%   Written by: John Volk (Nov 93)
%
%
%   Inputs:  R = input matrix
%
%   Outputs: T = Toeplitz portion of input matrix
%            S = residual portion of input matrix
%            Pt = Toeplitz algorithm iteration matrix
%
[p,q]=size(R);           % determine the size of R
for i=1:q
    c(i,1)=mean(diag(R,i-1)); % determine the average of each diagonal
end
%
T=toeplitz(c);           % compute the Toeplitz portion of R
S=R-T;                  % compute residual portion of R
Tinv=inv(T);            % invert the Toeplitz matrix
Pt=Tinv*(T-R);          % compute iteration matrix
return
```

full_weighting.m

```
function I2h=full_weighting(N)
%
%   Generates the multigrid full weighting restriction operator matrix
%   for a selected space
%
%   Written by: Dean Richter
%   Modified by: John Volk (Feb 94)
%
%
%   Inputs:   N = space for which full weighting operator is desired
%             (must be even number >= 4)
%
%   Outputs:  I2h = full weighting operator [(N/2)-1 x N-1 matrix]
%
if (rem(N,2)~=0 | N<4)           % check for even space N>=4
    error('N -- index of full weighting operator must be even no. >= 4')
end
op=[1 2 1]; index=2;
I2h=[op zeros(1,N-1-length(op))]; % compute row 1 of I2h
%
for i=2:N/2-2                   % compute remaining columns of I2h
    Itemp=[zeros(1,index) op zeros(1,N-1-(index+length(op)))];
    I2h=[I2h; Itemp];
    index=index+2;
end
%
if (N>4)
    Ifinal=[zeros(1,index) op];
    I2h=0.25*[I2h; Ifinal];      % multiply I2h by weighting factor
else
    I2h=0.25*I2h;
end
return
```

injection.m

```
function I2h=injection(N)
%
%   Generates the multigrid injection restriction operator matrix
%   for a selected space
%
%   Written by: John Volk (Jun 94)
%
%   Inputs:   N = space for which injection operator is desired
%             (must be even number >= 4)
%
%   Outputs:  I2h = injection operator [(N/2)-1 x N-1 matrix]
%
if (rem(N,2)~=0 | N<4)          % check for even space N>=4
    error('N -- index of injection operator must be even no. >= 4')
end
op=[0 1 0]; index=2;
I2h=[op zeros(1,N-1-length(op))]; % compute row 1 of I2h
%
for i=2:N/2-2                  % compute remaining columns of I2h
    Itemp=[zeros(1,index) op zeros(1,N-1-(index+length(op)))];
    I2h=[I2h; Itemp];
    index=index+2;
end
%
if (N>4)
    Ifinal=[zeros(1,index) op];
    I2h=[I2h; Ifinal];
end
return
```

row_lumping.m

```
function I2h=row_lumping(N)
%
%   Generates the row lumping restriction operator matrix
%   for a selected space
%
%   Written by: John Volk (Jun 94)
%
%   Inputs:   N = space for which row lumping operator is desired
%             (must be even number >= 4)
%
%   Outputs:  I2h = row lumping operator [(N/2) x N matrix]
%
if (rem(N,2)~=0 | N<4)           % check for even space N>=4
    error('N -- index of row lumping operator must be even no. >= 4')
end
op=[1 1]; index=2;
if (N>4)
    I2h=[op zeros(1,N-length(op))]; % compute row 1 of I2h
else
    I2h=op;
end
%
for i=2:N/2-1                    % compute remaining columns of I2h
    Itemp=[zeros(1,index) op zeros(1,N-(index+length(op)))];
    I2h=[I2h; Itemp];
    index=index+2;
end
%
if (N>4)
    Ifinal=[zeros(1,index) op];
    I2h=[I2h; Ifinal];
end
return
```

linear_interp.m

```
function Ih=linear_interp(N);
%
%   Generates the multigrid linear interpolation prolongation operator matrix
%       for a selected space
%
%   Written by: Dean Richter
%   Modified by: John Volk (Feb 94)
%
%   Inputs:   N = space for which linear interpolation operator is desired
%              (must be even number >= 4)
%
%   Outputs:  Ih = linear interpolation operator [N-1 x (N/2)-1 matrix]
%
%
if (rem(N,2)~=0 | N<4)          % check for even space N>=4
    error('N -- index into the linear interpolation operator must be an even no. >= 4!!')
end
op=[1;2;1]; index=2;
Ih=[op; zeros(N-1-length(op),1)]; % compute column 1 of Ih
%
for i=2:N/2-2                  % compute the remaining columns of Ih
    Itemp=[zeros(index,1); op; zeros(N-1-(index+length(op)),1)];
    Ih=[Ih Itemp];
    index=index+2;
end
%
if (N>4)
    Ifinal=[zeros(index,1); op];
    Ih=0.5*[Ih Ifinal];        % multiply Ih by weighting factor
else
    Ih=0.5*Ih;
end
return
```

toeplitz_algorithm.m

```
function [a,Q]=toeplitz_algorithm(T,R,r,k,a,invmeth)
%
%   Determines model FIR filter coefficients using the
%   Toeplitz Approximation Algorithm
%   and Levinson recursion to invert the Toeplitz matrix
%   (can use direct matrix inversion with optional keyword)
%
%   Written by: Dean Richter
%   Modified by: John Volk (Feb 94)
%
%   Inputs:   T = Toeplitz portion of input correlation matrix
%             R = input correlation matrix
%             r = input cross correlation vector
%             k = number of iterations desired
%             a = initial guess of filter coefficients
%             invmeth = 'direct' for direct matrix inversion (optional)
%
%   Outputs:  a = model FIR filter coefficients
%             Q = matrix containing iterations of 'a'
%
if (nargin ~= 6)
    Tinv=levinson_recursion(T);    % invert with Levinson recursion
else
    if (strcmp(invmeth,'direct'))
        Tinv=inv(T);              % direct invert of Toeplitz matrix
    else
        Tinv=levinson_recursion(T); % invert with Levinson recursion
    end
end
a0=Tinv*r;                       % solve for a constant a0
mconst=Tinv*R;                   % solve for a matrix constant
Q=zeros(length(a),k);           % set aside memory for saving iterations
%
for m=1:k
    a=a0+a-mconst*a;             % perform the iteration
    Q(:,m)=a;                   % save the iteration
end
return
```

levinson_recursion.m

```
function [Rinv]=levinson_recursion(R)
%
%   Computes the inverse of a Toeplitz matrix using Levinson recursion
%
%   Written by: John Volk (Jun 94)
%
%   Inputs:   R = Toeplitz matrix
%
%   Outputs:  Rinv = inverse of Toeplitz matrix
%
[n,n2]=size(R);           % set size of Toeplitz matrix
%
%   Set initial conditions for algorithm
%
a=[1];
s=R(1);
r=R(2);
L=[1 zeros(1,n-1)];
%
%   Run Levinson recursion
%
for i=2:n
    d=r*flipud(a')/s;
    a=[a 0] - d*[0 fliplr(a)];
    s=(1-d^2)*s;
    r=[R(2:i+1)];
    L=[L; fliplr(a) zeros(1,n-i)];
end
%
D=L*R*L';                 % compute diagonal matrix
Dinv=diag(1 ./ diag(D),0); % take reciprocal of diagonal elements
%
Rinv=L'*Dinv*L;           % compute inverse of Toeplitz matrix
return
```

nested_toeplitz.m

```
function [a]=nested_toeplitz(R,r,k)
%
%   Recursively runs Toeplitz algorithm on coarse grids to obtain
%       initial guesses of coefficient vector on finer grids
%
%   Written by: John Volk (Apr 94)
%
%   Inputs:   R = autocorrelation matrix
%             r = cross correlation vector
%             k = number of iterations for Toeplitz algorithm
%
%   Outputs:  a = coefficient vector
%
[p,q]=size(R);                % determine the size of R
R2h=full_weighting(p+1)*R*linear_interp(p+1); % autocorrelation matrix to coarse
grid
r2h=full_weighting(p+1)*r;    % cross correlation vector to coarse grid
if (p > 3)
    a2h=nested_toeplitz(R2h,r2h,k); % recurse until coarsest grid
    [T,S,Pt]=get_toeplitz(R2h);    % get Toeplitz matrix
    [a2h,Q]=toeplitz_algorithm(T,R2h,r2h,k,a2h); % run algorithm to get coefficients
else
    a2h=r2h/R2h;                % compute coefficients at coarsest grid
end
a=linear_interp(p+1)*a2h;      % return initial guess to next finer grid
return
```

multi_vcycle.m

```

function [ah,Qk]=multi_vcycle(Rh,rh,k,ah,P0,oper)
%
%   Determines model FIR filter coefficients using the
%       multigrid V-cycle scheme with the
%       Toeplitz Approximation Algorithm
%
%   Modified by: John Volk (Jun 94)
%
%
%   Inputs:   Rh = input correlation matrix
%             rh = input cross correlation vector
%             k = number of iterations desired
%             ah = initial guess of filter coefficients
%             P0 = number of coefficients to determine
%             oper = multigrid restriction operator
%                   (full_weighting, injection, row_lumping)
%
%   Outputs:  ah = model FIR filter coefficients
%             Qk = matrix containing iterations of 'ah'
%
P=length(ah) ;    % number of coefficients at current grid level
[Th,Sh,Ph]=get_toeplitz(Rh);    % get Toeplitz matrix
[ah,Q]=toeplitz_algorithm(Th,Rh,rh,k,ah,'levinson'); % run Toeplitz algorithm
if (length(ah)==P0)
    Qk=Q;    % store initial coefficient vector
end
row=strcmp(oper,'row_lumping');    % check for row lumping
if ((rem((P+1)/2,2)==0 | row) & (P+1)/2 >= 4) % recurse if not coarsest grid
    if (strcmp(oper,'injection'))
        I2h=injection(P+1);    % get injection operator
        Ih=linear_interp(P+1);    % get linear interpolation operator
    elseif (strcmp(oper,'row_lumping'))
        I2h=row_lumping(P);    % get row lumping restrictor
        Ih=0.5*I2h';    % compute row lumping interpolator
    else
        I2h=full_weighting(P+1);    % get full weighting operator
        Ih=linear_interp(P+1);    % get linear interpolation operator
    end
    R2h=I2h*Rh*Ih;    % coarse grid correlation matrix
    dh=rh-Rh*ah;    % compute the residual
    r2h=I2h*dh;    % coarse grid cross correlation vec.
    a2h=zeros((P-1)/2,1);    % initial guess for coarse grid

```

```

[T2h,S2h,P2h]=get_toeplitz(R2h);      % get Toeplitz matrix
[a2h,Q]=multi_vcycle(R2h,r2h,k,a2h,P0,oper); % recursively run V-cycle
ah=ah+Ih*a2h;                        % coarse grid correction
[ah,Q]=toeplitz_algorithm(Th,Rh,rh,k,ah,'levinson'); % run Toeplitz algorithm
Qk=[Qk; Q];                          % store coefficient matrix
end
return

```

multi_fmvcycle.m

```
function [ah,Qk]=multi_fmvcycle(Rh,rh,k,ah,oper)
%
%   Determines model FIR filter coefficients using the
%   multigrid FMV-cycle scheme with the
%   Toeplitz Approximation Algorithm
%
%   Modified by: John Volk (Jun 94)
%
%   Inputs:   Rh = input correlation matrix
%             rh = input cross correlation vector
%             k = number of iterations desired
%             ah = initial guess of filter coefficients
%             oper = multigrid restriction operator
%                  (full_weighting, injection, row_lumping)
%
%   Outputs:  ah = model FIR filter coefficients
%             Qk = matrix containing iterations of 'ah'
%
P=length(ah);      % number of coefficients at current grid level
row=strcmp(oper,'row_lumping');      % check for row lumping
if ((rem((P+1)/2,2) ~= 0 & ~row) | (P+1)/2 < 4) % coarsest grid?
    ah=inv(Rh)*rh;      % yes: solve on coarsest grid
else
    % no: recurse
    if (strcmp(oper,'injection'))
        I2h=injection(P+1);      % get injection operator
        Ih=linear_interp(P+1);    % get linear interpolation operator
    elseif (strcmp(oper,'row_lumping'))
        I2h=row_lumping(P);      % get row lumping restrictor
        Ih=0.5*I2h';             % compute row lumping interpolator
    else
        I2h=full_weighting(P+1); % get full weighting operator
        Ih=linear_interp(P+1);    % get linear interpolation operator
    end
    R2h=I2h*Rh*Ih;             % coarse grid correlation matrix
    dh=rh-Rh*ah;               % compute the residual
    r2h=I2h*dh;                % coarse grid cross correlation vec.
    a2h=zeros((P-1)/2,1);      % initial guess for coarse grid
    [a2h,Qk]=multi_fmvcycle(R2h,r2h,k,a2h,oper); % recursively run FMV-cycle
    ah=ah+Ih*a2h;              % coarse grid correction
end
[ah,Qk]=multi_vcyle(Rh,rh,k,ah,P,oper); % run multigrid V-cycle
return
```

save_eigens.m

```
% save_eigens
%
% Compute eigenvalues and eigenvectors of the Toeplitz iteration matrix for
% 8 different input signals and save in a file for use by other programs
%
% Written by: John Volk (Dec 93)
%
%
clear variables
fprintf('%50s\n','Save Eigenvalues and Eigenvectors');
seed=input('Seed? '); % input seed for random number
P=126; % set filter order
ns=[253,350,450,550,650,750,850,1000]; % number of points in each signal
for j=1:8
    [B,A,x,y]=get_signal(22,ns(j),seed); % get input signal
    [R,r,X]=get_correlation(x,y,P); % get correlation matrix
    [T,S,Pt]=get_toeplitz(R); % get Toeplitz iteration matrix
    [eigvec,eigval]=eig(Pt); % determine eigenvalues/eigenvectors
    if j==1 % store eigenvalues and eigenvectors
        eigval1=diag(eigval);
        eigvec1=eigvec;
    elseif j==2
        eigval2=diag(eigval);
        eigvec2=eigvec;
    elseif j==3
        eigval3=diag(eigval);
        eigvec3=eigvec;
    elseif j==4
        eigval4=diag(eigval);
        eigvec4=eigvec;
    elseif j==5
        eigval5=diag(eigval);
        eigvec5=eigvec;
    elseif j==6
        eigval6=diag(eigval);
        eigvec6=eigvec;
    elseif j==7
        eigval7=diag(eigval);
        eigvec7=eigvec;
```

```
elseif j==8
    eigval8=diag(eigval);
    eigvec8=eigvec;
end
end
eval(['save eigen_seed',int2str(seed)]) % save file
end
```

plot_eigenvectors.m

```
% plot_eigenvectors
%
% Determine eigenvectors corresponding to the 3 hi, 3 med, and 3 low
% eigenvalues of the Toeplitz iteration matrix, make plots of
% eigenvectors, and plot PSDs of eigenvectors
%
% Written by: John Volk (Dec 93)
%
%
clear variables
fprintf('%50s\n','Plot Eigenvectors');
P=126;
seed=input('Seed? '); % input seed for random number
eval(['load eigen_seed',int2str(seed)]) % load file created by save_eigens.m
% Find 3 hi, 3 med, and 3 low eigenvalues for ns=253 case
fprintf('%50s\n','Finding Eigenvectors');
fprintf('ns=253\n')
x=sort(abs(eigval1));
y=[x(1:3);x(63:65);x(125:127)];
for j=1:9
    for k=1:P+1
        if (y(j) == abs(eigval1(k)))
            val253(j)=eigval1(k);
            vec253(:,j)=eigvec1(:,k);
        end
    end
end
% Find 3 hi, 3 med, and 3 low eigenvalues for ns=350 case
fprintf('ns=350\n')
x=sort(abs(eigval2));
y=[x(1:3);x(63:65);x(125:127)];
for j=1:9
    for k=1:P+1
        if (y(j) == abs(eigval2(k)))
            val350(j)=eigval2(k);
            vec350(:,j)=eigvec2(:,k);
        end
    end
end
% Find 3 hi, 3 med, and 3 low eigenvalues for ns=1000 case
fprintf('ns=1000\n')
x=sort(abs(eigval8));
```

```

y=[x(1:3);x(63:65);x(125:127)];
for j=1:9
    for k=1:P+1
        if (y(j) == abs(eigval8(k)))
            val1000(j)=eigval8(k);
            vec1000(:,j)=eigvec8(:,k);
        end
    end
end
% Plot 9 sets of eigenvectors for ns=253 signal
fprintf('%50s\n','Plotting Eigenvectors');
for j=1:9
    b=' ';
    if (j == 1)
        b=['Seed=',int2str(seed),' ns=253'];
    end
    a1=num2str(val253(j));
    a2=['Eigval=',a1];
    subplot(3,3,j),plot(vec253(:,j)),
    title(b),
    xlabel(a2),
    ylabel('Eigenvector')
end
eval(['print seed',int2str(seed),'_eigvec-ns253'])
% Plot 9 sets of eigenvectors for ns=350 signal
for j=1:9
    b=' ';
    if (j == 1)
        b=['Seed=',int2str(seed),' ns=350'];
    end
    a1=num2str(val350(j));
    a2=['Eigval=',a1];
    subplot(3,3,j),plot(vec350(:,j)),
    title(b),
    xlabel(a2),
    ylabel('Eigenvector')
end
eval(['print seed',int2str(seed),'_eigvec-ns350'])
% Plot 9 sets of eigenvectors for ns=1000 signal
for j=1:9
    b=' ';
    if (j == 1)
        b=['Seed=',int2str(seed),' ns=1000'];
    end
end

```

```

    a1=num2str(val1000(j));
    a2=['Eigval=',a1];
    subplot(3,3,j),plot(vec1000(:,j)),
    title(b),
    xlabel(a2),
    ylabel('Eigenvector')
end
eval(['print seed',int2str(seed),'_eigvec-ns1000'])
% Plot 9 PSDs of sets of eigenvectors for ns=253 signal
fprintf('%50s\n','Plotting PSDs');
xaxis=0:63;
for j=1:9
    b=' ';
    if (j == 1)
        b=['Seed=',int2str(seed),' ns=253'];
    end
    a1=num2str(val253(j));
    a2=['Eigval=',a1];
    y=fft(vec253(:,j));
    ypsd=y.*conj(y)/(P+1);
    subplot(3,3,j),plot(xaxis,ypsd(1:64)),
    title(b),
    xlabel(a2),
    ylabel('Eigenvector PSD'),
    axis([0 64 0 max(ypsd)])
end
eval(['print seed',int2str(seed),'_eigpsd-ns253'])
% Plot 9 PSDs of sets of eigenvectors for ns=350 signal
for j=1:9
    b=' ';
    if (j == 1)
        b=['Seed=',int2str(seed),' ns=350'];
    end
    a1=num2str(val350(j));
    a2=['Eigval=',a1];
    y=fft(vec350(:,j));
    ypsd=y.*conj(y)/(P+1);
    subplot(3,3,j),plot(xaxis,ypsd(1:64)),
    title(b),
    xlabel(a2),
    ylabel('Eigenvector PSD'),
    axis([0 64 0 max(ypsd)])
end
eval(['print seed',int2str(seed),'_eigpsd-ns350'])

```

```

% Plot 9 PSDs of sets of eigenvectors for ns=1000 signal
for j=1:9
    b=' ';
    if (j == 1)
        b=['Seed=',int2str(seed),' ns=1000'];
    end
    a1=num2str(val1000(j));
    a2=['Eigval=',a1];
    y=fft(vec1000(:,j));
    ypsd=y.*conj(y)/(P+1);
    subplot(3,3,j),plot(xaxis,ypsd(1:64)),
    title(b),
    xlabel(a2),
    ylabel('Eigenvector PSD'),
    axis([0 64 0 max(ypsd)])
end
eval(['print seed',int2str(seed),'_eigpsd-ns1000'])
end

```

plot_748_max_eigens.m

```
% plot_748_max_eigens
%
% Compute maximum eigenvalues of the Toeplitz iteration matrix
% for 748 different input signals and plot
%
% Written by: John Volk (Dec 93)
%
%
clear variables
fprintf('%50s\n','Plot 748 Maximum Eigenvalues');
seed=input('Seed? '); % input seed for random number generator
P=126; % set filter order
ns=linspace(253,1000,748); % Number of points in each signal
for j=1:748
    [B,A,x,y]=get_signal(22,ns(j),seed); % get input signal
    [R,r,X]=get_correlation(x,y,P); % get correlation matrix
    [T,S,Pt]=get_toeplitz(R); % get Toeplitz iteration matrix
    [eigvec,eigval]=eig(Pt); % determine eigenvalues/eigenvectors
    max_eig(j)=max(abs(diag(eigval))); % determine maximum eigenvalue
end
% Plot 748 maximum eigenvalues
plot(ns,max_eig)
axis([200 1000 0 1])
title2=[' (Seed=',int2str(seed),' ,Filter order=',int2str(P),')'];
title(['Maximum Eigenvalues of Iteration Matrix',title2])
xlabel('No. Samples in Signal')
ylabel('Max. Eigenvalue')
print max_eigs_748
save max_eigs_748 max_eig
end
```

plot_8_max_eigens.m

```
% plot_8_max_eigens
%
% Plot maximum eigenvalues of the Toeplitz iteration matrix for each of
% 8 input signals
%
% Written by: John Volk (Dec 93)
%
%
clear variables
fprintf('%50s\n','Plot Maximum Eigenvalues');
seed=input('Seed? '); % get random number seed
infile=['eigen_seed',int2str(seed)]; % load file created by save_eigens.m
eval(['load ',infile])
fprintf('%22s%5d Filter order=%5d\n','Seed=',seed,P)
for k=1:8
    eval(['max_eig(k)=max(abs(eigval',int2str(k),'))']);
end
plot(ns,max_eig,'-',ns,max_eig,'*')
axis([200 1000 0 1])
title2=[' (Seed=',int2str(seed),',Filter order=',int2str(P),')'];
title(['Maximum Eigenvalues of Iteration Matrix',title2])
xlabel('No. Samples in Signal')
ylabel('Max. Eigenvalue')
eval(['print ',infile])
end
```

plot_8_max_eigens_12.m

```
% plot_8_max_eigens_12
%
% Plot maximum eigenvalues of the Toeplitz iteration matrix for 12 random seeds
% and 8 input signals on one page
%
% Written by: John Volk (Dec 93)
%
%
clear variables
fprintf('%50s\n','Plot Maximum Eigenvalues with Subplot');
seed=[1,2,3,5,7,11,13,17,19,23,29,31]; % set 12 random number seeds
for i=1:12
    infile=['eigen_seed',int2str(seed(i))];
    eval(['load ',infile]) % load files created by save_eigens.m
    for k=1:8
        eval(['max_eig(i,k)=max(abs(eigval',int2str(k),'));'])
    end
    a=['Seed = ',int2str(seed(i))];
    subplot(4,3,i),plot(ns,max_eig(i,:),'- ',ns,max_eig(i,:),'*'),
        axis([200 1000 0 1]),
        title(a),
        xlabel('No. Samples'),
        ylabel('Max. Eigs')
end
print eigs_12
end
```

plot_all_eigens.m

```
% plot_all_eigens
%
% Plot all eigenvalues of the Toeplitz iteration matrix for each
% of 8 signals on one page
%
% Written by: John Volk (Dec 93)
%
%
clear variables
fprintf('%50s\n','Plot All Eigenvalues with Subplot');
seed=input('Seed? '); % input seed for random number
infile=['eigen_seed',int2str(seed)]; % load file created by save_eigens.m
eval(['load ',infile])
for k=1:8
    eval(['max_eig(k)=max(abs(eigval',int2str(k),'));'])
    a=['subplot(3,3,k),plot(abs(eigval',int2str(k),')),''];
    a2=['axis([0 200 0 1]),'];
    b1=[int2str(ns(k)), ' Point Signal'];
    b='title(b1),';
    c1='Eigenvalue';
    c='ylabel(c1)';
    eval([a,a2,b,c])
end
d=['Seed=',int2str(seed)];
subplot(3,3,9),plot(ns,max_eig,'-',ns,max_eig,'*'),
    axis([0 1000 0 1]),
    title(d),
    xlabel('No. Samples'),
    ylabel('Max. Eigs')
eval(['print all_eigs',int2str(seed)])
end
```

plot_avg_max_eigens.m

```
% plot_avg_max_eigens
%
% Plot average maximum eigenvalues of the Toeplitz iteration matrix for
% 8 input signals and 12 random number seeds
%
% Written by: John Volk (Dec 93)
%
%
clear variables
fprintf('%50s\n','Plot Maximum Eigenvalues Average');
seed=[1,2,3,5,7,11,13,17,19,23,29,31]; % set 12 random number seeds
for i=1:12
    infile=['eigen_seed',int2str(seed(i))];
    eval(['load ',infile]) % load files created by save_eigens.m
    for k=1:8
        eval(['max_eig(i,k)=max(abs(eigval',int2str(k),'))']);
    end
end
for k=1:8
    maxeig_avg(k)=mean(max_eig(:,k)); % compute average of 8 maximum eigenvalues
end
plot(ns,maxeig_avg,'-',ns,maxeig_avg,'*')
axis([200 1000 0 1])
title2=[' (Avg of 12 seeds)'];
title(['Maximum Eigenvalues of Iteration Matrix',title2])
xlabel('No. Samples in Signal')
ylabel('Max. Eigenvalue')
print max_eigs_avg
save max_eigs_avg max_eig maxeig_avg ns
end
```

run1_toeplitz.m

```
% run1_toeplitz
%
% Run Toeplitz algorithm for selected seed and input signal using
% eigenvectors corresponding to the smallest, middle, and largest
% eigenvalues of the correlation matrix as initial guesses
% with solution set to 0 (r=0 is cross correlation vector)
%
% The following five plots are produced by this procedure:
% 1) All eigenvalues of correlation matrix
% All eigenvectors of smallest, middle, and largest eigenvalues
% [4 plots on one page]
% 2) Error norm vs iteration number for Toeplitz algorithm
% with three different initial guesses
% [3 plots on one axis]
% 3) PSD of errors at (0,1,2,3,4,5,10,15,20) iterations for
% Toeplitz algorithm with smallest eigenvector initial guess
% [9 plots on one page]
% 4) PSD of errors at (0,1,2,3,4,5,10,15,20) iterations for
% Toeplitz algorithm with middle eigenvector initial guess
% [9 plots on one page]
% 5) PSD of errors at (0,1,2,3,4,5,10,15,20) iterations for
% Toeplitz algorithm with largest eigenvector initial guess
% [9 plots on one page]
%
%
% Written by: John Volk (Feb 94)
%
%
clear variables
fprintf('\nRun Toeplitz Algorithm with solution zero\n');
%
% Load variables
%
seed=input('Seed? '); % get random number seed from user
ns=input('Number of points in input signal? '); % get signal size
Ps=22; % set IIR filter order (system)
P=126; % set FIR filter order (model)
%
% Compute required matrices
%
fprintf('\nCompute required matrices');
[B,A,x,y]=get_signal(Ps,ns,seed); % get input and output signals
```

```

[R,r,X]=get_correlation(x,y,P);    % get correlation matrices
r=r.*0;                          % set cross correlation vector to all zeros
[T,S,Pt]=get_toeplitz(R);        % get Toeplitz and iteration matrices
[eigvec,eigvals]=eig(R);         % determine eigenvalues and eigenvectors
eigval=diag(eigvals);
%
% Find smallest, middle, and largest eigenvalues of the correlation matrix
%
fprintf('\nFind smallest, middle, and largest eigenvalues');
eigval_sorted=sort(abs(eigval));
eigval_lmh=[eigval_sorted(1);eigval_sorted(64);eigval_sorted(127)];
for j=1:3
    for k=1:P+1
        if (eigval_lmh(j) == abs(eigval(k)))
            eigvec_lmh(:,j)=eigvec(:,k);
        end
    end
end
%
% Run Toeplitz algorithm using three sets of eigenvectors as initial guess
% j=1 (eigenvectors of smallest eigenvalue)
% j=2 (eigenvectors of middle eigenvalue)
% j=3 (eigenvectors of largest eigenvalue)
%
as=inv(R)*r;                      % compute expected coefficients
k=20;                            % set number of iterations for Toeplitz algorithm
nc=['small','mid ','large']; % set three cases for running Toeplitz algorithm
ic=[1 2 3 4 5 10 15 20]; % set iteration numbers for PSD plots
for j=1:3
    fprintf('\nRun Toeplitz algorithm for eigenvalue = %5s',nc(j,:));
    a=eigvec_lmh(:,j); % set initial guess for Toeplitz algorithm
    error_start(:,j)=as-a; % set initial error
    [a,Q]=toeplitz_algorithm(T,R,r,k,a); % run Toeplitz algorithm
    for i=1:k
        error_abs(:,i)=as-Q(:,i); % compute absolute error
        error_norm(j,i)=norm(error_abs(:,i)); % compute norm error
    end
    for l=1:8 % compute PSD of errors for smallest, middle, and largest cases
        efft=fft(error_abs(:,ic(l))); % compute FFT of error
        if (j == 1)
            epsd_low(:,ic(l))=efft.*conj(efft)/(P+1);
        end
        if (j == 2)
            epsd_med(:,ic(l))=efft.*conj(efft)/(P+1);
        end
    end
end

```

```

        end
        if (j == 3)
            epsd_hi(:,ic(1)) = efft.*conj(efft)/(P+1);
        end
    end
end
%
% Plot all eigenvalues of iteration matrix
%
fprintf('\nPlot eigenvalues and eigenvectors');
b1=['Seed=',int2str(seed),' ns=',int2str(ns)];
subplot(2,2,1),plot(abs(eigval)),
    title(b1),
    ylabel('Eigenvalue'),
    axis([0 150 0 max(abs(eigval))])
%
% Plot eigenvectors of smallest, middle, and largest eigenvalues
%
for j=1:3
    a1=['Eig_',nc(j,:),']='',num2str(eigval_lmh(j))];
    subplot(2,2,j+1),plot(eigvec_lmh(:,j)),
        title(a1),
        ylabel('Eigenvector')
end
subplot(111)
eval(['print seed',int2str(seed),'_eig1_ns',int2str(ns)])
%
% Plot error norms vs iteration number
%
fprintf('\nPlot error norms vs iteration number');
it=0:20; % set x axis values
elow=[norm(error_start(:,1)) error_norm(1,:)]; % set y axis small values
emed=[norm(error_start(:,2)) error_norm(2,:)]; % set y axis middle values
ehi =[norm(error_start(:,3)) error_norm(3,:)]; % set y axis large values
b1=['Seed=',int2str(seed),' ns=',int2str(ns), ...
    ' (o=',num2str(eigval_lmh(1)), ...
    ', x=',num2str(eigval_lmh(2)), ...
    ', +=',num2str(eigval_lmh(3)),')'];
semilogy(it,elow,'o',it,elow,'-', ...
    it,emed,'x',it,emed,'-', ...
    it,ehi,'+',it,ehi,'-')
title(b1)
xlabel('Iteration Number')
ylabel('Error Norm (a=0)')

```

```

eval(['print seed',int2str(seed),'_norm1_ns',int2str(ns)])
%
% Plot PSD of errors at selected iterations
%
fprintf('\nPlot PSD of errors at selected iterations');
xaxis=0:63;          % set plot points for x-axis
for j=1:3
    b2=['Seed=',int2str(seed),' ns=',int2str(ns), ...
        ' (Eig_',nc(j,:),')=',num2str(eigval_lmh(j)),')'];
    efft=fft(error_start(:,j));
    epsd=efft.*conj(efft)/(P+1);
    subplot(3,3,1),plot(xaxis,epsd(1:64)),
        title(b2),
        xlabel('Initial'),
        ylabel('Error PSD (a=0)'),
        axis([0 64 0 max(epsd)])
    for i=1:8
        a1=['Iteration=',int2str(ic(i))];
        if (j == 1)
            epsd=epsd_low(:,ic(i));
        end
        if (j == 2)
            epsd=epsd_med(:,ic(i));
        end
        if (j == 3)
            epsd=epsd_hi(:,ic(i));
        end
        subplot(3,3,i+1),plot(xaxis,epsd(1:64)),
            xlabel(a1),
            ylabel('Error PSD (a=0)'),
            axis([0 64 0 max(epsd)])
    end
    eval(['print seed',int2str(seed),'_psd1_ns',int2str(ns),nc(j,:)])
    subplot(111)
end
fprintf('\n\nEnd Toeplitz run\n\n');
end

```

run2_toeplitz.m

```
% run2_toeplitz
%
% Run Toeplitz algorithm for selected seed and input signal using
%   inv(T)*r as initial guess with solution set to inv(R)*r
%
% The following two plots are produced by this procedure:
%   1) Error norm vs iteration number for Toeplitz algorithm
%   2) PSD of error at (0,1,2,3,4,5,10,15,20) iterations for
%       Toeplitz algorithm with low eigenvector initial guess
%       [9 plots on one page]
%
% Written by: John Volk (Mar 94)
%
%
clear variables
fprintf('\nRun Toeplitz Algorithm\n');
%
% Load variables
%
seed=input('Seed? '); % get random number seed from user
ns=input('Number of points in input signal? '); % get signal size
Ps=22; % set IIR filter order (system)
P=126; % set FIR filter order (model)
%
% Compute required matrices
%
fprintf('\nCompute required matrices');
[B,A,x,y]=get_signal(Ps,ns,seed); % get input and output signals
[R,r,X]=get_correlation(x,y,P); % get correlation matrices
[T,S,Pt]=get_toeplitz(R); % get Toeplitz and iteration matrices
%
% Run Toeplitz algorithm using inv(T)*r as initial guess
%
as=inv(R)*r; % compute expected coefficients
k=20; % set number of iterations for Toeplitz algorithm
ic=[1 2 3 4 5 10 15 20]; % set iteration numbers for PSD plots
a=inv(T)*r; % set initial guess for Toeplitz algorithm
error_start=a-a; % set initial error
efft=fft(error_start);
epstd_start=efft.*conj(efft)/(P+1); % compute PSD of initial error
fprintf('\nRun Toeplitz algorithm');
```

```

[a,Q]=toeplitz_algorithm(T,R,r,k,a);    % run Toeplitz algorithm
for i=1:k
    error_abs(:,i)=as-Q(:,i);          % compute absolute error
    error_norm(i)=norm(error_abs(:,i)); % compute norm error
end
for l=1:8 % compute PSD of errors for selected iterations
    efft=fft(error_abs(:,ic(l)));
    epsd_iter(:,ic(l))=efft.*conj(efft)/(P+1);
end
%
% Plot error norms vs iteration number
%
fprintf('\nPlot error norms vs iteration number');
it=0:20; % set x axis values
enorm=[norm(error_start) error_norm]; % set y axis values
b1=['Seed=',int2str(seed),' ns=',int2str(ns)];
semilogy(it,enorm,'o',it,enorm,'-')
title(b1)
xlabel('Iteration Number')
ylabel('Error Norm')
eval(['print seed',int2str(seed),'_norm2_ns',int2str(ns)])
%
% Plot PSD of errors at selected iterations
%
fprintf('\nPlot PSD of errors at selected iterations');
xaxis=0:63; % set plot points for x-axis
b2=['Seed=',int2str(seed),' ns=',int2str(ns)];
subplot(3,3,1),plot(xaxis,epsd_start(1:64)),
title(b2),
xlabel('Initial'),
ylabel('Error PSD'),
axis([0 64 0 max(epsd_start)])
for i=1:8
    a1=['Iteration=',int2str(ic(i))];
    epsd=epsd_iter(:,ic(i));
    subplot(3,3,i+1),plot(xaxis,epsd(1:64)),
    xlabel(a1),
    ylabel('Error PSD'),
    axis([0 64 0 max(epsd)])
    eval(['print seed',int2str(seed),'_psd2_ns',int2str(ns)])
    subplot(111)
end
fprintf('\n\nEnd Toeplitz run\n\n');
end

```

run3_toeplitz.m

```
% run3_toeplitz
%
% Run Toeplitz algorithm using nested iteration multigrid
%
% Multigrid is used to reduce the auto-correlation matrix, cross-
% correlation vector, and coefficient vector to the 1x1 case
% and the Toeplitz algorithm is then run on finer grids
% using the solution at the next coarser grid to form the
% initial guess
%
% Written by: John Volk (Apr 94)
%
clear variables
fprintf('\nRun Toeplitz Algorithm\n');
%
% Load variables
%
seed=input('Seed? '); % get random number seed from user
ns=input('Number of points in input signal? '); % get signal size
Ps=22; % set IIR filter order (system)
P=126; % set FIR filter order (model)
%
% Compute required matrices
%
fprintf('\nCompute required matrices');
[B,A,x,y]=get_signal(Ps,ns,seed); % get input and output signals
[R,r,X]=get_correlation(x,y,P); % get correlation matrices
[T,S,Pt]=get_toeplitz(R); % get Toeplitz and iteration matrices
%
% Recursively run Toeplitz algorithm on coarser grids and return initial
% guess of coefficient vector
%
as=inv(R)*r; % compute expected coefficients
k=20; % set number of iterations for Toeplitz algorithm
ic=[1 2 3 4 5 10 15 20]; % set iteration numbers for PSD plots
fprintf('\nRun nested Toeplitz to get initial guess\n');
a=nested_toeplitz(R,r,k); % get initial guess for Toeplitz algorithm
error_start=a-a; % set initial error
efft=fft(error_start);
epsd_start=efft.*conj(efft)/(P+1); % compute PSD of initial error
fprintf('\nRun Toeplitz algorithm');
```

```

[a,Q]=toeplitz_algorithm(T,R,r,k,a);    % run Toeplitz algorithm
for i=1:k
    error_abs(:,i)=as-Q(:,i);          % compute absolute error
    error_norm(i)=norm(error_abs(:,i)); % compute norm error
end
for l=1:8 % compute PSD of errors for selected iterations
    efft=fft(error_abs(:,ic(l)));
    epsd_iter(:,ic(l))=efft.*conj(efft)/(P+1);
end
%
% Plot error norms vs iteration number
%
fprintf('\nPlot error norms vs iteration number');
it=0:20; % set x axis values
enorm=[norm(error_start) error_norm]; % set y axis values
b1=['Nested: Seed=',int2str(seed),' ns=',int2str(ns)];
semilogy(it,enorm,'o',it,enorm,'-')
    title(b1)
    xlabel('Iteration Number')
    ylabel('Error Norm')
eval(['print seed',int2str(seed),'_norm3_ns',int2str(ns)])
%
% Plot PSD of errors at selected iterations
%
fprintf('\nPlot PSD of errors at selected iterations');
xaxis=0:63; % set plot points for x-axis
b2=['Nested: Seed=',int2str(seed),' ns=',int2str(ns)];
subplot(3,3,1),plot(xaxis,epsd_start(1:64)),
    title(b2),
    xlabel('Initial'),
    ylabel('Error PSD'),
    axis([0 64 0 max(epsd_start)])
for i=1:8
    a1=['Iteration=',int2str(ic(i))];
    epsd=epsd_iter(:,ic(i));
    subplot(3,3,i+1),plot(xaxis,epsd(1:64)),
        xlabel(a1),
        ylabel('Error PSD'),
        axis([0 64 0 max(epsd)])
    eval(['print seed',int2str(seed),'_psd3_ns',int2str(ns)])
    subplot(111)
end
fprintf('\n\nEnd Toeplitz run\n\n');
end

```

run1_model.m

```
% run1_model
%
% Written by: John Volk (Jun 94)
%
%
clear variables
fprintf('\nRun Multigrid\n');
%
% Load variables
%
seed=input('Seed? '); % get random number seed from user
ns=input('Number of points in input signal? '); % get signal size
k=input('Number of iterations for algorithms? '); % get number of iterations
Ps=22; % set IIR filter order (system)
P=126; % set FIR filter order (model)
%
% Compute required matrices
%
fprintf('\nCompute required matrices');
[B,A,x,y]=get_signal(Ps,ns,seed); % get input and output signals
[R,r,X]=get_correlation(x,y,P); % get correlation matrices
[T,S,Pt]=get_toeplitz(R); % get Toeplitz and iteration matrices
%
% Direct matrix inversion method
%
fprintf('\nRun direct matrix inversion');
b0=inv(R)*r; % compute FIR filter coefficients
y0=filter(b0,1,x); % get model output
e0=norm(y-y0); % compute error norm
fft0=fft(y-y0);
psd0=fft0.*conj(fft0)/(P+1); % compute PSD of error
%
a=zeros(P+1,1); % set initial guess for multigrid
%
% Toeplitz algorithm with Levinson recursion
%
fprintf('\nRun Toeplitz algorithm');
[b1,Q1]=toeplitz_algorithm(T,R,r,k,a,'levinson'); % run Toeplitz algorithm
y1=filter(b1,1,x); % get model output
e1=norm(y-y1); % compute error norm
fft1=fft(y-y1);
psd1=fft1.*conj(fft1)/(P+1); % compute PSD of error
```

```

%
%   Multigrid V-cycle with Toeplitz algorithm and full weighting
%
fprintf('\nRun Multigrid V-cycle with full weighting');
[b2,Q2]=multi_vcycle(R,r,k,a,P+1,'full_weighting'); % run multigrid V-cycle
y2=filter(b2,1,x); % get model output
e2=norm(y-y2); % compute error norm
fft2=fft(y-y2);
psd2=fft2.*conj(fft2)/(P+1); % compute PSD of error
%
%   Multigrid V-cycle with Toeplitz algorithm and injection
%
fprintf('\nRun Multigrid V-cycle with injection');
[b3,Q3]=multi_vcycle(R,r,k,a,P+1,'injection'); % run multigrid V-cycle
y3=filter(b3,1,x); % get model output
e3=norm(y-y3); % compute error norm
fft3=fft(y-y3);
psd3=fft3.*conj(fft3)/(P+1); % compute PSD of error
%
%   Multigrid FMV-cycle with Toeplitz algorithm and full weighting
%
fprintf('\nRun Multigrid FMV-cycle with full weighting');
[b4,Q4]=multi_fmvcycle(R,r,k,a,'full_weighting'); % run multigrid FMV-cycle
y4=filter(b4,1,x); % get model output
e4=norm(y-y4); % compute error norm
fft4=fft(y-y4);
psd4=fft4.*conj(fft4)/(P+1); % compute PSD of error
%
%   Multigrid FMV-cycle with Toeplitz algorithm and injection
%
fprintf('\nRun Multigrid FMV-cycle with injection');
[b5,Q5]=multi_fmvcycle(R,r,k,a,'injection'); % run multigrid FMV-cycle
y5=filter(b5,1,x); % get model output
e5=norm(y-y5); % compute error norm
fft5=fft(y-y5);
psd5=fft5.*conj(fft5)/(P+1); % compute PSD of error
%
%   Write results to file
%
file=['seed',int2str(seed),'_model1_k',int2str(k),'_ns',int2str(ns)];
fprintf(file,'\nFilter order : %4.0f',P);
fprintf(file,'\nInput signal length : %4.0f',ns);
fprintf(file,'\nNumber of iterations : %4.0f\n',k);
fprintf(file,'
Error norm');

```

```

fprintf(file,'\nDirect matrix inversion                %12.8f',e0);
fprintf(file,'\nToeplitz algorithm                    %12.8f',e1);
fprintf(file,'\nMultigrid V-cycle with full weighting    %12.8f',e2);
fprintf(file,'\nMultigrid V-cycle with injection operator %12.8f\n',e3);
fprintf(file,'\nMultigrid FMV-cycle with full weighting operator %12.8f\n',e4);
fprintf(file,'\nMultigrid FMV-cycle with injection operator %12.8f\n',e5);
%
%   Plot PSD of errors
%
if (rem(ns,2) ~= 0)
    xmax=(ns+1)/2;
else
    xmax=ns/2;
end
fprintf('\nPlot PSD of errors\n');
xaxis=0:xmax-1;          % set plot points for x-axis
subplot(3,2,1),plot(xaxis,psd0(1:xmax)),
    title(['Seed=',int2str(seed),' ns=',int2str(ns),' P=',int2str(P)]),
    xlabel('Direct Inversion'),
    ylabel('Error PSD'),
    axis([0 xmax 0 max(psd0)])
subplot(3,2,2),plot(xaxis,psd1(1:xmax)),
    xlabel('Toeplitz algorithm'),
    ylabel('Error PSD'),
    axis([0 xmax 0 max(psd1)])
subplot(3,2,3),plot(xaxis,psd2(1:xmax)),
    xlabel('Multi V, full weighting'),
    ylabel('Error PSD'),
    axis([0 xmax 0 max(psd2)])
subplot(3,2,4),plot(xaxis,psd3(1:xmax)),
    xlabel('Multi V, injection'),
    ylabel('Error PSD'),
    axis([0 xmax 0 max(psd3)])
subplot(3,2,5),plot(xaxis,psd4(1:xmax)),
    xlabel('Multi FMV, full weighting'),
    ylabel('Error PSD'),
    axis([0 xmax 0 max(psd4)])
subplot(3,2,6),plot(xaxis,psd5(1:xmax)),
    xlabel('Multi FMV, injection'),
    ylabel('Error PSD'),
    axis([0 xmax 0 max(psd5)])
eval(['print ',file])
eval(['save psds1_k',int2str(k),'_ns',int2str(ns),' psd0 psd1 psd2 psd3 psd4 psd5'])
end

```

run2_model.m

```
% run2_model
%
% Written by: John Volk (Jun 94)
%
%
clear variables
fprintf('\nRun Multigrid Odd Filter\n');
%
% Load variables
%
seed=input('Seed? '); % get random number seed from user
ns=input('Number of points in input signal? '); % get signal size
k=input('Number of iterations for algorithms? '); % get number of iterations
Ps=22; % set IIR filter order (system)
P=127; % set FIR filter order (model)
%
% Compute required matrices
%
fprintf('\nCompute required matrices');
[B,A,x,y]=get_signal(Ps,ns,seed); % get input and output signals
[R,r,X]=get_correlation(x,y,P); % get correlation matrices
[T,S,Pt]=get_toeplitz(R); % get Toeplitz and iteration matrices
%
% Direct matrix inversion method
%
fprintf('\nRun direct matrix inversion');
b1=inv(R)*r; % compute FIR filter coefficients
y1=filter(b1,1,x); % get model output
e1=norm(y-y1); % compute error norm
fft1=fft(y-y1);
psd1=fft1.*conj(fft1)/(P+1); % compute PSD of error
%
a=zeros(P+1,1); % set initial guess for multigrid
%
% Toeplitz algorithm
%
fprintf('\nRun Toeplitz algorithm');
[b2,Q2]=toeplitz_algorithm(T,R,r,k,a,'levinson'); % run Toeplitz algorithm
y2=filter(b2,1,x); % get model output
e2=norm(y-y2); % compute error norm
fft2=fft(y-y2);
psd2=fft2.*conj(fft2)/(P+1); % compute PSD of error
```

```

%
%   Multigrid V-cycle with Toeplitz algorithm and row lumping
%
fprintf('\nRun Multigrid V-cycle with row lumping');
[b3,Q3]=multi_vcycle(R,r,k,a,P+1,'row_lumping'); % run multigrid V-cycle
y3=filter(b3,1,x); % get model output
e3=norm(y-y3); % compute error norm
fft3=fft(y-y3);
psd3=fft3.*conj(fft3)/(P+1); % compute PSD of error
%
%   Multigrid FMV-cycle with Toeplitz algorithm and row lumping
%
fprintf('\nRun Multigrid FMV-cycle with row lumping');
[b4,Q4]=multi_fmvcycle(R,r,k,a,'row_lumping'); % run multigrid FMV-cycle
y4=filter(b4,1,x); % get model output
e4=norm(y-y4); % compute error norm
fft4=fft(y-y4);
psd4=fft4.*conj(fft4)/(P+1); % compute PSD of error
%
%   Write results to file
%
file=['seed',int2str(seed),'_model2_k',int2str(k),'_ns',int2str(ns)];
fprintf(file,'\nFilter order : %4.0f',P);
fprintf(file,'\nInput signal length : %4.0f',ns);
fprintf(file,'\nNumber of iterations : %4.0f\n',k);
fprintf(file,'
Error norm');
fprintf(file,'\nDirect matrix inversion %12.8f',e1);
fprintf(file,'\nToeplitz algorithm %12.8f',e2);
fprintf(file,'\nMultigrid V-cycle with row lumping operator %12.8f',e3);
fprintf(file,'\nMultigrid FMV-cycle with row lumping operator %12.8f\n',e4);
%
%   Plot PSD of errors
%
if (rem(ns,2) ~= 0)
    xmax=(ns+1)/2;
else
    xmax=ns/2;
end
fprintf('\nPlot PSD of errors\n');
xaxis=0:xmax-1; % set plot points for x-axis
subplot(2,2,1),plot(xaxis,psd1(1:xmax)),
    title(['Seed=',int2str(seed),' ns=',int2str(ns),' P=',int2str(P)]),
    xlabel('Direct Inversion'),
    ylabel('Error PSD'),

```

```

    axis([0 xmax 0 max(psd1)])
subplot(2,2,2),plot(xaxis,psd2(1:xmax)),
    xlabel('Toeplitz algorithm'),
    ylabel('Error PSD'),
    axis([0 xmax 0 max(psd2)])
subplot(2,2,3),plot(xaxis,psd3(1:xmax)),
    xlabel('Multi V, row lumping'),
    ylabel('Error PSD'),
    axis([0 xmax 0 max(psd3)])
subplot(2,2,4),plot(xaxis,psd4(1:xmax)),
    xlabel('Multi FMV, row lumping'),
    ylabel('Error PSD'),
    axis([0 xmax 0 max(psd4)])
eval(['print ',file])
eval(['save psds2_k',int2str(k),'_ns',int2str(ns),' psd1 psd2 psd3 psd4'])
end

```

REFERENCES

1. Dean A. Richter, "Iterative System Modeling Using Multigrid Techniques", Master's thesis, Naval Postgraduate School, Monterey, California, December 1991.
2. William L. Briggs, *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1987.
3. Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1992.
4. Roger A. Horn and Charles A. Johnson, *Matrix Analysis*, Cambridge University Press, New York, New York, 1985.
5. W. Hackbusch and U. Trottenberg, *Multigrid Methods III*, Birkhauser Verlag, Boston, Massachusetts, 1991.
6. Gene H. Golub and Charles F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Maryland, 1989.
7. Robert E. Strum and Donald E. Kirk, *First Principles of Discrete Systems and Digital Signal Processing*, Addison-Wesley Publishing Co., New York, New York, 1989.
8. James A. Cadzow, *Foundations of Digital Signal Processing and Data Analysis*, Macmillan Publishing Co., New York, New York, 1987.
9. Bernard Widrow and Samuel D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1985.
10. Simon Haykin, *Introduction to Adaptive Filters*, Macmillan Publishing Co., New York, New York, 1984.

INITIAL DISTRIBUTION LIST

	No. of Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3. Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4. Professor Murali Tummala, Code EC/Tu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	4
5. Professor Van Henson Department of Mathematics, Code MA/Vh Naval Postgraduate School Monterey, California 93943-5000	1
6. Dr. R. Madan, Code 1114SE Office of Naval Research 800 North Quincy Street Arlington, Virginia 22217-5000	1
7. Computer Sciences Branch Attn: John S. Volk III 306 E. Popson Avenue Edwards AFB, California 93524	2