AN ADVANCED SIGNAL PROCESSING TOOLKIT
FOR JAVA APPLICATIONS

By
Vijay Pravin Shah

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

December 2002
Copyright by

Vijay Pravin Shah

2002
AN ADVANCED SIGNAL PROCESSING TOOLKIT
FOR JAVA APPLICATIONS

By

Vijay Pravin Shah

Approved:

Nicholas H. Younan
Professor of Electrical and Computer Engineering
(Graduate Program Director and Director of Thesis)

Lori Mann Bruce
Assistant Professor of Electrical and Computer Engineering
(Committee Member)

Anthony Skjellum
Associate Professor of Computer Science
(Committee Member)

A. Wayne Bennett
Dean of the College of Engineering
The aim of this study is to examine the capability, performance, and relevance of a signal processing toolkit in Java, a programming language for Web-based applications. Due to the simplicity, ease and application use of the toolkit and with the advanced Internet technologies such as Remote Method Invocation (RMI), a spectral estimation applet has been created in the Java environment. This toolkit also provides an interactive and visual approach in understanding the various theoretical concepts of spectral estimation and shows the need to create more application applets to better understand the various concepts of signal and image processing.

This study also focuses on creating a Java toolkit for embedded systems, such as Personal Digital Assistants (PDAs), embedded Java board, and supporting integer precision, and utilizing COordinate Rotation DIgital Computer (CORDIC) algorithm, both aimed to provide good performance in resource-limited environments. The results
show a feasibility and necessity of developing a standardized Application Programming Interface (API) for the fixed-point signal processing library.
DEDICATION

I would like to dedicate this research to my parents, Pravin and Chandraprabha Shah, and my sister Bhumika.
ACKNOWLEDGEMENTS

The author would like to thank his major professor, Dr. Nicholas H. Younan, for his guidance and support for the successful completion of this research work. Thanks to Dr. Anthony Skjellum for his invaluable advice for completion of this thesis and serving as a thesis committee member. The author would also like to thank Dr. Lori Mann Bruce for her willingness to be on the thesis committee and for providing useful knowledge during the graduate program. Also, thanks to Mr. Torey Alford and Mr. David Whitcomb of MPI software Technology, Inc. for their help in developing the application toolkit. Thanks to the National Science Foundation for supporting this research work.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDICATION</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2. Toolkit</td>
<td>2</td>
</tr>
<tr>
<td>II. JAVA</td>
<td>4</td>
</tr>
<tr>
<td>2.1. Evolution of Java</td>
<td>4</td>
</tr>
<tr>
<td>2.2. What is Java?</td>
<td>5</td>
</tr>
<tr>
<td>2.3. Java Programs</td>
<td>7</td>
</tr>
<tr>
<td>2.4. Java Native Interface</td>
<td>7</td>
</tr>
<tr>
<td>2.5. Remote Method Invocation</td>
<td>8</td>
</tr>
<tr>
<td>III. inAspect LIBRARY</td>
<td>11</td>
</tr>
<tr>
<td>3.1. Design of inAspect</td>
<td>11</td>
</tr>
<tr>
<td>3.2. Performance of inAspect</td>
<td>16</td>
</tr>
<tr>
<td>IV. SPECTRAL ESTIMATION APPLICATION</td>
<td>18</td>
</tr>
<tr>
<td>4.1. Classical Estimation</td>
<td>18</td>
</tr>
<tr>
<td>4.1.1. Periodogram</td>
<td>19</td>
</tr>
<tr>
<td>4.1.2. Correlogram</td>
<td>19</td>
</tr>
<tr>
<td>4.1.3. Average Periodogram</td>
<td>20</td>
</tr>
<tr>
<td>4.2. Parametric Estimation</td>
<td>22</td>
</tr>
<tr>
<td>4.2.1. Autoregressive Estimation (AR)</td>
<td>23</td>
</tr>
<tr>
<td>4.2.2. Moving Average Estimation (MA)</td>
<td>29</td>
</tr>
<tr>
<td>CHAPTER</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>4.2.3.</td>
<td>Autoregressive-Moving Average Estimation (ARMA)</td>
</tr>
<tr>
<td>4.3.</td>
<td>Statistical Estimation</td>
</tr>
<tr>
<td>4.3.1.</td>
<td>Recursive Maximum Likelihood Estimate</td>
</tr>
<tr>
<td>4.3.2.</td>
<td>Minimum Variance</td>
</tr>
<tr>
<td>4.4.</td>
<td>Spectral Estimation Applet Design</td>
</tr>
<tr>
<td>4.5.</td>
<td>Screen Shots of the JAVA Applet</td>
</tr>
<tr>
<td>V.</td>
<td>APPLICATION FOR HANDHELD DEVICES</td>
</tr>
<tr>
<td>5.1.</td>
<td>Cordic Algorithm</td>
</tr>
<tr>
<td>5.1.1.</td>
<td>CORDIC Iteration in Rotation Mode</td>
</tr>
<tr>
<td>5.1.2.</td>
<td>CORDIC Iteration in Vector Mode</td>
</tr>
<tr>
<td>5.2.</td>
<td>Fixed – Point Signal Processing</td>
</tr>
<tr>
<td>5.2.1.</td>
<td>Fixed-Point Fast Fourier Transform (FFT)</td>
</tr>
<tr>
<td>5.2.2.</td>
<td>Fixed-Point Real-Complex Fast Fourier Transform (RCFFT)</td>
</tr>
<tr>
<td>5.2.3.</td>
<td>Fixed-Point Autocorrelation</td>
</tr>
<tr>
<td>5.3.</td>
<td>Fixed-Point Classical Spectral Estimation</td>
</tr>
<tr>
<td>5.3.1.</td>
<td>Periodogram</td>
</tr>
<tr>
<td>5.3.2.</td>
<td>Correlogram</td>
</tr>
<tr>
<td>5.3.3.</td>
<td>Average Periodogram</td>
</tr>
<tr>
<td>5.4.</td>
<td>Result and Screen Shots of MIDlet</td>
</tr>
<tr>
<td>5.4.1.</td>
<td>Screen Shots of the Input Module of MIDlet</td>
</tr>
<tr>
<td>5.4.2.</td>
<td>Screen Shots of the Output Module</td>
</tr>
<tr>
<td>VI.</td>
<td>CONCLUSION AND FUTURE WORK</td>
</tr>
<tr>
<td>6.1.</td>
<td>Summary</td>
</tr>
<tr>
<td>6.2.</td>
<td>Future Work</td>
</tr>
<tr>
<td>REFERENCES</td>
<td></td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Different Editions of Java Platform</td>
</tr>
<tr>
<td>2.2</td>
<td>Role of JNI</td>
</tr>
<tr>
<td>2.3</td>
<td>RMI Architecture Overview</td>
</tr>
<tr>
<td>3.1</td>
<td>Programming Model</td>
</tr>
<tr>
<td>3.2</td>
<td>Layout of inAspect</td>
</tr>
<tr>
<td>3.3</td>
<td>Hierarchy of Block Class</td>
</tr>
<tr>
<td>3.4</td>
<td>Example for Creating a Specific View from a Block</td>
</tr>
<tr>
<td>3.5</td>
<td>Concept of Queuing Operation</td>
</tr>
<tr>
<td>3.6</td>
<td>Performance Comparison of Aspect with native Code</td>
</tr>
<tr>
<td>3.7</td>
<td>Performance Comparison of pure Java FFT to inAspect FFT</td>
</tr>
<tr>
<td>4.1</td>
<td>Functional Block Diagram for Classical PSD Estimation</td>
</tr>
<tr>
<td>4.2</td>
<td>Functional Block Diagram for PSD Estimation for AR model</td>
</tr>
<tr>
<td>4.3</td>
<td>Functional Block Diagram for PSD Estimation using MA model</td>
</tr>
<tr>
<td>4.4</td>
<td>Functional Block Diagram for PSD Estimation using ARMA model</td>
</tr>
<tr>
<td>4.5</td>
<td>Functional Block Diagram for Statistical PSD Estimation</td>
</tr>
<tr>
<td>4.6</td>
<td>Main Applet Structure</td>
</tr>
<tr>
<td>4.7</td>
<td>RMI Model Structure</td>
</tr>
<tr>
<td>4.8</td>
<td>Layout of Spectral Estimation Package</td>
</tr>
<tr>
<td>FIGURE</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>4.9</td>
<td>Screen Shot of the Java Applet.</td>
</tr>
<tr>
<td>4.10</td>
<td>An Actual Screen-shot of the User – Input Module from the JAVA Applet.</td>
</tr>
<tr>
<td>4.11</td>
<td>An Actual Screen-shot of the User – Input Module for the ARMA Model.</td>
</tr>
<tr>
<td>4.12</td>
<td>Sketch of Input Signal.</td>
</tr>
<tr>
<td>4.13</td>
<td>An Actual Screen-shot of the Display Module Illustrating the Results obtained from Classical PSD Estimation.</td>
</tr>
<tr>
<td>4.14</td>
<td>An Actual Screen-shot of the Display Module Illustrating the Results obtained from Parametric Estimation using Autoregressive Model.</td>
</tr>
<tr>
<td>4.15</td>
<td>An Actual Screen-shot of the Display Module Illustrating the Results obtained from Parametric Estimation using Moving Average Model.</td>
</tr>
<tr>
<td>4.16</td>
<td>An Actual Screen-shot of the Display Module Illustrating the Results obtained from Parametric Estimation using Autoregressive Moving Average Model.</td>
</tr>
<tr>
<td>4.17</td>
<td>An Actual Screen-shot of the Display Module Illustrating the Results obtained from Statistical Estimation.</td>
</tr>
<tr>
<td>5.1</td>
<td>Flow graph for a Single Butterfly.</td>
</tr>
<tr>
<td>5.2</td>
<td>Screen shots at the input.</td>
</tr>
<tr>
<td>5.3</td>
<td>Screen shots at the output.</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

1.1. Background

The introduction of Java by Sun Microsystems has brought revolution into the computing world. Java is the most effective programming language for web-based applications. Java technology is not only limited to the workstations and servers, but it also targets a wide range of platforms such as pagers, hand-held devices, cellular phone, screen phones, digital set-top boxes, and car navigation systems.

Standard signal processing libraries, such as Vector Signal and Image processing library (VSIPL) in C [1], Signal Processing in C++ (SPUC) [2], IEEE DSP program (McMaster) in Fortran [3] are available. With these libraries, why should a toolkit be developed for Java? Although Java is considered to be relatively slow for computational type applications, it has several advantages such as simplicity, object-oriented programming, portability, robustness, security, exception handling capability, Internet-application language, simplifying the multiprocessing tasks of a signal processing system using Remote Method Interface (RMI), and last but not least its usefulness in developing a teaching toolkit for signal processing applications. Current available packages in Java are customized, distinct, and do not allow an easy creation of specific new applications
for signal processing [4]. Having a standard Java toolkit simplifies the problems of developing new applications as well as improving the current applications. The remainder of this chapter discusses the background of this study, its objective, and an overview of the inAspect toolkit.

1.2. Toolkit

Supported by the National Science Foundation (NSF), this study in conjunction with MPI Software Technology examines the viability, performance, and applications of a signal processing toolkit in Java. The creation of a Java applet demonstrates the functionality of the toolkit for developing advanced signal-processing algorithms.

In 1996, a group consisting of computer scientists and signal processing experts from both academia and industry met to discuss the creation of a standard API for digital signal processing and matrix arithmetic (linear algebra). The outcome was the VSIPL, a standardized high-performance API for C [1]. VSIPL is now used worldwide.

inAspect provides the standard signal and image processing library to the Java community. All the C/C++ signal processing applications use the Vector Signal Image Processing Library (VSIPL) specifications. The inAspect package acts as a “wrapper package” around the VSIPL Library and VSI/PRO. VSI/PRO, designed by MPI-Software Technology Inc, is based on VSIPL standards and “co-optimized” with MPI/Pro®.

The development of the inAspect occurred in two phases. The first phase involved the feasibility study and various design requirements, issues, and implementation methods described by Leong, et al [5]. The study showed that through the usage of the
Java Native Interface (JNI), high-performance signal processing would be possible. The second phase involved the creation of the Java package for advanced signal processing and comparing the performance of the Java package with other standard packages using other high-level languages.

CHAPTER II

JAVA

After the introduction of Java in 1995, it has become the programming language for web-based applications. This chapter begins with the evolution of Java and discusses Java. It then explains the different types of Java programs and, at the end, provides a brief discussion about the Java Native Interface (JNI) and Remote Method Invocation (RMI). The official website of Sun Microsystems and Zukorwski discusses Java technology and the Java programming language in-depth [6-7].

2.1. Evolution of Java

In 1990, Sun Microsystems began a project called “Green” to develop software for consumer electronics such as toasters, VCRs, and PDAs. Considering its flexibility with embedded system, C++ was used for the software development. Because C++ requires the programmer to keep track of the restricted system resources on an embedded system, a programmer had a significant burden to overcome the development of reliable and portable software for consumer electronics. The solution to this problem was a new language called “Oaks”, which later in 1995, was renamed as Java. Now, a single language incorporated the best features of several other languages. Features, such explicit
resource references, pointer arithmetic, and operator overloading, were omitted considering the dangerous effects that resulted from improper use by the programmer. Considering the potential use of Oak in web technology, Sun MicroSystems developed a product known as *WebRunner*, later renamed *HotJava*. The developer community considered Java a standard for Internet development, even before the first release of the Java compiler in January 1996.

2.2. What is Java?

The ease of portability has increased the usage and popularity of Java technology. The basic idea of the Java technology-based software is to run the same application on different machines like desktops, Sun workstations, Macintoshes, consumer gadgets, and other devices. This technology has helped the Internet and private networks to become a “Computing Environment”. Java technology consists of three terms: Java Programming Language, Java Virtual Machine, and the Java Platform. Java applications, such as applets, servlets, etc., are written in the Java programming language, an object-oriented language with syntax similar to C. The Java Virtual Machine (VM), implemented in hardware or software, interprets and executes the byte code generated after the compilation of a Java program. Java programs are only portable to platforms with a ported Java interpreter. Not only Desktops but also the set-top box and version for handheld devices running PalmOS and Windows CE are available with ported Java VM. The Java Platform is different from Java programming language and Java VM. Java programs use the predefined set of available classes on every Java installation. This
A predefined set of Java Classes is known as Java Platform. Optional standard extensions help to extend the available platform. Java classes are organized by functionality in groups, known as packages, that include input/output, networking, graphics, user-interface creation, and security. Currently Java platforms are available in three different editions, as one size does not fit all. The Standard Edition of Java 2 platform, known as J2SE™ technology, is used to develop the client-side enterprise applications. This platform provides a satisfactory high-speed performance and functionality demanded by Web Users. It is the base tool for creating sophisticated and valuable applications [8]. Another component of the Java 2 platform, the Enterprise Edition (J2EE™ technology), is a modular base that simplifies the Enterprise development and deployment [9]. Also considering the memory limitations of consumer devices such as PDAs, set-top boxes, pagers, and cell-phones, Sun Microsystems has introduced the Java 2 platform, the Micro Edition (J2ME™ technology) [10]. Figure 2.1 shows different editions of the Java platform. The Java Platform is freely available from http://java.sun.com.

Figure 2.1 Different Editions of Java Platform
2.3. Java Programs

Java programs are classified as applets and standalone applications. When the program is executed within a Java-enabled web browser, running its own VM, it is called an Applet. On the other hand, it is called a Java Application if it runs independent of a web browser. An applet is loaded and executed in a web browser by including the \texttt{<APPLET>} HTML tag in the HTML file. For executing a standalone application, the Java development kit (JDK) interpreter must be installed. The limitations of the applet are that it cannot read files from the originating universal resource locator (URL) and the transportation is via HTTP. Exception is if the originator gives the permission to read files from the local computer. This makes it slow compared to standalone applications that are executed locally by the interpreter. Detail discussion of writing an applet or standalone application is provided in [6,7]. The lower performance of Java programs is improved by using the Remote Method Invocation, a feature that allows calling a method from a program in one Java VM to another Java VM located in a remote server.

2.4. Java Native Interface

The Java Native Interface (JNI) provides the standard interface to integrate native code (written using C, C++, or assembly) into a Java code [11]. JNI acts as the “glue” between native and Java applications. Figure 2.2 shows the role of JNI in the interfacing of native-side and Java-side applications.
Figure 2.2 Role of JNI

JNI saves the time of rewriting high performance applications that already exist in native language. The native implementations are compiled into a dynamic link library (known shared objects (.so) in Solaris and dynamic link libraries (.dll) in Win32). The operating system loads and links this library into the process that is running the Java Virtual Machine.

2.5. Remote Method Invocation

The Remote Method Invocation facilitates writing an application to distribute computing across the networking environment. RMI takes a step further in object-oriented design by selecting the appropriate machine for performing a specific task [7]. The RMI architecture consists of three layers: the stubs/skeleton layer, the remote reference layer (RRL), and the transport layer. Figure 2.3 shows the relationships among these layers [12].
The role of the stubs/skeletons layer is maintaining a connection between the objects of client-servers. The stub represents the client-side proxy of a remote object and is referred as a local object by the program running on the client side. The skeleton interfaces with the server side RRL and it reads any arguments sent to the remote object to call the actual object implementation on the server side.

The remote reference layer (RRL) provides the data stream to stubs and skeletons, and it deals with the transport layer. It thus manages the “liveliness” and communication between the client/server.

The transport layer is the layer that creates and maintains the actual connection between client and server, sending information over wire. The four concepts of the transport layer are:

- An endpoint, specific to transport instance, references the address space, which contains a Java Virtual Machine.
A channel is the pathway between the two address spaces. It manages any connection between client and server.

A connection defines the concept of transferring data (argument and return values) between client and server.

Transport sets up a channel between a local address space and a remote endpoint and accepts the incoming connections to the address space containing the abstraction.

Zukowski describes five steps to create an application accessible to remote clients [7]

- Create the Interface definition for the remote classes.
- Create the Interface implementation for the remote class.
- Create stub and skeletons.
- Create and compile server and client applications.
- Start the RMI registry and server applications.
CHAPTER III

inAspect LIBRARY

The Internet Enabled Signal and Image Processing Library (inAspect) is a package that provides high-performance portable signal and image processing functionality to the Java community [13]. The package provides also an interface for Java developers as well as maintains standards within the digital signal processing community. This chapter has a brief discussion about the design of inAspect, its performance, and issues with the library.

3.1. Design of inAspect

inAspect has been designed to provide a portable, high-performance library for signal processing in Java. Considering the fact that Java is computationally slow, inAspect uses native libraries on supported platforms. Currently, it provides the interface to the native library developed by VSIPL and VSI/PRO [1,14]. The programming model of inAspect using JNI is shown in Figure 3.1, which also shows the utility and importance of the VSIPL-VSI/PRO native library in the Aspect package.
inAspect follows the standard of VSIPL, keeping the API similar to that of VSIPL when feasible. Figure 3.2 shows the basic layout of inAspect.
Sample, Block, and View are the base classes for storage and operation of the user data. Sample and its subclass provide the support for storing and retrieving the real and complex values of different primitive data types, including double, float, integer, and Boolean. Block is the basic unit of the inAspect package and the concept of allocating continuous memory similar to a “block” of VSIP, L. The data type specific classes ComplexBlock and RealBlock extend the Block class. These classes are extended according to the available primitive data types, double and float. Currently Java does not support complex data types. Hence, the ComplexBlock is formed by interleaved format or split format ways of storing complex values in arrays of primitives. In the interleaved format, all real and imaginary values of complex numbers are placed one after another in the single array to form an array of 2n size for n complex numbers. In the split format, two arrays for the real and imaginary elements are formed. inAspect uses the interleaved format internally. Figure 3.3 shows the class hierarchy diagram for the Block class.

![Block Class Hierarchy Diagram]

Figure 3.3 Hierarchy of Block Class
View is the base class for the manipulation of data in the inAspect package. VectorView, MatrixView, and TensorView extend View to represent a form that is more specific. VectorView, MatrixView, and TensorView represent the block in one-dimensional, two-dimensional, and three-dimensional form respectively. A User can attach more than one view to a single block and can manipulate either part of or the whole block using the Views attached to that block or by rebinding the whole set of data to the block. Figure 3.4 shows an example of creating a different type of view from a real block of size 25.

For an example consider a real block of size = 25

| 0 | 1 | 2 | 3 | 4 | 5 | …………… | 20 | 21 | 22 | 23 | 24 |

Let us form a Vector View with

- offset = 3 (4th element of the block)
- length = 6 (number of elements it can access from a given block)
- stride = 3 (space between consecutive element in vector view (3, 6, 9, ...))

The Resultant view is:

| 3 | 6 | 9 | 12 | 15 | 18 |

Let us form a Matrix View from same block with

- Offset = 0 (1st element of the block)
- xlength = 5 (number of columns)
- ylength = 5 (number of rows)
- xStride = 1 (space between consecutive column elements)
- yStride = 1 (space between consecutive row elements)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

The Resultant view is:

Figure 3.4 Example for Creating a Specific View from a Block
TensorView considers attributes for the z-direction i.e. length and stride, including the attributes used in the creation of MatrixView.

The JNI introduces the overhead of transferring data from the native side to Java. This overhead may be large enough to efface the high performance obtained using the native approach. With this issue in mind, inAspect uses the concept of queuing the operations. All the operations to be performed on data are queued in the queue object, eliminating the need for switching between C and Java for each operation. All the operations are performed using the single native call. Figure 3.5 illustrates the concept of Queuing [13].

Figure 3.5 Concept of Queuing Operation
3.2. Performance of inAspect

The average performance of the inAspect package for computing 1024-point Complex-Complex FFT, performed one thousand times in order, attains sufficient accuracy on a system with an Intel Pentium III 850MHz with 384M RAM and Windows 2000 OS. Figure 3.6 shows a comparison between the inAspect package using the VSIPL VSI/Pro library, and a native implementation of the native library. This figure clearly shows the introduction of overhead by JNI.

![Figure 3.6 Performance Comparison of Aspect with native Code](image-url)
Figure 3.7 shows the poor performance of Java performing a FFT. This poor performance is due to the overhead and extra checking that Java performs during execution.

![Performance Comparison of pure Java FFT to inAspect FFT](image)

In order to maintain an acceptable level of performance, a new algorithm design is required for pure Java implementation.
CHAPTER IV

SPECTRAL ESTIMATION APPLICATION

One topic of interest in signal processing applications is spectral analysis, a process that involves the estimation of the spectral content, i.e. the distribution of power over frequency, of a time series data from a finite set of measurements. Traditional techniques, based solely on Fourier transformation, are mainly used for relating the time domain representation of a signal to its frequency domain. In addition, since most signals have noisy or random components, statistics play a major role in characterizing these signals. Accordingly, robust and reliable signal processing algorithms are needed to improve the overall performance, such as better frequency or spectral resolution and higher signal detection ability. The algorithms, developed using the inAspect package and identified for this task, are grouped as Classical Estimation (Non-Parametric Method), Parametric Estimation, and Statistical Estimation.

4.1. Classical Estimation

Classical estimation relies totally on estimating the power spectral density (PSD) of time series data via the discrete Fourier transform (DFT) [15,16]. A fast way to implement the DFT is the well-known fast Fourier transform (FFT). This transform relationship is considered as a nonparametric description of the second order moment of
the time series data. The most commonly used classical spectral estimators are the Periodogram, Correlogram, and Average Periodogram.

4.1.1. Periodogram

The periodogram is obtained by a statistical averaging of the squared magnitude of the Fourier transform of a given data set divided by the data length. Accordingly, for a given data set \(x(n) = \{x(0), x(1), \ldots, x(N-1)\}\), the PSD estimate of \(x(n)\) based on the periodogram method is defined as:

\[
\hat{P}_{PER}(f) = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j2\pi fn} \hat{\rho}^2.
\]  

(4.1)

4.1.2. Correlogram

The correlogram is obtained from the estimate of the correlation function and then Fourier transforms it. Accordingly,

\[
\hat{P}_c(f) = \sum_{k=-\lfloor(N-1)/2\rfloor}^{\lfloor(N-1)/2\rfloor} \hat{r}[k] e^{-j2\pi fk}
\]  

(4.2)

where \(\hat{r}(k)\) denotes an estimate of the correlation \(r(k)\), obtained from the available sample \(x(n)_{n=1}^N\). Assuming stationarity, the standard way of obtaining the sample correlation required in (4.2) is:

\[
\hat{r}[k] = \frac{1}{N} \sum_{n=0}^{N-k-1} x^*[n]x[n+k], \quad 0 \leq k \leq N - 1
\]  

with

\[
\hat{r}[k] = \hat{r}^*[k], \quad -(N-1) \leq k \leq -1
\]  

(4.3)
where “*” denotes a complex conjugate.

### 4.1.3. Average Periodogram

In order to improve the statistical properties of the Periodogram, assuming K independent data records are available, each over the interval $0 \leq n \leq L-1$, and all the realizations of the same random process. The data are grouped as

$$\{x_0[n], 0 \leq n \leq L-1; \ x_1[n], 0 \leq n \leq L-1; \ldots; x_{K-1}[n], 0 \leq n \leq L-1\}$$

Then, the average Periodogram estimator is defined as

$$\hat{P}_{AVPER}(f) = \frac{1}{K} \sum_{m=0}^{K-1} \hat{P}^{(m)}_{PER}(f). \quad (4.4)$$

where $\hat{P}^{(m)}_{PER}(f)$ is the Periodogram of the $m$th data set given by

$$\hat{P}^{(m)}_{PER}(f) = \frac{1}{N} \left| \sum_{n=0}^{N-1} x_m[n] e^{-j 2\pi f n} \right|^2 \quad (4.5)$$

Figure 4.1 shows the functional block diagram for estimating the PSD using classical estimation methods. It also demonstrates the application of the queuing concept of the inAspect package mentioned previously in Figure 3.5.
Create required Blocks, Views, FFT Object, and Correlation Object
(Required Parameters: Method, Data Type (Float or Double), Real or Complex Data)

Create Object of Classical Estimation
(Required Parameters: Window Length, Window Type, Parameters for FFT Object)

Call Methods of Classical Estimation
(Required Parameters: Input View, Output View, Process Queue)

Data queued to obtain correlation sequence

Correlated sequence queued to perform windowing

Windowed correlated sequence, queued to perform FFT

Compute PSD in dB

Process the queue

Estimate PSD for new Data?

Queue each Segment to compute PSD by Periodogram method

Correlogram

Data queued to perform windowing

Windowed Data queued to perform FFT

Compute Average PSD in dB

j = 0
Number of Segments = m

j < m

increment j

j < m

No

Yes

Rebind new Data to Input View

Yes

No

Destroy all the created Blocks, Views, FFT Object, and Correlation Object
(Required Parameter: Method)

Note:

Data is in Queue, user can not modify queued data.

Steps for estimating PSD.

Data block can be modified.

Figure 4.1 Functional Block Diagram for Classical PSD Estimation
4.2 Parametric Estimation

Parametric estimation relies on the assumption that the second order moment describing the time series data fits a rational model. Accordingly,

\[
\sum_k a(k)x(n-k) + \sum_k b(k)u(n-k) = \sum h(k)u(n-k)
\]

(4.6)

where \(x(n)\) is the output of a linear time invariant system with an impulse response of \(h(n)\), \(u(n)\) is the corresponding input, and \(b(0) = a(0) = 1\). In the z-domain, this can be expressed as

\[
H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{k=0}^{q} b(k)z^{-k}}{1 + \sum_{k=1}^{p} a(k)z^{-k}}
\]

(4.7)

where \(B(z)\) and \(A(z)\) are polynomials of orders \(q\) and \(p\) respectively and whose coefficients describe the model parameters. These coefficients represent the zeros and poles of \(H(z)\). The procedure to parametric modeling involves three steps:

**Step 1:** Select an appropriate model that fits the data.

**Step 2:** Obtain an estimate of the model parameters.

**Step 3:** Describe the PSD estimate in terms of the model parameters and noise variance.

In general, parametric modeling achieves better PSD estimates with higher resolution. This class of spectral estimation includes the autoregressive model (AR), the moving average model (MA), and the autoregressive-moving average model (ARMA).
4.2.1. Autoregressive Estimation (AR)

The AR model can be described directly from the general model of (4.7) by setting $q$ equal to zero. Accordingly, the AR model transfer function is $H(z) = \frac{1}{A(z)}$ and is known as an all-pole model. The PSD estimate of an AR model is obtained from

$$\hat{P}_{AR}(f) = \frac{\hat{\sigma}^2}{|A(f)|^2}$$

(4.8)

where $\hat{\sigma}^2$ is an estimate of the noise variance. Several techniques are available to obtain an estimate of the model parameters of an AR model. Below is a brief description of commonly used methods.

4.2.1.1. Autocorrelation Method

The model parameters are obtained by minimizing the prediction error power given by

$$\hat{\rho} = \frac{1}{N} \sum_{n=-\infty}^{\infty} \left| x[n] + \sum_{k=1}^{p} a[k] x[n-k] \right|^2$$

(4.9)

where the unavailable samples of $x[n]$, i.e., not in the range of $0 \leq n \leq N-1$, are set to zero. Using a complex gradient, i.e., differentiating the prediction error given in (4.9) with respect to the $a[k]$’s yields

$$\frac{1}{N} \sum_{n=-\infty}^{\infty} \left( x[n] + \sum_{k=1}^{p} a[k] x[n-k] \right) x^*[n-l] = 0 \quad l = 1, 2, \ldots, p. \tag{4.10}$$

In matrix form, this can be written as
\[
\begin{bmatrix}
\hat{r}_{xx}[0] & \hat{r}_{xx}[-1] & \ldots & \hat{r}_{xx}[-(p-1)] \\
\hat{r}_{xx}[1] & \hat{r}_{xx}[0] & \ldots & \hat{r}_{xx}[-(p-2)] \\
\vdots & \vdots & \ddots & \vdots \\
\hat{r}_{xx}[p-1] & \hat{r}_{xx}[p-2] & \ldots & \hat{r}_{xx}[0]
\end{bmatrix}
\begin{bmatrix}
\hat{a}[1] \\
\hat{a}[2] \\
\vdots \\
\hat{a}[p]
\end{bmatrix}
= -
\begin{bmatrix}
\hat{r}_{xx}[1] \\
\hat{r}_{xx}[2] \\
\vdots \\
\hat{r}_{xx}[p]
\end{bmatrix}
\] 
(4.11)

where

\[
\hat{r}_{xx}[k] = \frac{1}{N} \sum_{n=0}^{N-k-1} x'[n]x[n + k], \quad 0 \leq k \leq p
\]
with
\[
\hat{r}_{xx}[k] = \hat{r}_{xx}^*[k], \quad -(p - 1) \leq k \leq -1.
\]

The estimation of the white noise variance \( \hat{\sigma}^2 \) is obtained directly from \( \hat{\rho}_{\text{MIN}} \), i.e.

\[
\hat{\sigma}^2 = \hat{r}_{xx}[0] + \sum_{k=1}^{p} \hat{a}[k] \hat{r}_{xx}[-k].
\] 
(4.13)

### 4.2.1.2 Covariance Method

For this method, the estimated prediction error power, given by

\[
\hat{\rho} = \frac{1}{N-p} \sum_{n=p}^{N-1} \| x[n] + \sum_{k=1}^{p} \hat{a}[k] x[n-k] \|^2,
\] 
(4.14)

is minimized to obtain an estimate of the AR parameters. Note that all the data points are needed to compute the prediction error power. Similar to the autocorrelation method, the minimization is obtained by applying the complex gradient and the AR parameter are obtained as the solution of
where

\[ c_{xx}[j,k] = \frac{1}{N - p} \sum_{n=p}^{N-1} x'[n-j]x[n-k] \]  

(4.16)

and the estimation of the white noise variance \( \hat{\sigma}^2 \) is

\[ \hat{\sigma}^2 = c_{xx}[0,0] + \sum_{k=1}^{p} \hat{a}[k]c_{xx}[0,k]. \]  

(4.17)

4.2.1.3 Modified Covariance Method

The modified covariance method is based on minimizing the average of the estimated forward and backward prediction error powers of the AR parameters. Accordingly,

\[ \hat{\rho} = \frac{1}{2} (\hat{\rho}^f + \hat{\rho}^b) \]  

(4.18)

where

\[ \hat{\rho}^f = \frac{1}{N - p} \sum_{n=p}^{N-1} \left| x[n] + \sum_{k=1}^{p} a[k]x[n-k] \right|^2 \]  

(4.19)

\[ \hat{\rho}^b = \frac{1}{N - p} \sum_{n=0}^{N-1-p} \left| x[n] + \sum_{k=1}^{p} a[k]x[n+k] \right|^2. \]
The model coefficients can then be obtained from the formulation in (4.15) by replacing $c_{xx}[j,k]$ in (4.16) with

$$c_{xx}[j,k] = \frac{1}{2(N - p)} \left( \sum_{n=p}^{N-1} x^*[n - j]x[n - k] + \sum_{n=0}^{N-1-p} x^*[n + j]x[n + k] \right) (4.20)$$

### 4.2.1.4 Burg Method

The Burg method does not estimate the AR parameter directly (as in the autocorrelation, covariance, and modified covariance methods). First, the reflection coefficients are calculated and then the Levinson algorithm is used to obtain the AR parameters. The iterative steps for the Burg method are as follows:

**Step 1:** Initial Conditions

$$\hat{r}_{xx}[0] = \frac{1}{N} \sum_{n=0}^{N-1} |x[n]|^2$$  

(4.21)

$$\hat{\rho}_0 = \hat{r}_{xx}[0], \hat{x}_0^f = x[n], 0 \leq n \leq N - 1$$  

(4.22)

$$\hat{x}_0^b = x[n], 0 \leq n \leq N - 2.$$  

(4.23)

**Step 2:** Solve for the reflection coefficients

For $0 \leq k \leq p$,

$$\hat{k}_k = \frac{-2 \sum_{n=k}^{N-1} \hat{e}_{k-1}^f[n] \hat{e}_{k-1}^b[n - 1]^*}{\sum_{n=k}^{N-1} \left( |\hat{e}_{k-1}^f[n]|^2 + |\hat{e}_{k-1}^b[n - 1]|^2 \right)}$$  

(4.24)
\[
\hat{\rho}_k = \left(1 - \left| \hat{\rho}_k \right|^2 \right) \hat{\rho}_{k-1}
\]  
(4.25)

\[
\hat{a}_k[i] = \begin{cases} 
\hat{a}_{k-1}[i] + \hat{k}_k \cdot \hat{a}_{k-1}^*[i], & 1 \leq i \leq k - 1 \\
\hat{k}_k, & i = k 
\end{cases}
\]  
(4.26)

If \( k = 1 \), \( \hat{a}[1] = \hat{k}_1 \)

\[
\hat{e}^f_k[n] = \hat{e}^f_{k-1}[n] + \hat{k}_k \cdot \hat{e}^b_{k-1}[n-1], \quad k + 1 \leq n \leq N - 1
\]  
(4.27)

\[
\hat{e}^b_k[n] = \hat{e}^b_{k-1}[n-1] + \hat{k}_k \cdot \hat{e}^f_{k-1}[n], \quad k \leq n \leq N - 2
\]

**Step 3:** Obtain the model parameters

The model parameter estimates are given as the set of \( \left\{ \hat{a}_p[1], \hat{a}_p[2], \ldots, \hat{a}_p[p], \hat{\rho}_p \right\} \).

Figure 4.2 show a functional block diagram for estimating the PSD using the autoregressive model.
Create Object of Autoregressive Estimation
(Required Parameters: Window Length, Window Type, AR order, Optional Parameters for FFT Object)

Create Blocks, View and required objects
(Required Parameters: Method, Data Type (Float or Double), Real or Complex Data)
If method is:
1. Autocorrelation: Create Correlation Object, Equation Solver Object
2. Covariance and Modified Covariance Method: Create Cholesky Object
Create Object of Autoregressive Estimation
(Required Parameters: Window Length, Window Type, AR order, Optional Parameters for FFT Object)

Estimate PSD or AR Parameter?

Estimate PSD

Create FFT Object

If estimating PSD, data queued to perform windowing. Otherwise skip it

Now data available for estimation?

Yes

No

Rebind new Data to Input View

Process the queue

Yes

No

Perform FFT on AR Parameters

Compute PSD in dB

Steps for estimating PSD.

Initialize the forward and backward error Coefficient, variance equal to autocorrelation Value at lag zero.

Solve the Reflection Coefficient, to obtain AR Parameter and noise Variance

Yes

No

Increment j

AR parameter Estimated by Cholesky Decomposition

Calculate the extra term of Eq. 4.16, using correlation function

Yes

No

Increment j

modified method

j = 1

Correlation Sequence queued, AR parameter and noise variance estimated, using Levinson Algorithm

Windowed data queued to obtain correlation sequence

Initialise the forward and backward error Coefficient, variance equal to autocorrelation Value at lag zero.

j = 0

Elements of jth column, of upper triangular part of Covariance Matrix and jth row of RHS side of eq 4.15 are calculated using correlation function.

Modified method

Yes

No

Increment j

Steps for estimating PSD.

Data is in Queue, user can not modify queued data.

Data block can be modified.

Figure 4.2 Functional Block Diagram for PSD Estimation for AR model
4.2.2. Moving Average Estimation (MA)

The MA model can be described directly from the general model of (4.7) by setting $p$ equal to zero. Accordingly, the MA model transfer function is $H(z) = B(z)$ and is known as an all zero model. The PSD estimate of an MA model is obtained from

$$\hat{P}_{Ma}(f) = \hat{\sigma}^2 |B(f)|^2$$

(4.28)

where $\hat{\sigma}^2$ is an estimate of the noise variance. Several techniques are available to obtain an estimate of the model parameters of an MA model. The most commonly used is the Durbin’s method.

4.2.2.1. Durbin Method

The Durbin method is based on estimating the MA(q) process by from an AR(L) process, where $q \ll L \ll N$. That is, the MA(q) process described by

$$x[n] = \sum_{k=0}^{q} b[k] u[n-k]$$

is equivalent to the AR($\infty$) process described by

$$x[n] = -\sum_{k=1}^{\infty} a[k] x[n-k] + u[n].$$

Note that the AR(L) process will be a good approximation of the MA(q) process as the impulse response of $1/B(z)$ has decayed to zero. Thus, instead of considering the likelihood function of the data directly, the likelihood function of the AR parameter estimates is considered. If $\hat{\theta}$ is the corresponding parameter estimate of an AR (L) model obtained by the autocorrelation method, then the MA filter parameters are obtained from
\[ \hat{b} = -\hat{R}_{aa} \hat{r}_{aa} \]  

where,

\[
\begin{align*}
[\hat{R}_{aa}]_{ij} &= \frac{1}{L+1} \sum_{\nu=0}^{L+i+j} \hat{a}[\nu] \hat{a}[\nu+i-j], \quad 1 \leq i, j \leq q \\
[\hat{r}_{aa}]_{ij} &= \frac{1}{L+1} \sum_{\nu=0}^{L+i+j} \hat{a}[\nu] \hat{a}[\nu+j], \quad 1 \leq i, j \leq q \\
\hat{b} &= \{ \hat{b}[1], \hat{b}[2], \ldots, \hat{b}[q] \}.
\end{align*}
\]  

Figure 4.3 shows a functional block diagram for estimating the PSD using the moving average model.
Create Object of Moving Average Estimation
(Required Parameters: Window Length, Window Type, MA order, Optional Parameters for FFT Object)

Create Blocks, View and AR Estimation Object
(Required Parameters: Method, Data Type (Float or Double), Real or Complex Data)

Call Method of Moving Average Estimation
(Required Parameters: Input View, Output View, Process Queue)

If estimating PSD, data queued to perform windowing, Otherwise skip it

Windowed data queued to estimate AR parameter of large order (data length / 5) using Autocorrelation Method

Estimated AR parameter queued, to estimate MA Parameter using Autocorrelation Method

Estimating PSD ?

Yes

Perform FFT on AR and MA Parameters

No

Process the queue

Compute PSD in dB

Yes

New data available for estimation ?

No

Rebind new Data to Input View

Destroy all the created Blocks, Views, created Objects
(Required Parameter: Method)

Note:
- Data is in Queue, user can not modify queued data.
- Data block can be modified.

Steps for estimating PSD:

Figure 4.3 Functional Block Diagram for PSD Estimation using MA model.
4.2.3. Autoregressive-Moving Average Estimation (ARMA)

The ARMA model is one that satisfies (4.6) and (4.7) respectively. Accordingly, the PSD estimate of an ARMA model is obtained from

\[ \hat{P}_{\text{ARMA}}(f) = \frac{\hat{\sigma}^2 |B(f)|^2}{|A(f)|^2}. \] (4.31)

Several suboptimal procedures are available to estimate the ARMA parameters. The most commonly used are the ones that are based on making separate estimates of the AR and MA parameters. The steps involved in this process are as follows: first the AR parameters are estimated using any of the techniques presented earlier. Then, the AR parameters estimate is used to construct an inverse filter that is applied directly to the original time-series data, and finally, the residuals of the inverse filter are representative of an MA process to which the MA parameter estimator can be applied.

4.2.3.1 Yule-Walker Least Square Estimation

In general, the autocorrelation function of an ARMA(p,q) process satisfies the following relationship,

\[ r_x[k] = \sum_{l=1}^{p} a[l]r_x[k-l], \quad k \geq q+1. \] (4.32)

If the autocorrelation function is accurately estimated, then for \( k = q+1, q+2, \ldots, q+p \), the AR parameters can be obtained directly from...
\[
\begin{bmatrix}
    r_{xx}[q+1] \\
    r_{xx}[q+2] \\
    \vdots \\
    r_{xx}[M]
\end{bmatrix}
= -
\begin{bmatrix}
    r_{xx}[q] & r_{xx}[q-1] & \ldots & r_{xx}[q-p+1] \\
    r_{xx}[q+1] & r_{xx}[q] & \ldots & r_{xx}[q-p+2] \\
    \vdots & \vdots & \ddots & \vdots \\
    r_{xx}[M-1] & r_{xx}[M-2] & \ldots & r_{xx}[M-p]
\end{bmatrix}
\begin{bmatrix}
    a[1] \\
    a[2] \\
    \vdots \\
    a[p]
\end{bmatrix}
\]

or
\[
r = -Ra
\]  

(4.33)

where \( R \) is the \((M-1) \times p\) correlation matrix. Thus, assuming \( M-q > p \) and making use of the autocorrelation estimate, equation (4.33) becomes
\[
\hat{r} = -\hat{R}\hat{a} + e.
\]  

(4.34)

Note that if \( e = 0 \), then the estimation is equal to the theoretical value. Accordingly, the AR parameters can be obtained via the least square technique, i.e.,
\[
\hat{a} = -\left(\hat{R}^H\hat{R}\right)^{-1}\hat{R}^H\hat{r}
\]  

(4.35)

where
\[
\hat{R} =
\begin{bmatrix}
    \hat{r}_{xx}[q] & \hat{r}_{xx}[q-1] & \ldots & \hat{r}_{xx}[q-(p-1)] \\
    \hat{r}_{xx}[q+1] & \hat{r}_{xx}[q] & \ldots & \hat{r}_{xx}[q-(p-2)] \\
    \vdots & \vdots & \ddots & \vdots \\
    \hat{r}_{xx}[M-1] & \hat{r}_{xx}[M-2] & \ldots & \hat{r}_{xx}[M-p]
\end{bmatrix}
\]  

and
\[
\hat{r} = [\hat{r}_{xx}[q+1] \hat{r}_{xx}[q+2] \ldots \hat{r}_{xx}[M]]^T.
\]  

Finally, the MA parameters are obtained using the Durbin Algorithm.
4.2.3.2. Yule-Walker Levinson Estimation

The AR parameters are first obtained by solving

\[
\begin{bmatrix}
\hat{r}_{xx}[q] & \hat{r}_{xx}[q-1] & \ldots & \hat{r}_{xx}[q-(p-1)] \\
\hat{r}_{xx}[q+1] & \hat{r}_{xx}[q] & \ldots & \hat{r}_{xx}[q-(p-2)] \\
\vdots & \vdots & \ddots & \vdots \\
\hat{r}_{xx}[M-1] & \hat{r}_{xx}[M-2] & \ldots & \hat{r}_{xx}[M-p]
\end{bmatrix}
\begin{bmatrix}
\hat{a}[1] \\
\hat{a}[2] \\
\vdots \\
\hat{a}[p]
\end{bmatrix} =
\begin{bmatrix}
\hat{r}_{xx}[q+1] \\
\hat{r}_{xx}[q+2] \\
\vdots \\
\hat{r}_{xx}[q+p]
\end{bmatrix}
\]  

(4.37)

using the Levinson recursion algorithm. Once the AR parameters are obtained, the MA parameters are recursively calculated according to the following procedure:

**Step 1:** Initialization

\[
a_1[1] = -\frac{r_{xx}[q+1]}{r_{xx}[q]} , \quad b_1[1] = -\frac{r_{xx}[q-1]}{r_{xx}[q]} , \quad \text{and } \rho_1 = (1 - a_1[1]b_1[1])r_{xx}[q]
\]  

(4.38)

**Step 2:** for \( k = 2, 3, \ldots, p \), compute

\[
a_k[k] = -\frac{r_{xx}[q+k] + \sum_{l=1}^{k-1} a_{k-1}[l]r_{xx}[q+k-l]}{\rho_{k-1}}
\]

(4.39)

\[
a_k[i] = a_{k-1}[i] + a_k[k]b_{k-1}[k-i], 1 \leq i \leq k-1
\]

If \( k = p \), then exit function

Else, continue to do,

\[
b_k[k] = -\frac{r_{xx}[q-k] + \sum_{l=1}^{k-1} b_{k-1}[l]r_{xx}[q-k-l]}{\rho_{k-1}}
\]

(4.40)

\[
b_k[i] = b_{k-1}[i] + b_k[k]a_{k-1}[k-i], 1 \leq i \leq k-1
\]

\[
\rho_k = (1 - a_k[k]b_k[k])\rho_{k-1}.
\]

(4.41)

Figure 4.4 show a functional block diagram for estimating the PSD using the autoregressive moving average model.
Create object of ARMA Estimation
(Required Parameters: Window Length, Window Type, AR order, MA order Optional Parameters for FFT Object)

Create Blocks, View, Correlation object, MA Estimation object
FIR object and other required objects
(Required Parameters: Method, Data Type (Float or Double), Real or Complex Data)

Call Method of ARMA Estimation
(Required Parameters: Input View, Output View, Process Queue)

If estimating PSD, data queued to perform windowing, Otherwise skip it

Windowed data queued to obtain correlation sequence

Yule-Walker Least Square

Yule-Walker Levinson

If Method is

Create Correlation Matrix.

Cholesky Decomposition of Correlation Matrix to estimate AR Parameter and noise variance

Estimated AR parameter used as FIR filter coefficient to filter input Data

Estimated MA parameter From filtered input Data, using Durbin Method

Correlation Sequence queued, AR parameter and noise variance estimated, solving eq. by General Levinson Algorithm

Perform FFT on AR and MA Parameters

Yes

No

Estimating PSD ?

Process the queue

Compute PSD in dB

New data available for estimation?

Yes

Rebind new Data to Input View

Data is in Queue, user can not modify queued data.

Data block can be modified.

Steps for estimating PSD.

Figure 4.4 Functional Block Diagram for PSD Estimation using ARMA model
4.3. Statistical Estimation

Statistical Estimation is based on the statistical properties of the time-series data. Statistical estimation methods are known to be more robust techniques for estimating deterministic parameters. The most commonly used is the maximum likelihood estimation.

4.3.1. Recursive Maximum Likelihood Estimate

The recursive maximum likelihood estimate (RMLE) method is mainly based on maximizing the exact likelihood function of an AR(1) process and then using this estimate to generate a higher order AR model via the Levinson recursion method. The likelihood function is maximized with respect to all the reflection coefficients. The procedure for implementing this method is as follows:

**Step 1:** Define

\[
\begin{align*}
   \alpha'_{k-1} &= [1 \ \hat{a}_{k-1}[1] \ \hat{a}_{k-1}[2] \ldots \ \hat{a}_{k-1}[k-1]]' \\
   \beta'_{k-1} &= [\hat{a}_{k-1}[k-1] \ \hat{a}_{k-1}[k-2] \ldots \ \hat{a}_{k-1}[1] \ 1]' 
\end{align*}
\]

(4.42)

\[C_k \text{ and } D_k\] are defined as the \(k \times k\) partitions of \(S\), i.e.,

\[
[C_k]_{ij} = S_{i-1,j}, 1 \leq i, j \leq k \\
[D_k]_{ij} = S_{i,j}, 1 \leq i, j \leq k \\
\]

where \(S\) is \((p+1) \times (p+1)\) with elements

\[
S_{ij} = \sum_{n=0}^{N-1-i-j} x[n+i]x[n+j], 0 \leq i, j \leq p \\
\]

(4.43)
Step 2: Initialize

\[ \epsilon_0 = S_{00}, \ c_i = S_{01}, \ d_i = S_{11}, \]

Step 3: For \( k = 1 \), compute

\[ \hat{k}_1 \] by solving

\[ k_1^3 + \frac{(N-2)c_i}{(N-1)d_i} k_1^2 - \frac{\epsilon_0 + Nd_i}{(N-1)d_i} k_1 - \frac{Nc_i}{(N-1)d_i} = 0 \]  (4.44)

where a single root within \([-1,1]\) is selected and

\[ \hat{\epsilon}_1 = \hat{k}_1, \ \epsilon_1 = S_{00} + 2\hat{k}_1 S_{01} + \hat{k}_1^2 S_{11}, \ \hat{\rho}_1 = \frac{1}{N} \epsilon_1 \]  (4.45)

Step 4: For \( k = 2, 3, \ldots, p \), compute

\[ c_k = a_{k-1}^T C_k b_{k-1}^T \]  (4.46)

\[ d_k = b_{k-1}^T D_k b_{k-1}^T \]

and \( \hat{k}_k \) is obtained by solving

\[ k_k^3 + \frac{(N-2)c_k}{(N-1)d_k} k_k^2 - \frac{k \epsilon_{k-1} + Nd_k}{(N-k)d_k} k_k - \frac{Nc_k}{(N-k)d_k} = 0 \]  (4.47)

where a single root within \([-1,1]\) is selected. For two or more root within this interval, select one that maximizes

\[ \left( 1 - k_k^2 \right)^{k/2} \]

\[ \left[ \frac{1}{N} \left( \epsilon_{k-1} + 2c_k k_k + d_k k_k^2 \right) \right]^{N/2} \]  (4.48)

updating the coefficients,
4.3.2. Minimum Variance

The minimum variance (MV) method estimates the spectral content of a time series data by minimizing the variance of the output narrowband filter. The resulting estimate obtained does not represent the total power of the process. Hence, it is not a true PSD function [15]. If the autocorrelation sequence is known, then using the AR parameter, the MV parameters can be obtained [15].

An appropriate spectral estimator given by the minimum variance method is given as

\[
P_{MV}(f) = \frac{T}{\mathbf{e}(f) \mathbf{R}_p^{-1} \mathbf{e}(f)},
\]

where,
\[
\mathbf{e}(f) = \begin{bmatrix} 1 \\ e^{j2\pi f T} \\ \vdots \\ e^{j2\pi p T} \end{bmatrix}
\]

and

\[
\mathbf{R}_p = \begin{bmatrix} r_{ss}[0] & \cdots & r_{ss}[p] \\ \vdots & \ddots & \vdots \\ r_{ss}[p] & \cdots & r_{ss}[0] \end{bmatrix}
\]

\(T = \text{Sample Interval.}\)
The relationship between the AR and MA parameters is given by equation (4.51)

\[ \frac{1}{P_{MV}(p, f)} = \sum_{k=0}^{p} \frac{1}{P_{AR}(k, f)}. \]  

(4.51)

The PSD estimator for the $p$th order AR model is given by

\[ P_{AR}(f) = \frac{T \rho_p}{1 + \sum_{k=1}^{p} a_p[k] e^{-j2\pi kT}} \]  

(4.52)

Expressing equation (4.52) in an alternative form, we have

\[ P_{AR}(f) = \frac{T \rho_p}{\sum_{k=-p}^{p} \psi_{AR}[k] e^{-j2\pi kT}} \]  

(4.53)

where,

\[ \psi_{AR}[k] = \begin{cases} \frac{1}{\rho_p} \sum_{i=0}^{k} a_p[k+i] a_p^*[i] & \text{for } 0 \leq k \leq p, \\ \psi_{AR}^*[k] & \text{for } -p \leq k \leq -1. \end{cases} \]

Substituting $R_p^{-1}$ in equation (4.50), by

\[ R_p^{-1} = \frac{1}{\rho_p} T_p T_p^H - \frac{1}{\rho_p} S_p S_p^H \]  

(4.54)

where,

\[ T_p = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ a_p[1] & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_p[p-1] & a_p[p-2] & \ldots & 1 & 0 \\ a_p[p] & a_p[p-1] & \ldots & a_p[1] & 1 \end{pmatrix} \]

and

\[ S_p = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ a_p[1] & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_p[p-1] & a_p[p-2] & \ldots & 1 & 0 \\ a_p[p] & a_p[p-1] & \ldots & a_p[1] & 1 \end{pmatrix} \]
and solving equation (4.50), the spectral estimation is given by

$$\mathbf{S}_p = \begin{pmatrix}
0 & 0 & \ldots & 0 & 0 \\
\alpha_p^*[p] & 0 & \ldots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\alpha_p^*[p-1] & \alpha_p^*[p-2] & \ldots & 0 & 0 \\
\alpha_p^*[p] & \alpha_p^*[p-1] & \ldots & \alpha_p^*[1] & 0
\end{pmatrix}$$

where

$$P_{MV} (f) = \frac{TP_p}{\sum_{k=-p}^{p} \psi_{MV}[k] e^{-j2\pi kfT}}$$

(4.55)

That is, the \( \psi_{MV}[k] \) can be obtained from the AR parameters by linearly weighting (windowing) its correlation sequence.

Figure 4.5 shows a functional block diagram for estimating the PSD using statistical estimation methods.
Create object of Statistical Estimation
(Required Parameters: Window Length, Window Type, AR order, time Interval, Optional Parameters for FFT Object)

PSD Parameter

Estimate PSD or Parameters ?

Create Blocks, View, Correlation object, AR Estimation object
FIR object and other required objects
(Required Parameters: Method, Data Type (Float or Double), Real or Complex Data)

Call Method of Statistical Estimation
(Required Parameters: Input View, Output View, Process Queue)

If estimating PSD, data queued to perform windowing. Otherwise skip it

Create the Matrix S (Eq. 4.43) and j=0;

RMLE Minimum Variance

Find the Coefficient Required in Cubic Equation (4.44)

Solve Cubic equation, and set reflection coefficient equal to the roots of cubic equation maximizing the likelihood function.

Yes

Increment j:

j<AR order

No

Update Likelihood function

Estimating PSD?

Yes

Perform FFT on AR and MA Parameters

No

Process the queue

Yes

New data available for estimation?

No

Rebind new Data to Input View

Destroy all the created Blocks, Views, created Objects
(Required Parameter: Method)

Note:
- Data is in Queue, user can not modify queued data.
- Data block can be modified.

Steps for estimating PSD.

Figure 4.5 Functional Block Diagram for Statistical PSD Estimation
4.4. Spectral Estimation Applet Design

The class inaspectSpectrumAnalysis extends the JApplet class and uses spectImp to call library functions that are bundled in the SpectralEstimation package. The graphical interface and parsing of various functions are done using the graph package. Figure 4.6 shows the class relationships for the main applet, where

- Class inaspectSpectrumAnalysis: is the main applet class. It is the client application.
- Class inaspectInputPanel: provides functions to obtain input from the user.
- Class inaspectSignalPanel: displays the input and output results.
- Class armaParameters: provides a dialog box for entering various parameters required by the AR, MA, ARMA and Statistical Methods.

Figure 4.6 Main Applet Structure.
The RMI structure for the applet is shown in Figure 4.7 with the followings:

- **Class spectEst**: This class provides the interface for calculating the power spectral density, depending upon the selected method.
- **Class spectEstImp**: This class provides the implementation of the interface defined in spectEst.
- **Class spectEstImp_Stub/spectEstImp_Skel**: These classes help in maintaining the connection between the objects of client applets and servers.
- **Class spectServ**: It is the server application and helps binding the server name to the object spectEstImp.

![Figure 4.7 RMI Model Structure](image-url)
The Layout of the SpectralEstimation package is shown in Figure 4.8, where

- Class SpectralEstimation: is the base class for all the other classes except EquationSolver.
- Class EquationSolver: solves a linear system of equations using the Levinson recursion and the general Levinson algorithm, and it finds the roots of the cubic equation.
- Class ClassicalEstimation: implements the classical spectral estimation methods, i.e., Periodogram, Correlogram, and Average Periodogram.
- Class AREstdimation: implements the autocorrelation, covariance, modified covariance, and Burg methods for the parametric AR spectral estimation.
- Class MAEstimation: implements the Durbin method for the parametric MA spectral estimation.
- 45 -

- Class ARMAEstimation: implements the Yule-Walker method using the Levinson algorithm and the least squares methods for the parametric ARMA spectral estimation.
- Class StatisticalEstimation: implements the recursive maximum likelihood and Minimum Variance method for the statistical spectral estimation.
- Class WindowData: performs windowing on the input data.

4.5. Screen Shots of the JAVA Applet

The main JAVA applet is divided into three modules; the user-input module, the computational module, and the display module. Figure 4.9 illustrates an actual screen-shot of the JAVA applet. A brief description of each module follows:

User Input Module: The user-input module is divided into menus and it lets the user select different methods, signal type, and window type from the Method menu, Signal menu, and Window menu respectively. The user can draw or clear the spectrum plot from the Spectrum menu. Furthermore, the user is able to input the various parameters of a signal, such as sampling frequency, number of samples, FFT length, window size, and overlap length for the Average Periodogram. A separate dialog box is provided to enter the model orders for the AR, MA, and ARMA methods. Figure 4.10 illustrates various screen-shots of the user-input module and Figure 4.11 shows a screen-shot of additional user-input for the ARMA Model.
Figure 4.9 Screen Shot of the Java Applet.

Figure 4.10 An Actual Screen-shot of the User – Input Module from the JAVA Applet
Computational Module: A function that is entered by the user in the signal generation box is parsed and the signal is sampled at the user-defined frequency. Instead of generating a signal, the user can directly load the signal from an external file by selecting the File menu item from the Signal menu. The following parameters are required for estimating the power spectral density: signal data, window type, sampling frequency, FFT size, window size, overlap-size (for the of average periodogram), AR order, MA order, and algorithm type. These parameters are then passed to the server (spectServ). The server estimates the PSD using the Spectral Estimation package. The results are returned to the client. The Spectral Estimation package, used for computation purpose, implements different methods for estimating the spectral content of a time series data as discussed earlier.

Display Module: This module displays the given signal and outputs the corresponding PSD in dB. Figure 4.12-Figure 4.17, shows the actual screen-shots of the display module. The output results are obtained for a set of 1500 data samples that consists of a 50 Hz sinusoid with unit amplitude and additive white noise of unit variance (SNR = -3 dB), sampled at 800 Hz, using rectangular window, and taking 1024 point FFT.
Figure 4.12 Sketch of Input Signal

(a) Periodogram PSD estimates
(b) Cross-Lagogram PSD estimates
(c) Average Periodogram PSD estimates,
Overlap = 512

Figure 4.13 An Actual Screen-shot of the Display Module Illustrating the Results obtained from Classical PSD Estimation.
(a) Autocorrelation Method PSD estimates, AR order = 10
(b) Covariance Method PSD estimates, AR order = 10
(c) Modified Covariance PSD estimates Method, AR order = 10
(d) Burg PSD estimates, AR order = 10.

Figure 4.14 An Actual Screen-shot of the Display Module Illustrating the Results obtained from Parametric Estimation using Autoregressive Model.

(a) Durbin PSD estimates, MA order = 20

Figure 4.15 An Actual Screen-shot of the Display Module Illustrating the Results obtained from Parametric Estimation using Moving Average Model.
(a) Yule-Walker, Least Square Estimation PSD estimates, AR Order = 7, MA order = 4, Number of Yule-Walker equation = 15

(b) Yule-Walker PSD estimates, Levinson Method, AR order = 7, MA order = 4

Figure 4.16 An Actual Screen-shot of the Display Module Illustrating the Results obtained from Parametric Estimation using Autoregressive Moving Average Model.

(a) Recursive Maximum Likelihood PSD estimate, AR order = 10

(b) Minimum Variance PSD estimates, AR order = 10

Figure 4.17 An Actual Screen-shot of the Display Module Illustrating the Results obtained from Statistical Estimation
CHAPTER V
APPLICATION FOR HANDHELD DEVICES

The JAVA 2 Micro Edition platform provides the solution for creating applications on embedded devices. Currently, the J2ME API does not support floating-point operations and native interface. Hence, fixed-point signal processing algorithms must be developed for the J2ME API. Algorithm development in higher languages uses floating-point operations. However, due to certain J2ME limitations, efficient algorithms with fixed-point operations need to be developed. One-way to achieve this is to use efficient hardware algorithms, which, in turns, use shift-add operations [17]. Vector rotation by arbitrary angles using shifts and add operation can be performed by the COordinate Rotation DIgital Computer (CORDIC) algorithm as described in [17,18]. As mentioned earlier, traditional techniques for spectral estimation solely depend on Fourier transformation, and an efficient fixed-point FFT algorithm, for example, can speed up the process of implementing such algorithms on hand-held devices.

5.1. Cordic Algorithm

The COordinate Rotation DIgital Computer algorithm is first described by Jack E. Volder and can be useful in calculating trigonometric functions, polar to rectangular conversion, rectangular to polar conversion, and vector magnitude [17,18]. It is an
iterative method of performing vector rotation by required angles using shift-and-add operations. An algorithm can be derived from the general rotation transform. For example, a \([x,y]\) vector can be rotated by a certain angle \(\theta\) to form new vector \([x',y']\). Hence

\[
x' = x\cos\theta - y\sin\theta \quad \text{and} \quad y' = y\cos\theta + x\sin\theta.
\] (5.1)

Rearranging the above equation gives

\[
x' = \cos\theta(x - y\tan\theta) \quad \text{and} \quad y' = \cos\theta(y + x\tan\theta)
\] (5.2)

As suggested in [17], if the rotation of the vector is limited to \(\tan\theta = \pm 2^i\), a multiplication operation is simply replaced by a shift operation. This iterative process requires a certain number of precision bits to perform smaller rotations and, at each iteration, a decision to rotate a vector in a clockwise or counterclockwise direction is made. Hence,

\[
x_{i+1} = K_i[x_i - y_i \cdot d_i \cdot 2^{-i}] \quad \text{and} \quad y_{i+1} = K_i[y_i + x_i \cdot d_i \cdot 2^{-i}],
\]

where

\[
K_i = 1/\sqrt{1+2^{-2i}} \quad \text{and} \quad d_i = \pm 1.
\]

For each iteration, the angle should be accumulated and hence a third equation is added to the CORDIC algorithm. That is, \(z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})\). Removing the scale constant from the iterative equation and storing the value of \(\tan^{-1}(2^{-i})\) in a small lookup table, this set of equations involves only shift and add operations. Note that the CORDIC rotator operates only in “rotation” and “vectoring” modes.
5.1.1. CORDIC Iteration in Rotation Mode

In the rotation mode, the CORDIC iterations rotate a given vector by an arbitrary angle. For this mode, the set of equations used are:

\[
x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}
\]

(5.3)

\[
y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}
\]

(5.4)

and

\[
z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})
\]

(5.5)

where

\[
d_i = \begin{cases} 
1, & \text{if } z_i \geq 0 \\
-1, & \text{otherwise}
\end{cases}
\]

Accordingly, the following equations are obtained:

\[
x_n = A_n[x_0 \cos \theta - y_0 \sin \theta]
\]

\[
y_n = A_n[y_0 \cos \theta + x_0 \sin \theta]
\]

(5.6)

\[
z_n = 0 \text{ and } A_n = \Pi \sqrt{1 + 2^{-2i}}
\]

In the rotation mode, the sine and cosine of the required angle are found and a polar-to-rectangular transform is performed.

Application to find Rectangular Coordinates from Polar form:

From the previous equations, it can be seen that having \(x_0 = r, y_0 = 0, \) and \(\theta = \text{polar phase, gives}

\[
x_n = r \cos \theta \text{ and } y_n = r \sin \theta.
\]

Note that the above equation is obtained after compensating for the gain resulting from performing a vector rotation (Multiplication by 0.6073 for the 12-bit precision).
**Application to find Sine and Cosine:**

To obtain the sine and cosine values of a given angle,

initialize $x_0 = 1/A_n$, $y_0 = 0$, and $\theta =$ required angle and substitute this value in equation (5.6). This results in

$x_n = \cos \theta$ and $y_n = \sin \theta$.

**5.1.2. CORDIC Iteration in Vector Mode**

When operating in the vector mode, the input vector is rotated by some angle such that the output vector is aligned with the x-axis [17]. For the vectoring mode, the set of equations remains the same as in equations (5.3), (5.4), and (5.5) with

$$d_i = \begin{cases} 1, & \text{if } y_i < 0 \\ -1, & \text{otherwise} \end{cases}$$

Accordingly,

$$x_n = A_n \sqrt{x_0^2 + y_0^2}$$

$$y_n = 0$$

$$z_n = z_0 + \tan^{-1} \left(\frac{y_0}{x_0}\right) \text{ and } A_n = n \sqrt{1 + 2^{-2i}}. \quad (5.7)$$

Using the vectoring mode of operation, we can directly find the arctangent, vector magnitude, and polar transformation from the Cartesian coordinates.
Application to Find Vector Magnitude:

From equation 5.3, it can be seen that after scaling $x_n$ by the gain $A_n$ gives

$$x_n = \sqrt{x_0^2 + y_0^2},$$

i.e. the magnitude component is equal to the x-component of the rotated vector [17].

Application to find Polar form from Rectangular Coordinate:

Setting $z_0=0$ and scaling $x_n$ by the gain $A_n$ in equation (5.4) yields the following:

$$r = x_n = \sqrt{x_0^2 + y_0^2}$$

(5.8)

and

$$z_n = \tan^{-1}\left(\frac{y_0}{x_0}\right)$$

(5.9)

Application to find Arctangent:

The arctangent can be directly obtained by setting $z_0=0$ and performing CORDIC vector rotation. The arctangent is given by

$$z_n = \tan^{-1}\left(\frac{Y_0}{X_0}\right)$$

5.2. Fixed – Point Signal Processing

5.2.1. Fixed-Point Fast Fourier Transform (FFT)

The fast Fourier transform (FFT) is a fast technique that computes the N-point discrete Fourier transform (DFT) of a time series data using the decimation in frequency scheme (DIF). Note that although the decimation in time (DIT) is commonly used to
implement the FFT, the number of angle shifts used in the decimation in frequency scheme is half the number of angle shifts in the decimation in time scheme. Furthermore, if the length of the windowed data is not an integer power of 2, then the concept of zero padding is used to extend the data to the required length. A flow graph for the single butterfly FFT is shown in Figure 5.1.

![Flow graph for a Single Butterfly](image)

The following equations can be obtained from this butterfly diagram:

\[
\text{Output(Upper)} = \text{Input(Upper)} + \text{Input(lower)} \quad (5.10)
\]

and

\[
\text{Output(Lower)} = (\text{Input(Upper)} - \text{Input(lower)}) \cdot W^l \quad (5.11)
\]

where \( W = e^{j2\pi l/N} \), with \( N \) being the length. From equations (5.5) and (5.6), it can be seen that, for an \( N \)-point radix-2 FFT algorithm, the number of additions required is \( N \cdot \log N \) and the number of complex multiplications is \( N/2 \cdot \log N \). In Equation (5.11), we have a multiplication of a complex vector with the unity vector with an angle \( W \). As described in [18], a multiplication operation can be replaced by a vector rotation according to the CORDIC method. Thus, a total of \( N/2 \cdot \log N \) CORDIC rotation is required for an \( N \)-
point FFT. As the CORDIC rotation is performed by a shift-add, the implemented FFT algorithm uses only shift and addition operations.

### 5.2.2. Fixed-Point Real-Complex Fast Fourier Transform (RCFFT)

For a real input data $x_n$, $n = 0, 1, \ldots, N-1$, in order to perform the N-point FFT, the imaginary part of the complex data is set to zero. Performing FFT on this data introduces redundancy in the FFT transform, thus increasing the unnecessary computational time. In order to avoid this redundancy, we can reorder the real data sequence into a complex data sequence of half the original length, having the real part as an even-index of the original data and the imaginary part as an odd-index of the original data [19]. Accordingly,

$$(5.12)\quad cx_n = x_n + jx_{2n+1}, \quad n = 0, 1, \ldots, N/2-1,$$

where $cx_n$ denotes the complex data representation of given signal $x_n$.

Performing FFT on this complex data reduces the computation by half. Accordingly, the output is

$$(5.13)\quad CX_n = X_n^e + iX_n^o, \text{ for } n = 0, 1, \ldots, N-1$$

with

$$(5.14)\quad X_n^e = (\text{Re}(CSum) + \text{Im}(Csum) * \cos \theta - \text{Re}(CDiff) * \sin \theta)$$

and

$$(5.15)\quad X_n^o = (\text{Im}(CDiff) - \text{Im}(Csum) * \sin \theta - \text{Re}(CDiff) * \cos \theta)$$

where,

$$CSum = (CX_n + CX_{N/2-n})/4$$

$$CDiff = (CX_n - CX_{N/2-n})/4$$
\[ \theta = 2\pi n/N, \ n=1, 2, ..., N-1 \]

and

\[ X_0 = \frac{(X_n^e + X_n^o)}{2} \]

From the above equations, it can be seen that this recombination introduces extra computations (2N multiplications, 2N additions, and N complex addition). However, this computation is reduced to N/2 additions, N complex additions, and an N/2 CORDIC rotation of the vector \([\text{Im}(\text{CSum}), \text{Re}(\text{CDiff})]\) by angle \(\theta\).

5.2.3. Fixed-Point Autocorrelation

The biased autocorrelation estimation of the available samples \(x(n)_{n=1}^N\), assuming stationarity, is given by:

\[
\hat{r}[k] = \frac{1}{N} \sum_{n=0}^{N-1} x^*[n]x[n+k], \quad 0 \leq k \leq N-1
\]

with \(\hat{r}[k] = \hat{r}^*[k], \quad -(N-1) \leq k \leq -1\) (5.15)

For a 32-bit integer with 12 bit of precision, there is always a chance that a summation after a multiplication will result in an overflow for small values. Since the number of samples divides this summation, we can simply substitute multiplication and division by adding and subtraction in the logarithmic mode. Equation (5.15) can be rewritten as

\[
\hat{r}[k] = \sum_{n=0}^{N-1} \text{sgn} \cdot e^{-\log(|x[n]|) + \log(|x[n+k]|) - \log(N)}, \quad 0 \leq k \leq N-1
\]

with \(\hat{r}[k] = \hat{r}^*[k], \quad -(N-1) \leq k \leq -1\) (5.15)

where \(\text{sgn} = \begin{cases} -1, & \text{if } x \text{ or } \{x[n], x[n+k]\} < 0 \\ 0, & \text{if } x[n] = 0 \text{ or } x[n+k] = 0 \\ 1, & \text{otherwise} \) for real data
Note that for the complex data, the phase can be obtained by adding the phase of two complex vectors. For the real data, the sign of the product depends on the input data. For the unbiased autocorrelation estimation, $\log(N)$ is simply replaced by $\log(N-k)$.

5.3. Fixed-Point Classical Spectral Estimation

Making use of the fixed-point fast Fourier transform, classical spectral estimation methods can be implemented on hand-held devices. The procedure for estimating the power spectral density based on the Periodogram, Correlogram, and Average Periodogram classical estimators follows.

5.3.1. Periodogram

Procedure:

- Data windowing.
- Calculate the N-point Real–Complex FFT or N-point Complex–Complex FFT of the windowed data.
- If the absolute value of the FFT is zero, ceil it to a minimum bit value and if it is less than zero, which indicates an overflow, set the value to a maximum value. If we have 12 bit of precision, then the highest possible value for a 32-bit integer in J2ME is $2^{19}$.
- Calculate the power spectral density (PSD) in dB.
5.3.2. Correlogram

Procedure:

- Perform a biased autocorrelation of the input data.
- Window the correlated data with lags Window.
- Calculate the N-point real – complex FFT or Complex-Complex FFT of the windowed data.
- If the absolute value of FFT is zero, ceil it to a minimum bit value and if it is less than zero, which indicates an overflow, set the value to a maximum value.
- Calculate the PSD values in dB.

5.3.3. Average Periodogram

Procedure:

- Select a segment from the data, depending on the selected Overlap Size.
- Calculate the Periodogram for each segment.
- Calculate the average Periodogram over all segments.

5.4. Result and Screen Shots of MIDlet

The JAVA 2MicroEdition Wireless toolkit was used to develop MIDlet for the classical spectral estimation. MIDlet was tested on the palm-pilot Emulator and also on the emulator of other 32-bit devices like RIM Java Handheld, Motorola_i85, and Default Color Phone of J2MEWTK.
5.4.1. Screen Shots of the Input Module of MIDlet

The user-input module is divided into two sub-modules: Configuration and Input Parameters. The Configuration sub-module consists of a “Method choice group”. It is used to select one from the three classical methods. The “Signal choice group” allows the selection of a pure sinusoid and a noisy sinusoid with uniform or normally distributed noise, and the “Window choice group” allows the user to select a window type from a menu that includes Rectangular, Hanning, Hamming, Bartlett, and Blackman windows. The Input Parameters sub-module allows the user to input the frequency and amplitude of a sinusoid, sampling frequency, number of samples, FFT length, and window size. Figure 5.2 shows the screen shots of the input.

![Configuration sub-module](image1)

![Input Parameter sub-module](image2)

Figure 5.2: Screen shots at the input
5.4.2. **Screen Shots of the Output Module**

Let the input signal be a noisy sinusoid with a sinusoidal frequency of 50 Hz, amplitude of 5 units, and additive white noise of unit variance (SNR = 11 dB). The input signal is sampled at 1000 Hz. The resulting power spectral density estimate is illustrated in Figure 5.3. Note that a rectangular window of length 128 is applied to the 500 samples of the input signal.

![Screen shots of the output](image)

(a) Periodogram of noisy sinusoid

(b) Correlogram of noisy sinusoid

(c) Average Periodogram of noisy sinusoid with an overlap Size of 10

Figure 5.3 Screen shots at the output
CHAPTER VI
CONCLUSION AND FUTURE WORK

6.1. Summary

Various independent and customized signal processing packages in Java are available, but developing a new application from these packages is limited. Hence, the development of a signal processing toolkit provides more usability and better performance. inAspect™ provides the high performance Signal Processing and Image Processing toolkit in Java. This high performance is attributed to JNI and the concept of queuing objects. Performances and comparisons show that the functions in the toolkit are slower than the functions in C languages. This is mainly due to additional overhead introduced by JNI.

This study first discusses the introduction and the development of Java as a programming language. Various aspects of Java Language like Java Native Interface (JNI) and Remote Method Interface (RMI) are studied in order to show the benefit of having the Signal processing toolkit in Java and applying it in the toolkit. Currently Java is the evolving language, but it is backwardly compatible. i.e. no need of rewriting the existing program code.

The inAspect toolkit is based on the standards specified by the Vector Signal Image Processing Forum. These standards are specified to provide a common operating
environment for signal processing applications and for different signal processing hardware vendor. The creation of a spectrum analysis applet using the inAspect package shows the ease of using this toolkit for the development of various signal processing applications. This applet will be useful as a teaching toolkit for understanding the concept of spectrum analysis.

The study also shows that, by using fixed-point operations, it is possible to implement various signal processing algorithms on handheld devices. The results obtained with fixed-point algorithms are compared with the ones obtained by Matlab, and an accuracy of up to 2 decimal places is easily obtained. For a 32-bit device, using a precision of 12 bits, a power spectral density in the range of 114 dB to –72 dB can be estimated.

6.2. Future Work

It will be of interest to develop standard fixed-point signal processing libraries in higher-level languages for embedded platforms, using the maximum available bits for a specific device, benchmarking various algorithms on embedded devices, and developing a better GUI for the MIDlet.

Currently all Java platforms do not support the use of Java Native Interface. Hence, the optimized algorithms for signal processing within Java (i.e. not using JNI) that maintain the standard specified by VSIPL need to be developed.

Other research goal will be the incorporation of multi-threading in the spectrum analysis applet to make it real time and improve its performance, the development of
additional application applets to understand the concept of signal and image processing, and the creation of an online tool analogous to Matlab for performing various signal and image processing functions.
REFERENCES


