FINAL REPORT
VOLUME 1
DESIGN AND IMPLEMENTATION
OF A SPEECH CODING ALGORITHM
AT 9600 B/S

Department of
ELECTRICAL ENGINEERING
UNIVERSITY OF NOTRE DAME, NOTRE DAME, INDIANA
FINAL REPORT
VOLUME 1
DESIGN AND IMPLEMENTATION
OF A SPEECH CODING ALGORITHM
AT 9600 B/S

Prepared for

Defense Communications Agency
Defense Communications Engineering Center
1860 Wiehle Avenue
Reston, Virginia 22090

Contract No. DCA 100-79-C-0005

30 April 1980

Distribution is unlimited.
**Title:** Design and Implementation of a Speech Coding Algorithm at 9600 B/S

**Authors:** James L. Melsa, et al.

**Performing Organization:** Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556

**Contract:** DCA 100-79-C-0005

**Security Class:** Unclassified

**Abstract:**

This report describes a speech coding algorithm for digital transmission of speech at a rate of 9600 bits per second and the implementation of this algorithm on a speech processing system. The algorithm combines:

- Pitch extraction loop
- Pitch compensating adaptive quantizer
- Sequentially adaptive linear predictor
- Adaptive source coding

**Distribution Statement:**

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.
to generate very high quality speech output. Although each of these elements has been previously applied to speech coding, the combination of all four of these elements has not been studied before. The speech coding algorithm has been implemented on a pair of CSPI MAP 300 Array Processors in real-time in the full-duplex mode.

This report has been bound in two volumes. The first volume contains the narrative description of the algorithm and its development and includes Chapters 1 through 11 and Appendices A through D of the report. The second volume describes the real-time MAP implementation and includes Chapters 12 and Appendices E through G.
ABSTRACT

This report describes a speech coding algorithm for digital transmission of speech at a rate of 9600 bits per second and the implementation of this algorithm on a speech processing system. The algorithm combines

- Pitch extraction loop
- Pitch compensating adaptive quantizer
- Sequentially adaptive linear predictor
- Adaptive source coding

to generate very high quality speech output. Although each of these elements has been previously applied to speech coding, the combination of all four of these elements has not been studied before. The speech coding algorithm has been implemented on a pair of CSPI MAP 300 Array Processors in real-time in the full-duplex mode.

This report has been bound in two volumes. The first volume contains the narrative description of the algorithm and its development and includes Chapters 1 through 11 and Appendices A through D of the report. The second volume describes the real-time MAP implementation and includes Chapters 12 and Appendices E through G.
PROJECT PERSONNEL

Arvind Arora, research assistant
David L. Cohn, co-principal investigator
James M. Kresse, research assistant
James L. Melsa, principal investigator
Arun K. Pande, research assistant
Maw-lin Yeh, research assistant
# FINAL REPORT
DCA CONTRACT 100-79-C-0005

<table>
<thead>
<tr>
<th>Abstract</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

## Project Personnel

<table>
<thead>
<tr>
<th>1. Introduction and Outline of Report</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Summary of Algorithm Requirements</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Outline of Report</td>
<td>4</td>
</tr>
</tbody>
</table>

## Algorithm Description

<table>
<thead>
<tr>
<th>2. Algorithm Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Transmitter Buffer System</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Adaptive Low-Pass Filter</td>
<td>16</td>
</tr>
<tr>
<td>2.4 Pitch Extraction</td>
<td>18</td>
</tr>
<tr>
<td>2.5 Adaptive Residual Coder</td>
<td>20</td>
</tr>
<tr>
<td>2.5.1 Adaptive Predictor</td>
<td>20</td>
</tr>
<tr>
<td>2.5.2 Adaptive Quantizer</td>
<td>22</td>
</tr>
<tr>
<td>2.6 Pitched Repetition</td>
<td>26</td>
</tr>
<tr>
<td>2.7 Noiseless Source Coder</td>
<td>28</td>
</tr>
<tr>
<td>2.8 Receiver</td>
<td>34</td>
</tr>
</tbody>
</table>

## Synchronization

<table>
<thead>
<tr>
<th>3. Synchronization</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Sync. Algorithm Description</td>
<td>38</td>
</tr>
<tr>
<td>3.1.1 Synchronization Acquisitions</td>
<td>40</td>
</tr>
<tr>
<td>3.1.2 Synchronization Monitor</td>
<td>43</td>
</tr>
</tbody>
</table>

## Pitch Extraction Studies

<table>
<thead>
<tr>
<th>4. Pitch Extraction Studies</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Introduction</td>
<td>46</td>
</tr>
<tr>
<td>4.2 Pitch Extraction Algorithms</td>
<td>47</td>
</tr>
<tr>
<td>4.3 Redundancy Removal</td>
<td>54</td>
</tr>
<tr>
<td>4.4 References</td>
<td>61</td>
</tr>
</tbody>
</table>

## Tree Coding

<table>
<thead>
<tr>
<th>5. Tree Coding</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
<td>62</td>
</tr>
<tr>
<td>5.2 The (N,L) Algorithm</td>
<td>63</td>
</tr>
<tr>
<td>5.2.1 Description</td>
<td>63</td>
</tr>
<tr>
<td>5.2.2 Results</td>
<td>65</td>
</tr>
<tr>
<td>5.3 Adaptive Tree Coding</td>
<td>69</td>
</tr>
<tr>
<td>5.3.1 Description</td>
<td>69</td>
</tr>
<tr>
<td>5.3.2 Results</td>
<td>71</td>
</tr>
<tr>
<td>5.4 Conclusions and Suggestions for Further Research</td>
<td>75</td>
</tr>
<tr>
<td>5.5 References</td>
<td>77</td>
</tr>
</tbody>
</table>
6. Backward PARC
   6.1 Introduction 78
   6.2 System Structure 80
   6.3 Performance Evaluation and Parametric Studies 85
   6.4 Transmission Error Studies 90
   6.5 Conclusions 95

7. Tandem Operation
   7.1 Introduction 96
   7.2 PARC in Tandem with CVSD 97
   7.3 Effect of Background Noise on PARC Performance 100
   7.4 References 103

8. Transmission Errors
   8.1 Introduction 104
   8.2 Simulation of Transmission Error 105
   8.3 Minimizing Transmission Error Effects 108
   8.4 Conclusion 112
   8.5 Reference 113

9. Filtering
   9.1 Introduction 114
   9.2 Evaluation of Pre-emphasis 115
   9.3 Filter Selection 118
   9.4 Results 121
   9.5 Low-Pass Filtering vs. Entropy 123
   9.6 Results and Conclusions 125
   9.7 References 128
   9.8 Derivation of Filter 129

10. Buffer Control
    10.1 Introduction 136
    10.2 Overflow control 137
    10.3 Underflow control 138
    10.4 Special considerations at the receiver 139
    10.5 Conclusions and suggestions for further research 140

11. Source and Error Control Coding
    11.1 Introduction 141
    11.2 Source Coding
       11.2.1 Quantizer levels 142
       11.2.2 Side information 144
    11.3 Error Control Coding 145
    11.4 Conclusion and suggestions for further research 146

A. FORTRAN Simulation of Algorithm 147
CHAPTER 1
INTRODUCTION AND OUTLINE OF REPORT

1.1 Introduction

This report describes the results of a sixteen month effort under NASA Contract 100-79-C-0005 to develop and implement a speech coding algorithm designed to produce very good quality speech at 9600 bits per second. The technique is based on the latest developments in speech digitization and is formulated to comply with the requirements described in the statement of work.

The method is based on techniques which have been reported in the literature but which are brought together here for the first time. The system combines elements of adaptive predictive coding and ADPCM systems and is known as PARC, Pitch extraction Adaptive Residual Coder. The pitch extraction loop used in adaptive predictive coding provided the input to a sequentially adaptive predictor, using backward coefficient adaptation which forms an estimate of the pitch-reduced signal. The error in this estimate is quantized by a pitch compensating adaptive quantizer. The resulting quantizer output is coded using an adaptive source coding procedure. The source code also permits transmission of pitch information and synchronization signals.

This algorithm was implemented on a pair of CSPI MAP 300 signal processors to generate a real-time, full-duplex speech encoding system.

This algorithm has several features which are significant in a full system application. The waveform reconstruction nature of the algorithm provides excellent performance in tandem with CVSD and in the presence of background noise. If bit rates higher than 9600 b/s are permitted,
the algorithm is easily adaptive to them. For example, a 16 kb/s version of the same algorithm will differ only in the number of quantization levels and the source code algorithm. It could easily be implemented using the same software.
1.2 Summary of Algorithm Requirements

The following requirements for the speech coding algorithm have been determined from the Statement of Work.

1. The speech processing system shall operate a transmission data rate of 9600 b/s.

2. The speech processing system shall produce very high quality speech reproduction. This requirement is interpreted to mean a signal-to-noise ratio of approximately 20 db.

3. The audio bandwidth of the speech coder shall be greater than or equal to 3200 Hz.

4. The speech coder shall produce good quality speech under conditions of a random transmission bit error rate of 1 percent.

5. The speech coder shall produce intelligible speech under conditions of acoustic background noise (60 db referenced to 20 μ Newton/meter²) such as office noise.

6. The speech coder shall perform satisfactorily in tandem with a CVSD speech coder operating at a data rate of 16 kb/s. This tandem configuration shall provide speech intelligibility with minimal degradation compared with a single link of CVSD operating at 16 kb/s.

The algorithm shall be implemented on a pair of CSPI MAP 300 signal processor in real-time, full duplex mode with appropriate synchronization.
1.3 Outline of Report

Following the introductory material in this chapter, the next two chapters of the report describe the general details of the PARC algorithm. Chapter 2 describes the final form of the PARC speech digitization algorithm developed in this study.

The following nine chapters of the report describe in more depth the details of the algorithm and various studies that were made during this contract period but which do not necessarily appear in the final algorithm. Chapter 3 delineates the details of synchronization for full duplex operation. Chapter 4 describes various pitch extraction studies, while Chapter 5 describes details of the tree coding studies that were conducted. A special form of the PARC algorithm in which backward adaptation is used for the pitch extraction operation is described in Chapter 6. The operation of the algorithm in tandem with CVSD is discussed in Chapter 7, while Chapter 8 is concerned with transmission studies. The final algorithm uses an adaptive filter on the input speech to improve subjective performance. This algorithm is described in Chapter 9. Chapter 10 is concerned with the buffer control algorithms used in the PARC system, while Chapter 11 is concerned with source and error control coding for full-duplex operation.

Chapter 12 describes the real-time implementation of the algorithm on the MAP processor. The remainder of the report is a series of appendices which describe various details of the programming of the algorithms in both Fortran and on the MAP, as well as various support packages.
CHAPTER 2
ALGORITHM DESCRIPTION

2.1 Introduction

This chapter will describe the final form of the PARC speech digitization algorithm developed during this study. The algorithm will be decomposed into its constituent elements and each element will be described in turn. The underlying theory will be discussed and the details of the recommended implementation will be presented. Later chapters will describe the real-time implementation and the various studies which led to the recommended form of the algorithm.

The PARC digital speech communication system can be represented as shown in Fig. 2.1. The analog speech signal \( s(t) \) is converted to a sequence of finely quantized samples \( s(k) \). These quantized samples are stored in a buffer whose delay \( B_1 \) can vary over time. The PARC transmitter section does the actual data reduction to produce the quantizer level sequence. This is represented by the \( q(k-B_1) \). This sequence, along with the side information quantized \( \beta \) and \( T \), is noiselessly encoded into the bit stream \( b(m) \) for transmission on the channel.

At the receiver end, the process is essentially reversed. The bit stream \( b'(m) \) is decoded into the sequence \( q'(k-B_1) \) and the side information \( \beta' \) and \( T' \). The primes are used to allow for channel errors. The PARC receiver device converts this information into reconstructed speech \( \hat{s}(k-B_1) \). This is buffered with a variable delay \( B_2 \). A D/A output unit presents a filtered version of the delayed speech to the user. The overall system delay \( B_1+B_2 \) is a constant.
Fig. 2.1 Speech Digitization System

a) Transmitter Section

b) Receiver Section
The analog filters were designed by GTE under a separate contract. They are described elsewhere. Therefore, the discussion here will begin after the continuous signal has been converted to \( s(k) \).

Figure 2.2 shows a more detailed block diagram of the transmitter system. The system consists of the following major components:

- SAMPLE buffer
- Adaptive Low-pass filter
- Pitch extraction loop
- Adaptive Residual Coder
- Noiseless source coder.

As noted, the SAMPLE buffer receives the incoming speech samples and holds them for further processing. For notational simplicity, the samples at the output of this buffer will be referred to as \( s(k) \). The adaptive low-pass filter is used as needed to help prevent an overflow of the SAMPLE buffer. The output of the adaptive low-pass filter \( s_f(k) \) forms the input to the pitch extraction loop. The pitch extraction loop uses a block of these filtered samples \( s_f(k) \) to estimate the pitch period \( T \) and the correlation coefficient \( \beta \). Using this information, the pitch-reduced speech samples \( v(k) \) are calculated by

\[
v(k) = s_f(k) - \beta s(k-T)
\]  

The pitch-reduced speech samples \( v(k) \) are then processed by the Adaptive Residual Coder. An estimate \( p(k) \) of \( v(k) \), produced by the adaptive predictor is subtracted from \( v(k) \) to form the prediction error \( e(k) \). The prediction error \( e(k) \) is then passed through an adaptive quantizer to yield the quantizer level \( q(k) \).
Fig. 2.2 PARC Transmitter
input to both the inverse quantizer (to update the rest of the system) and the noiseless source coder (to be transmitted down the channel). The noiseless source coder combines the quantizer level information \( q(k) \), the pitch period \( T \), and the correlation coefficient \( B \), and generates the binary bit stream \( b(m) \) to convey this information to the receiver.

The underlying design principle of the adaptation procedure is that all information used in updating the inverse quantizer and the predictor be available both at the transmitter and the receiver. This will allow the receiver to replicate these devices. Since the only information sent from the transmitter to the receiver is the quantizer output and pitching information, the adaptation procedures for the inverse quantizer and the predictor must use quantities derivable from them and a from pre-arranged initial state.

Although the PARC is basically a sequential system, the use of pitch redundancy reduction forces a block structure. For each block, new values of \( B \) and \( T \) are computed. The resulting block structure appears throughout the real-time implementation of the system.

Subsequent sections will describe each of the subsystems in the transmitter. The next section explains the operation of the transmitter buffer system. Since the noiseless source coder produces a variable number of bits for each sample, the rate at which samples are processed varies with time. Thus, the buffering operation is quite complex and important.

Section 2.3 describes the implementation of the adaptive low-pass filter. The filter is only activated when the SAMPLE buffer is almost full and it is only used to reduce the rate at which bits are generated. Thus, its operation is closely related to that of the SAMPLE buffer.
full and it is only used to reduce the rate at which bits are generated. Thus, its operation is closely related to that of the SAMPLE buffer.

The pitch extraction algorithm is a fairly standard AMDF system and is explained in Section 2.4. Although \( \Delta \) and \( T \) are computed for a fixed number of speech samples, the number of samples they are actually used on will vary.

The adaptive residual coder is a sophisticated ADPCM system. Section 2.5 describes parametric modifications used to optimize the structure for pitch-reduced speech and to combat channel errors. A new feature, known as pitched repetition, has been added to the ARC to reduce the bit rate during voiced speech. It is described in Section 2.6.

A central feature of the PARC system is the noiseless source coding structure. It allows the most efficient use of all channel bits to yield the highest fidelity speech. In the final implementation, it includes the binary representation of \( q(k) \), \( \Delta \) and \( T \) as well as synchronization and error control. Section 2.7 gives the details of this component.

The next-to-last section of this chapter describes the structure of the receiver. It too has a complex buffer structure which must be considered during the system design. The final section is a list of references to the various techniques employed by PARC.
2.2 Transmitter Buffer System

The buffering of data flows in the transmitter is a fairly complex procedure. Due to the variable rate coding procedure, the overall delay experienced by a speech sample traveling through the transmitter varies with time. During unvoiced speech or silence, the delay will be short but during voiced speech, it can be over a thousand sample times. However, the receiver must produce one reconstructed speech sample for each sample that enters the transmitter. Thus, if there are no channel errors, the overall system delay is fixed.

There are actually four separate buffers used in the transmitter buffer system. Three are used to facilitate data transfer and to allow parallel processing; one accommodates the variable delay. The buffers are the ADAM buffer, the SAMPLE buffer, the LEVEL buffer and the BIT buffer. They are shown schematically in Fig. 2.3. The ADAM buffer, the LEVEL buffer, and the BIT buffer are each double buffers which allow parallel data processing and have essentially a fixed delay. The SAMPLE buffer, though, is a large circular buffer which accommodates the system's variable-rate encoding.

The operation of the four buffers is best described by first explaining the input and output for each of them. Continuous speech signal enters the PARC system through an analog speech interface. This interface conditions the signal for the Analog Data Acquisition Module (ADAM) by low-pass filtering it and by adjusting the signal level to the range of the analog-to-digital converter in the ADAM. The ADAM samples this signal at a rate of 6.4 Kss and places the samples into the ADAM buffer. Thus, the input to the buffer is a sequence of single speech samples at the rate of one each 156.25 μsec.
Fig. 2.3 Transmitter Buffer System
Data is removed from the ADAM buffer in blocks of 126 samples each 19,687.5 μsec. Thus, data enters and leaves the buffer at an average rate of 6400 samples per second. The blocks of 126 samples are transferred to the SAMPLE buffer. The rate at which these samples are processed by the noise reducer and then are removed from the SAMPLE buffer depends on the number of bits they generate. The block-time of 19,687.5 μsec. was selected to correspond to both 126 input sampling intervals and to the time allowed to transmit 189 bits on the channel. Thus, the number of samples $N_B$ removed from the SAMPLE buffer in one block-time will be the number which generates at least 189 channel bits. This can range from as many as 500 or more during silence to fewer than 80 during voiced speech. The SAMPLE buffer must be large enough to accommodate this kind of variation without introducing undue delay.

The samples removed from the SAMPLE buffer in a given block time are processed by the PARC algorithm and the corresponding quantizer levels $q(k)$ are generated. The $N_B$ samples will generate $N_B$ quantizer levels. These are placed in the LEVEL buffer. Since PARC operates on a sample-by-sample basis, the $q(k)$ are loaded into their buffer one at a time.

The source coder generates full blocks of 189 bits. Therefore, it removes $N_B$ quantizer levels from the LEVEL buffer at one time. The resulting block of bits are stored in the bit buffer. They are clocked out of this buffer at the rate of 9600 bits per second.

The ADAM buffer, as all double buffers, actually has two half-buffers, each of which holds 126 speech samples. While the ADAM is filling one half-buffer with the incoming samples, the other half-buffer can be emptied into the SAMPLE buffer. The SAMPLE buffer is a circular buffer which holds up to 1024 speech samples. There are two pointers associated with the buffer which keep track of where samples are to enter the buffer.
and from where they are to leave the buffer. The distance between the
pointers indicates how many samples are currently in the sample buffer.

Before the samples are removed from the SAMPLE buffer, three operations
are performed. If the SAMPLE buffer is over a pitched repetition threshold,
it will signal the transmitter to do the pitched repetition as described
in Section 2.6. If the adaptive low-pass filter is called for, the oldest
80 samples in the buffer are filtered. The original samples are replaced
with the filtered samples. The $\beta$ and $T$ values for de-pitching are then
calculated based on the 80 oldest samples in the buffer. Samples are then
passed to the PARC for processing and the corresponding quantizer levels
are put into the LEVEL buffer. For each sample, the number of information
bits required to represent the quantizer level is computed. The process
will stop at either of the following three conditions:
1. The transmitter has generated the required 157 information bits.
2. The transmitter has no more input samples to process, i.e., the SAMPLE
   buffer is in an underflow condition. In this case, the transmitter will
   output a NULL code to the source encoder.
3. The transmitter has processed the maximum number of samples allowed
to be in real-time.

The LEVEL buffer is a large double buffer capable of holding up to
1200 quantizer levels. Even though the buffer can hold 1200 levels, it
will never have more than two blocks each of which generates 157 informa-
tion bits. One half-buffer is used for incoming quantizer levels, while
the other half-buffer is available to the noiseless source coder. The
noiseless source coder takes these quantizer levels, and, with the associ-
ciated quantized $\beta$ and $T$, generates a 189-bit block which is placed in the
BIT buffer.
The BIT buffer is a double buffer which holds 378 bits. One half-buffer receives bits from the channel coder, while the other half-buffer is available for output to the channel. Output takes place through the Input/Output Scroll (IOS-2). The IOS-2 provides the bits to the modem interface contained in the speech interface unit.
2.3 Adaptive Low-Pass Filter

Adaptive low-pass filtering is used in the system to provide a soft-failure capability under certain circumstances. When the SAMPLE buffer fills more rapidly than it is being emptied, for example during voiced speech, it is possible that it could overflow. It was found that low-pass filtering the speech mitigated this problem. When the SAMPLE buffer is nearly full, therefore, the speech is low-pass filtered to help guard against buffer overflow.

The recommended low-pass filter has been derived from a first order Butterworth filter, using the bilinear transformation to obtain a digital filter. The general form of the transfer function of this type of filter is

\[ H(z) = \frac{\omega_{CA} z^{-1}}{(\omega_{CA} + 1) + (\omega_{CA} - 1) z^{-1}} \]  

(2.2)

where

\[ \omega_{CA} = \tan \frac{\pi f_c}{f_s} \]

\[ f_c = \text{cutoff frequency in Hz.} \]

\[ f_s = \text{sampling frequency in Hz.} \]

The selected parameters are \( f_c = 1800 \text{Hz} \) and \( f_s = 6400 \text{ Hz} \). Transforming back into the sample domain, the filter equation can be written as

\[ s_f(k) = a_s(k) + a_s(k-1) + b_s f(k-1) \]  

(2.3)
The adaptive low-pass filtering operates by checking the number of samples in the SAMPLE buffer just prior to the pitch extraction calculations. If there are fewer than 501 samples in the SAMPLE buffer at that time, the low-pass filtering is skipped. If there are 501 or more samples in the same buffer, the filtering is implemented. Thus, filtering is only used when buffer overflow is threatened.

When filtering is called for, a block of 80 samples are low-pass filtered and the original samples are replaced by the filtered samples. A block of $N_B$ samples are then processed by PARC. Note that $N_B$ may be less than 80 so that some filtered samples may remain in the buffer and may be filtered again during the next block time.

The filtering is performed in blocks of 80 for two reasons. First, filtering in large blocks reduces the number of transitions between filtered and unfiltered speech. Second, the larger block causes some samples to be multiple-filtered if the sample buffer continues to fill. The design of the filter causes the effect of this multiple-filtering to be similar to filtering with a lower cutoff frequency. For the recommended parameters, using the 1800 Hz cutoff frequency filter twice results in an overall filter with a cutoff frequency of about 1350 Hz. In this way, the low-pass filtering is automatically increased when needed.

where

\[
A = \frac{\frac{1}{CA}}{CA + 1}
\]

\[
B = \frac{\frac{1}{CA} - 1}{CA + 1}
\]
2.4 Pitch Extraction

It is well known that voiced speech is highly correlated from pitch period to pitch period. A long-term prediction of $s_f(k)$ given by

$$s(k|k-T) = \beta s_f(k-T)$$

(2.4)

can be a good approximation to $s_f(k)$ for proper choice of $\beta$ and $T$. The optimum scale factor $\beta$ depends on the correlation between $s_f(k)$ and $s_f(k-T)$, and the best $T$ is an estimate of the pitch period measured in samples. The goal in selecting $\beta$ and $T$ is to minimize the time average prediction error over a block of $K$ samples

$$E = \frac{1}{K} \sum_{j=1}^{K} [s_f(j) - \beta s_f(j-T)]^2$$

(2.5)

In most applications, $\beta$ and $T$ are computed and used over a given block of $K$ samples. This procedure is modified in PARC. New values of $\beta$ and $T$ are computed each block-time. Therefore, they are held constant for $N_B$ samples and, it will be recalled, $N_B$ varies from block to block. However, it is not possible to determine $N_B$ until after $\beta$ and $T$ have been chosen. Therefore, $\beta$ and $T$ are always calculated for a fixed block size and used for a variable number of samples. In the recommended implementation, the fixed computation block is $K = 80$.

The pitch period estimate $T$ is computed by forming the Average Magnitude Difference Function (AMDF) and picking the value of $T$ for which this function is minimized. The AMDF function is

$$A(T) = \sum_{j=1}^{K} |s_f(j) - s_f(j-T)|$$

(2.6)
In this way, the value of $T$ selected usually matches the value obtained by minimizing the error $E$ in Eq. (2.5) but with far fewer computations. Once $T$ has been found, the error $E$ is minimized by selecting $B$ according to

$$B = \frac{\sum_{j=1}^{K} s_f^2(j-T)}{\sum_{j=1}^{K} s_f(j-T)}$$

(2.7)

For some blocks of speech, such as those including transition regions from silence or unvoiced speech to voiced speech, the value of $B$ given by Eq. (2.7) can be large. The use of such large values, however, can actually decrease the overall performance. Therefore, $B$ was limited to the range of $[-2,2]$. The $B$'s were uniformly quantized over this range. It was found that system performance was relatively insensitive to quantization noise on $B$. Therefore, $B$ is quantized to 97 levels. This is represented in the transmitted block with seven bits. The 31 patterns of these seven bits which do not represent valid $B$ values are used for another purpose.

By extracting pitch from different sentences and different speakers, it was found that the pitch period $T$ varied between 24 and 70 samples. Hence, the searching range was chosen to be between 20 and 83 yielding 64 possible values of $T$, which requires an 8-bit codeword for $T$. 
2.5 Adaptive Residual Coder

The heart of the PARC system is an Adaptive Residual Coder (ARC). The version recommended here has been modified to optimize its operation as a part of PARC. The input to the ARC is the pitch-reduced speech.

\[ v(k) = s_f(k) - \hat{s}(k-T) \]  

(2.10)

where \( \hat{s}(k-T) \) is the reconstructed version of \( s_f(k-T) \) available at both the transmitter and receiver. The ARC consists of two principal subsystems: an adaptive predictor and an adaptive quantizer. These will be described in separate subsections.

2.5.1 Adaptive Predictor

The adaptive predictor produces a linear prediction \( p(k) \) given by

\[ p(k) = \sum_{i=1}^{N} a_i(k) \hat{v}(k-i) \]  

(2.11)

which is to be an estimate of \( v(k) \). The \( \hat{v}(k-i) \) are the receiver's estimate of \( v(k-i) \). It can be argued that the predictor order \( N \) should match the order of the system which generates the \( v(k) \). However, predictors of order larger than 4 yield unsatisfactory performance in the presence of channel errors. Therefore, \( N=4 \) is used.

If the \( a_i(k) \) accurately model the \( v(k) \), and if the \( \hat{v}(k-i) \) are close to the \( v(k-i) \), then \( p(k) \) will be a good approximation to \( v(k) \). The \( a_i(k) \) are adaptive, and after \( p(k) \) is formed, they are updated. They are adapted according to steepest descent of \( e^2(k) \). This is approximated in the system by the following updating algorithm:

\[ a_i(k+1) = \delta b_i + (1-\delta) \left[ a_i(k) + \frac{2 \hat{v}(k-i) \hat{e}(k)}{\langle \hat{v}(k) \rangle^2} \right] \]  

(2.12)
where $<|\hat{v}(k)|>$ is a biased exponential time average of $|\hat{v}(k)|$

$$<|\hat{v}(k)|> = \frac{1}{1-a} \sum_{j=0}^{\infty} a^{j} |\hat{v}(k-j)| + \text{RMSMIN}$$

Thus, the $a_i(k)$ updating algorithm has eight parameters: $\delta$, $g$, $\alpha$, RMSMIN and $b_i$ for $i=1,2,3$ and 4. Three of them, $\delta$, $g$ and $\alpha$, essentially determine how much memory there is in the updating process. In order to minimize the effect of channel errors, the memory time was reduced from what would be optimal in the error-free case. This did not significantly degrade performance. The recommended values of these parameters are

$$\delta = 0.01$$
$$g = 0.02$$
$$\alpha = 0.90$$

The parameters $b_i$ represent the quiescent values of the coefficients $a_i(k)$. The values used in the original ARC are also recommended here.

$$b_i = \begin{cases} 
0.7 & i=1 \\
0 & i=2, 3 \text{ or } 4
\end{cases}$$

The quantity RMSMIN is perhaps the most sensitive parameter in the algorithm. It determines the minimum value of $<|\hat{v}(k)|>$ which affects both the adaptive predictor and the adaptive quantizer. The lower RMSMIN, the more the system responds during low level signals. This reduces granular noise and increases the data rate. The higher data rate means that the sample buffer fills faster leading to more low-pass filtering and increased use of pitched repetition. The value selected for RMSMIN must be matched to the dynamic range of $s(k)$. When $s(k)$ is represented on the interval $(-2048, 2047)$, an RMSMIN of

$$\text{RMSMIN} = 50$$

produces a good tradeoff.
2.5.2 Adaptive Quantizer

The prediction error $e(k)$ is the input to the adaptive quantizer whose basic design is illustrated in Fig. 2.4. The input is normalized by an adaptive scaling factor $\sigma(k)$ and the result is compared to a set of thresholds $T_i$. The recommended thresholds are symmetric and are illustrated in Fig. 2.5 and listed in Table 2.1. The level in which the normalized input falls specifies the quantizer output $q(k)$. The inverse quantizer output $\hat{e}(k)$ is the quantized version of the quantizer input. It is the product of a scaling factor $f(q(k))$ and the state variable $\sigma(k)$. The recommended scale factors are tabulated in Table 2.2. The recommended thresholds were computed to be equidistant between the scaling factors.

The state variable $\sigma(k)$ is designed to be an approximation to the standard deviation of $e(k)$. Most of the time the scaled average of $|\hat{v}(k)|$ is an acceptable estimate. However, in voiced speech at the beginning of a pitch period, $e(k)$ is much larger than usual. Therefore, whenever one of the outermost quantizer level occurs, $\sigma(k)$ is significantly increased. If no further outer level occurs, $\sigma(k)$ decays back to the scaled average of $|\hat{v}(k)|$. Thus, $\sigma(k)$ is updated by

$$\sigma(k) = \max\{\text{SMIN} \cdot |\hat{v}(k)|, \phi[q(k)]\sigma(k-1)\} \quad (2.13)$$

The first term in the braces of Eq. (2.13) usually dominates. This means that the quantizer behavior is largely determined by $\text{SMIN} \cdot |\hat{v}(k)|$ and, hence, by the product of $\text{SMIN}$ and $\text{RMSMIN}$. It is recommended that the scale factor $\text{SMIN}$ be set to 0.3.

The second term in the braces only affects performance at the beginning of pitch periods. The quantizer expansion factors $\phi[q(k)]$ are given in Table 2.2.
Fig. 2.4 Adaptive Quantizer
Fig. 2.5 Quantizer Thresholds and Output Levels
Table 2.1 Quantizer Thresholds

<table>
<thead>
<tr>
<th>i</th>
<th>( T_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.90</td>
</tr>
<tr>
<td>2</td>
<td>3.03</td>
</tr>
<tr>
<td>3</td>
<td>5.38</td>
</tr>
<tr>
<td>4</td>
<td>7.75</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 2.2 Quantizer Scaling Factors and Expansion Factors

<table>
<thead>
<tr>
<th>q(k)</th>
<th>( f[q(k)] )</th>
<th>( \phi[q(k)] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.7</td>
</tr>
<tr>
<td>2,7</td>
<td>( \pm 1.80 )</td>
<td>0.8</td>
</tr>
<tr>
<td>3,8</td>
<td>( \pm 4.25 )</td>
<td>0.9</td>
</tr>
<tr>
<td>4,9</td>
<td>( \pm 6.50 )</td>
<td>1.1</td>
</tr>
<tr>
<td>5,10</td>
<td>( \pm 8.00 )</td>
<td>1.5</td>
</tr>
<tr>
<td>6,11</td>
<td>( \pm 12.00 )</td>
<td>2.2</td>
</tr>
</tbody>
</table>
2.6 Pitched Repetition

Even with adaptive low-pass filtering, sections of voiced speech can generate a large number of bits rapidly. This could cause the SAMPLE buffer to overflow. To avoid the overflow problem, the system uses a technique known as Pitched Repetition. When overflow is imminent, a block of samples is deleted at the transmitter and replaced at the receiver with previous reconstructed speech. The replacement samples are the reconstructed samples from the previous pitch period. In order to have an absolute control on the overflow condition, the repetition size has to be carefully set. During a voiced region, the bit rate may be as high as 3.4 bits per sample. Thus, the repetition size of 80 samples is enough to force more than 126 samples to be transmitted in a time slot, i.e., the total number of output samples is more than that of input samples. The decision to use pitched repetition is made at the beginning of each block and a special signal is used to alert the receiver.

If the SAMPLE buffer contains more than 850 samples at the beginning of a block-time, pitched repetition is employed. First, the pitch period $T$ is computed in the usual way. The output pointer in the SAMPLE buffer is then moved forward 80 samples. Thus, these 80 samples will not be processed by PARC. They are, however, involved in the $\delta$ calculation which takes place with the output pointer at its new location.

The receiver must still produce an output for those samples which are not processed by PARC. It does this by using previous outputs delayed by one pitch period. Thus, if the first sample skipped is $s_f(k)$, it is represented at the receiver by $\hat{s}(k-T)$. Similarly, $s_f(k+1)$ is represented by $\hat{s}(k+1-T)$ and so on. The transmitter must know this since it uses $\hat{s}(k)$ values in computing $v(k)$. Therefore, the $\hat{s}(k)$ buffer is filled
with prior \( \hat{s}(k-T) \) values as part of the pitched repetition. After this, PARC resumes normal operation.

This method of repeating short intervals of samples from the previous pitch period works with little distortion during voiced speech because samples one pitch period apart are highly correlated. In this way, the buffer overflow problem is overcome with a minimum of additional distortion.

The receiver must be informed that a period of pitched repetition is taking place. As detailed in the next section, this is accomplished through use of the unused values of \( \delta \).
2.7 Noiseless Source Coder

The function of the noiseless source coder is to combine all of the information from PARC and produce the corresponding bit stream for transmission. The output of the coder are blocks of 189 bits. Each block represents one set of values of \( \beta \) and T and \( N_B \) level variables. In addition, the block provides for synchronization and error control.

The format of a typical block is illustrated in Fig. 2.6. The block is divided into three 63-bit frames to facilitate error control. The last six bits of each frame, denoted E1, E2 and E3 in Fig. 2.6, are used for error correction. A single-error-correcting \((57,63)\) Hamming code is used. The first 57 bits in each frame are available for information. The \((57,63)\) codes can each correct any single error so up to three errors per block can be corrected.

The first bit in the block, shown as Field A, is for block synchronization. It is set to 0 in odd numbered blocks and 1 in even numbered blocks. The receiver detects this pattern and knows where the block begins.

The next field in the block is six bits long and contains information specifying on the pitch period T. The pitch period is constrained to be an integer between 20 and 83, so six bits can transmit the pitch period without quantization error.

Field C in a normal frame is seven bits long and contains information on the pitch correlation coefficient \( \beta \). It was found that \( \beta \) can be quantized fairly coarsely with negligible degradation, so that 97 possible values in the range of \([-2,2]\) are allowed. This means that 97 of the possible 128 values of Field C are used to represent \( \beta \). If one of the other 31 patterns appears, it indicates that this is not a normal block. Rather, it is one using pitched repetition. The pitched repetition blocks are discussed later.
Bit #

Frame A

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D1</th>
<th>E1</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>78</td>
<td>1415</td>
<td>57</td>
<td>58</td>
</tr>
</tbody>
</table>

Frame B

<table>
<thead>
<tr>
<th>D2</th>
<th>E2</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>120 121 126</td>
</tr>
</tbody>
</table>

Frame C

<table>
<thead>
<tr>
<th>D3</th>
<th>E3</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>183 184 189</td>
</tr>
</tbody>
</table>

Field | Content
---|---
A | Synchronization Bit
B | Pitch Period T
C | Pitch correlation coefficient B
D1,D2,D3 | Quantizer levels
E1,E2,E3 | Parity bits

Fig. 2.6 Normal Block Format
Following the $\beta$ field in a normal block are the 157 bits representing the quantizer levels. These are denoted as Fields $D_1$, $D_2$ and $D_3$. The source code used for the quantizer levels are described in Table 2.3.

Quantizer levels 2 through 11 are each represented by a variable length bit pattern. Quantizer level 1, however, occurs so often that there are two ways of representing it. Isolated occurrences are represented by the 1-bit sequence 0. If level 1 occurs 14 times in a row, the entire string is represented by the sequence 1110. Thus, the source code is an overfull variable-length to variable-length mapping.

There is also a bit pattern associated with the null quantizer level sequence; this is used to fill out a block when the samples in the sample buffer do not generate at least 157 bits. Because of the variable number of bits used for different quantizer level sequences, an integral number of samples will not always generate exactly 157 bits. If there are more than 157 bits generated by a set of samples, the excess bits are the first bits transmitted in the next block's quantizer level field.

The normal format described above is used under most circumstances. If pitched repetition block is to be signaled to the receiver, however, the block format is changed slightly in the first frame of 63 bits, as shown in Fig. 2.7. Pitched repetition is signaled by using a special bit pattern, a "false $\beta$", in the field $C$ which is usually reserved for $\beta$. The next 7 bits are then taken from the quantizer level field $D_1$ to create Field $C'$ which transmits the actual $\beta$. Thus, in such a block, only 150 bits are used for quantizer levels.

To protect against missing a "false $\beta$", or deciding one in error, the bit patterns for the $\beta$ were carefully selected. They were designed so that neither situation can occur due to a single bit error. The "false $\beta$"
Table 2.3 Description of Quantizer Level Source Code

<table>
<thead>
<tr>
<th>Quantizer Level Sequence</th>
<th>Bit Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
</tr>
<tr>
<td>8</td>
<td>1100</td>
</tr>
<tr>
<td>3</td>
<td>1101</td>
</tr>
<tr>
<td>14 1's</td>
<td>1110</td>
</tr>
<tr>
<td>9</td>
<td>111100</td>
</tr>
<tr>
<td>4</td>
<td>111101</td>
</tr>
<tr>
<td>10</td>
<td>1111100</td>
</tr>
<tr>
<td>5</td>
<td>1111101</td>
</tr>
<tr>
<td>11</td>
<td>11111100</td>
</tr>
<tr>
<td>6</td>
<td>11111101</td>
</tr>
<tr>
<td>null</td>
<td>11111110</td>
</tr>
<tr>
<td>Bit #</td>
<td>A</td>
</tr>
<tr>
<td>-------</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Synchronization bit</td>
</tr>
<tr>
<td>B</td>
<td>Pitch period T</td>
</tr>
<tr>
<td>C</td>
<td>False B</td>
</tr>
<tr>
<td>C'</td>
<td>Pitch correlation coefficient B</td>
</tr>
<tr>
<td>D</td>
<td>Quantizer levels</td>
</tr>
<tr>
<td>E1</td>
<td>Parity bits</td>
</tr>
</tbody>
</table>

Fig. 2.7 First Frame During Pitched Repetition
is represented by 0000000. The 7 patterns with a single one and the 21 patterns with two ones are never transmitted. Thus, only 99 patterns are available for true \( \beta \) values. At the receiver, if field C contains either all zeros or a single 1, it is interpreted as the "false \( \beta \)". If no more than one channel error has occurred, this will happen if and only if a "false \( \beta \)" was actually transmitted.
2.8 Receiver

The principal elements of the receiver are illustrated in Fig. 2.8. The received bit sequence $b(m)$ is monitored by a synchronizer which establishes the beginning of a block. The decoder transforms the bits in a block into the quantizers levels $q(k)$ and the pitch parameters $\hat{f}$ and $T$. These are then processed by the inverse of the PARC algorithm. The $\hat{s}(k)$ values are stored in a variable delay receiver buffer. They are clocked out of the buffer and interpolated to produce the un-sampled output signal $\hat{s}_{\text{out}}$. The primes on all of these quantities introduced earlier to account for possible channel errors have been dropped for notational simplicity.

The synchronization actually operates in two modes: initial establishment and monitoring. During the establishment phase, the rest of the receiver system is disabled. The synchronizer looks for a sequence of bits spaced by 189 bit-times whose polarity oscillates. When a sequence of 10 bits with perfect oscillation is found, the synchronizer decides that it must represent the sync bit in 10 successive 189-bit blocks. The rest of the receiver is enabled and the synchronizer changes to the monitor mode. If it detects a significant number of errors in the sync bits, it assumes block synchronization has been lost. The receiver is disabled and the synchronizer returns to the synchronization mode.

Once synchronization has been established, the decoder can go to work. It basically inverts the operations performed by the noiseless source coder. Thus, it works on a full block of bits. The Hamming codes are decoded and any correctable bit errors are corrected. The values of $\hat{f}$ and $T$ are found and the sequence of quantizer level values is formed.
Fig. 2.8 Receiver Structure
If pitched repetition is being used, a special code is set. All of this information is placed in a double-buffer which interfaces with the PARC receiver.

The PARC receiver processes a full block of data. The S and T values are fixed for the block but the number of samples handled is the variable $N_B$. If pitched repetition is called for, the old $s(k)$ values are produced before the ARC receiver is enabled.

The output $s(k)$ from the PARC receiver are stored in the circular receiver buffer. This buffer complements the variable delay transmitter buffer and requires a carefully designed control algorithm. If there are no channel errors, the total delay for the system will be a constant. Therefore, the sum of the transmitter delay $B_1$ and the receiver delay $B_2$ will be fixed.

The buffer control logic in the decoder is designed to prevent the received sample buffer from ever overflowing or underflowing. In normal operation neither of those can happen but channel errors can add or delete samples. Since samples are removed from the buffer at a known rate, the buffer control logic will always know how many samples are in the buffer. If there is not enough room in the receiver buffer for all of the $s(k)$ in a block, the excess $q(k)$ are simply discarded. Since a full receive buffer corresponds to an empty transmitter buffer, this usually occurs during silence. The deletion of silence is generally not a problem.

As the transmitter buffer fills, the receiver buffer empties. If channel errors have caused the deletion of samples, it is possible for the receiver to run out of $s(k)$ values. This underflow condition is also prevented by the buffer control logic. If the number of $s(k)$ values in a block plus the number of $s(k)$ stored in the buffer do not total at least 126,
additional q(k) representing silence are added to the LEVEL buffer. Thus, there are always enough.

The upsampling has been added to reduce the sampling noise caused by the non-ideal nature of the analog output filter. The 6400 samp/sec sampling rate is at the Nyquist rate of the 3200 Hz output filter. Thus, severe aliasing is possible. The upsampler effectively increases the sampling rate to 12800 samp/sec. It does this by interpolating between successive s(k) values. It was found that linear interpolation was sufficiently accurate to greatly reduce the aliasing.
CHAPTER 3
SYNCHRONIZATION

Although PARC is basically a sequential algorithm, the use of pitch redundancy reduction and error control forces on it a block structure. The transmitter quantizes a block of $N_B$ speech samples using a set of pitched reduction parameters and encodes this into frames of binary information. At the other end, the receiver must identify these frames to be able to properly decode the information it receives. This necessitates some sort of synchronization between the transmitter and the receiver. In this chapter, the synchronization technique used in PARC and its implementation is described. Further, the operation is analyzed to illustrate its satisfactory performance.

There are two aspects to the synchronization operation performed in the receiver. The receiver must first locate the frame boundaries in the received bit stream. This is called synchronization acquisition. After acquisition, it must monitor the frame boundaries on a continuing basis to ensure that sync is not lost. This is called synchronization monitor. Contract requirements specify that bits are not dropped during transmission. Therefore, once synchronization is properly acquired, there should be no way of losing it. However, there are several reasons for providing the sync monitor. Sync acquisition is a probabilistic operation; and although there is a high probability of acquiring sync properly in one attempt, in case of a wrong decision, the sync monitor provides a way for re-attempting sync acquisition. There are other abnormal conditions that inevitably occur during algorithm development and testing which can also cause sync to be lost. Some of these are bad connections on the digital I/O connectors, faulty
IOS operation (see Chapter 12), reinitialization of the algorithm at one end of the communication system. All these make it imperative to provide the algorithm with the sync monitor, in other words, with re-syncing capabilities to ensure proper uninterrupted operation.
3.1 **Sync Algorithm Description**

The technique selected here to synchronize the receiver and the transmitter is similar to that used in the T1 Carrier System. A bit pattern called the synchronization pattern is selected. One bit at a time from the predetermined pattern is interjected at regular intervals into the bit stream generated at the transmitter. In this system, the sync bit is inserted at the beginning of each frame of 189 bits generated by processing a block of $N_B$ samples. The receiver tries to locate the sync pattern embedded in the received bit stream, thereby locating the frame boundaries.

Any bit pattern can be used for the sync pattern as long as it does not coincide with some naturally generated pattern at the transmitter. Using a shorter sync pattern reduces the memory space and computation required at the receiver during sync acquisition. After some consideration, a two bit pattern 01 was selected for the synchronization pattern.

The following subsections detail the algorithm for the two aspects of synchronization. A short analysis is also presented with each to get some idea of the performance of these operations.

3.1.1 **Synchronization Acquisition**

There are two considerations in selecting this algorithm. First, it should not take too long for each acquisition operation. And secondly, the probability of making the right decision should be reasonably high to ensure that the right synchronization is achieved in a couple of attempts, if not in one.

The sync acquisition algorithm consists of segmenting the received bit stream into blocks of 189 bits each. One of the 189 bits is the sync bit,
and the corresponding bit position follows the sync pattern over the blocks of received bits. The decoder generates the sync pattern at the receiver, and checks its correlation with each of the 189 bit positions. The position that correlates exactly with the sync pattern for 10 blocks is picked to be the sync bit, marking the beginning of subsequent frames.

For sync acquisition to be unambiguously successful, only the sync bit of the 189 bits in a frame must correlate exactly with the sync pattern for 10 blocks. If there were no transmission errors, the problem here would consist of picking a deterministic sequence from the midst of a stochastic process. However, the received sync pattern is corrupted by transmission errors, an average error rate of 1%. Each of the other 188 bits are random 1's and 0's, and they correlate with the sync pattern with a probability of 0.5. With these and the assumption that the channel affects the bits independently, the probability of making a correct decision about sync acquisition can be determined.

The probability that the sync bit is transmitted without errors for 10 consecutive blocks is

\[ \alpha = (0.99)^{10} = 0.9044 \]

The probability that one of the other bit positions correlates perfectly with the sync pattern for 10 consecutive blocks is

\[ \beta = (0.5)^{10} = 0.000976 \]

The probability that none of the other 188 bit positions correlates perfectly with the sync pattern is \((1-\beta)^{188}\). The probability of an unambiguous sync acquisition decision is

\[ \alpha(1-\beta)^{188} = 0.9044 \times 0.8322 = 0.7526 \]
Thus, the probability of successful sync acquisition in several attempts can be computed:

In one attempt, \( P\{\text{sync acquisition}\} = 0.7526, P\{\text{failure}\} = 0.2474 \)

In two attempts, \( P\{\text{sync acquisition}\} = 0.9388, P\{\text{failure}\} = 0.0612 \)

In three attempts, \( P\{\text{sync acquisition}\} = 0.9848, P\{\text{failure}\} = 0.0151 \)

The implementation of this algorithm is slightly different from the description here. This was done to reduce the amount of computation required.
3.1.2 Synchronization Monitor

There are three considerations in selecting the algorithm here. The sync monitor checks to see if the received sync bits follow the sync pattern. Because of transmission errors, there are some errors in the received sync bits inspite of the error control. After allowing for these errors, the algorithm should decide that sync is retained. The probability of erroneously deciding that sync is lost should be extremely small to ensure that the receiver operates uninterrupted for long periods of time. Secondly, if sync is lost, the algorithm should realize this in a reasonably short time. And finally, the algorithm should be computationally simple to implement.

With these considerations in mind, the following algorithm is suggested. A correlation between the received sync bit $S_i$ and the expected sync bit $\hat{S}_i$ is computed. Based on the correlation, a value is assigned to a r.v. $x_i$.

$$x_i = \begin{cases} +2 & \text{if } S_i \neq \hat{S}_i \\ -1 & \text{if } S_i = \hat{S}_i \end{cases}$$

This variable is used to update a sync variable $v_i$.

$$v_i = v_{i-1} + x_i$$

If at any time, the variable $v_i$ exceeds the threshold $T$, it is decided that sync has been lost. The variable $v_i$ starts with an initial value 0, and is constrained to be non-negative for all $i$. If its value drops below 0, it is reinitialized to 0. The threshold $T$ used here is 12.

This algorithm has the effect of switching the rate of change of the sync variable from +2 when $(S_i \neq \hat{S}_i)$ to -1 when $(S_i = \hat{S}_i)$. If the channel error rate $r$ is less than 1/3, the rate of change is negative. The sync variable decays to 0 and stays there. If the channel error rate is greater than 1/3,
the rate of change is positive and the sync variable drifts towards the
threshold T. The maximum positive slope is 2 and occurs when the channel
error rate is 1.

The rate of change $x_i$ is a binary random variable. Over a period of
time, its average is the slope $\Delta v$ of the sync variable. The slope $\Delta v$
can be used to determine the time before sync is lost.

$$t = T/\Delta v$$

The slope $\Delta v$ is a binomial random variable. Its function $t$, the time before
sync is lost, which is also a random variable can be described by a dis-
tribution similar to the binomial distribution.

$$P(t) = \frac{\Gamma(t+1)}{\Gamma(t/3+5) \Gamma(2t/3-3)} \frac{r(t/3+4)}{1-r} \frac{(1-r)(2t/3-4)}{t \geq 6}$$

Its expected value is

$$<t> = T/(3r-1)$$

Using these, some estimates of the performance of the sync monitor algorithm
can be obtained. While proper sync is retained, the error rate for the sync
bit is reduced to 0.005 by the error control. At this error rate, the
algorithm would never lose sync. If the error rate deviated from its average
value to 1, it would take 6 frames for sync to be lost. The probability of
this can be computed to be $1.5625 \times 10^{-14}$.

If sync is actually lost, the average error rate for the sync bit is
0.5. The algorithm would take an average of 20 frames before deciding it has
lost sync.

The implementations of these two algorithms are described in Chapter 12.
The sync acquisition algorithm is implemented slightly differently from its
description here to reduce the computation involved. The sync monitor
algorithm is implemented as described here using simple logical and
shift operations.
CHAPTER 4
PITCH EXTRACTION STUDIES

4.1 Introduction

The aim of efficient coding methods is to reduce the channel capacity required to transmit a signal with specified fidelity. To achieve this objective, it is desirable to reduce the redundancy of the transmitted signal. One well-known procedure for removing signal redundancy is predictive coding. In predictive coding, redundancy is removed by subtracting from the signal that part which can be predicted from its past. The PARC system is essentially APC system which includes a pitch extraction loop for long-term redundancy removal.

In this chapter, several studies concerning pitch redundancy removal in PARC are described. The correlation technique, as well as AMDF algorithm, for pitch extraction is outlined in Section 4.2. The complete algorithm with pitch extraction loop was simulated on a digital computer; simulation results are discussed in Section 4.3.
4.2 Pitch Extraction Algorithms

It is well known that voiced speech is highly correlated from pitch period to pitch period [1]. The long term prediction of $s(k)$ is given by

$$\hat{s}(k|k-T) = \beta s(k-T)$$

(4.1)

Here $\beta$ is a scalar which depends on the correlation between $s(k)$ and $s(k-T)$ while $T$ is an estimate of the pitch period (in samples). The use of $\tau$ reflects the amplitude changes of speech signal which occur from period to period especially during the beginning and end of the voiced segments. For unvoiced speech, $\beta$ is generally small and long-term prediction is relatively ineffective. The long-term prediction $\hat{s}(k|k-T)$ is subtracted from $s(k)$ to form the pitch-reduced-speech $v(k) = s(k) - \hat{s}(k|k-T)$.

The goal in selecting $\beta$ and $T$ is to minimize the error

$$E_1 = \frac{1}{K} \sum_{j=1}^{K} [s(j) - \beta s(j-T)]^2$$

(4.2)

Here block adaptation with block length of $K$ has been assumed. The choice of $K$ depends on various factors and will be discussed in a next section. The derivative of $E_1$ with respect to $\beta$ yields

$$\frac{3E_1}{3\beta} = \frac{2}{K} \sum_{j=1}^{K} [s(j) - \beta s(j-T)] s(j-T)$$

(4.3)

Equating this derivative to zero and solving for $\beta$ gives

$$\beta = \frac{\sum_{j=1}^{K} s(j)s(j-T)}{\sum_{j=1}^{K} s^2(j-T)}$$

(4.4)
If this result is substituted in Eq. (4.2), the equation becomes function of $T$ alone given by

$$E_1(T) = \frac{1}{K} \sum_{j=1}^{K} s^2(j) - \frac{1}{K} \sum_{j=1}^{K} s(j)s(j-T) \frac{1}{K} \sum_{j=1}^{K} s^2(j-T)$$

(4.5)

Therefore to minimize $E_1$ with respect to $T$ it is necessary to maximize the rightmost term of equation (4.5). The approach used was to compute

$$A(T) = \frac{\sum_{j=1}^{K} s(j)s(j-T)}{\sum_{j=1}^{K} s^2(j-T)}$$

(4.6)

for all values of $T$, $T_{\text{min}} \leq T \leq T_{\text{max}}$ and then select the value for which $A(T)$ is maximum. The lower limit, $T_{\text{min}}$, of the search range was selected to be smaller than minimum value of pitch periods for different speakers while the upper limit, $T_{\text{max}}$, is influenced by various factors such as number of bits available for transmission of $T$'s, processing time limitations due to real time application and maximizing energy reduction.

The above method, though simple to implement, involves extensive computation. For example, if the block length is $K$ and searching range is $R$ then for each value of $R$ there are $2K+2$ multiplications and $K$ additions. Hence the total number of multiplications for finding the pitch period becomes $R(2K+2)$; if $R=100$ and $K=100$ then this number is 20200 (this is just for one block of 100 samples. This many multiplications consume significant processing time which is crucial in real time implementation in Macro Arithmetic Processor (MAP).
The stringent requirement on timing in the MAP, led to a modification of the above correlation technique for pitch extraction. This is done by forming Average Magnitude Difference Function (AMDF)[2]

\[ A'(T) = \sum_{j=1}^{K} |s(j) - s(j-T)| \quad (4.7) \]

\[ T_{\text{min}} \leq T \leq T_{\text{max}} \]

It is easy to see that for any periodic function, the above sum is minimum for \( T \) equal to the period. Hence, the pitch parameter \( T \) was determined by minimizing the function \( A'(T) \) with respect to \( T \). The gain parameter was obtained by substituting this value of \( T \) into Eq. (4.4). The computational saving in this method is apparent since there is no multiplication involved.

This modification of correlation method gives exactly the same values of \( T \) (and hence of \( \beta \)) in voiced speech but differs in unvoiced speech. However, unvoiced speech is non-periodic and \( T \) is arbitrary and hence not important.

Figure 4.1 shows the plot of \( \beta \) and \( T \) against speech samples. The correlation coefficient \( \beta \) jumps to a high value for a voiced speech block which is followed by silence. This high value amplifies the quantization noise thus decreasing overall signal to noise ratio. Limiting \( \beta \) to \([-2, 2]\) was found to eliminate this problem and give satisfactory performance. Further discussion of limiting in context with quantization of \( \beta \) and coding is presented in Chapter 11.
Fig. 4.1 Pitch Period T and Correlation Coefficient B vs Sample Number (Set I: Male Speaker)
PARC algorithm was simulated on PDP 11/60 computer. The following phonetically balanced sentences were chosen for the simulation work.

Male speaker: sent 1 - "Cats and Dogs each hate the other."
Female speaker: sent 11 - "The pipe began to rust while new."

Beta's and T's were extracted using both the correlation and AMDF methods discussed above. The results of this study are shown in Table 4.1.
### TABLE 4.1a

Comparison of Correlation and AMDF Method for Pitch Extraction

**Sentence 11: Female Speaker**

<table>
<thead>
<tr>
<th>Sample No.</th>
<th>Correlation Technique</th>
<th>AMDF Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( B )</td>
<td>( T )</td>
</tr>
<tr>
<td>3500</td>
<td>-0.26</td>
<td>23</td>
</tr>
<tr>
<td>3600</td>
<td>-0.23</td>
<td>25</td>
</tr>
<tr>
<td>3700</td>
<td>0.42</td>
<td>22</td>
</tr>
<tr>
<td>3800</td>
<td>0.85</td>
<td>58</td>
</tr>
<tr>
<td>3900</td>
<td>1.03</td>
<td>29</td>
</tr>
<tr>
<td>4000</td>
<td>0.95</td>
<td>29</td>
</tr>
<tr>
<td>4100</td>
<td>0.91</td>
<td>88</td>
</tr>
<tr>
<td>4200</td>
<td>0.93</td>
<td>88</td>
</tr>
<tr>
<td>4300</td>
<td>0.99</td>
<td>29</td>
</tr>
<tr>
<td>4400</td>
<td>0.99</td>
<td>29</td>
</tr>
<tr>
<td>4500</td>
<td>0.80</td>
<td>29</td>
</tr>
<tr>
<td>4600</td>
<td>1.02</td>
<td>29</td>
</tr>
<tr>
<td>4700</td>
<td>1.00</td>
<td>29</td>
</tr>
<tr>
<td>4800</td>
<td>0.92</td>
<td>29</td>
</tr>
<tr>
<td>4900</td>
<td>0.61</td>
<td>28</td>
</tr>
<tr>
<td>5000</td>
<td>0.31</td>
<td>25</td>
</tr>
<tr>
<td>5100</td>
<td>0.32</td>
<td>20</td>
</tr>
<tr>
<td>5200</td>
<td>0.56</td>
<td>94</td>
</tr>
<tr>
<td>5300</td>
<td>0.30</td>
<td>62</td>
</tr>
<tr>
<td>5400</td>
<td>0.93</td>
<td>27</td>
</tr>
<tr>
<td>5500</td>
<td>0.87</td>
<td>29</td>
</tr>
<tr>
<td>5600</td>
<td>0.89</td>
<td>30</td>
</tr>
<tr>
<td>5700</td>
<td>0.99</td>
<td>30</td>
</tr>
<tr>
<td>5800</td>
<td>1.01</td>
<td>30</td>
</tr>
<tr>
<td>5900</td>
<td>1.01</td>
<td>30</td>
</tr>
<tr>
<td>6000</td>
<td>1.07</td>
<td>30</td>
</tr>
<tr>
<td>6100</td>
<td>1.07</td>
<td>30</td>
</tr>
<tr>
<td>6200</td>
<td>1.08</td>
<td>30</td>
</tr>
<tr>
<td>6300</td>
<td>1.04</td>
<td>30</td>
</tr>
<tr>
<td>6400</td>
<td>1.01</td>
<td>30</td>
</tr>
<tr>
<td>6500</td>
<td>0.97</td>
<td>30</td>
</tr>
<tr>
<td>6600</td>
<td>0.94</td>
<td>30</td>
</tr>
<tr>
<td>6700</td>
<td>0.86</td>
<td>30</td>
</tr>
<tr>
<td>6800</td>
<td>0.79</td>
<td>30</td>
</tr>
<tr>
<td>6900</td>
<td>0.66</td>
<td>31</td>
</tr>
<tr>
<td>7000</td>
<td>0.19</td>
<td>63</td>
</tr>
</tbody>
</table>
TABLE 4.1b
Comparison of Correlation and AMDF Method for Pitch Extraction

Sentence 1: Male Speaker

<table>
<thead>
<tr>
<th>Sample No.</th>
<th>Correlation Technique</th>
<th>Sample No.</th>
<th>AMDF Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\beta$</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>1700</td>
<td>0.53</td>
<td>141</td>
<td>1700</td>
</tr>
<tr>
<td>1800</td>
<td>0.35</td>
<td>141</td>
<td>1800</td>
</tr>
<tr>
<td>1900</td>
<td>0.48</td>
<td>135</td>
<td>1800</td>
</tr>
<tr>
<td>2000</td>
<td>0.11</td>
<td>148</td>
<td>2000</td>
</tr>
<tr>
<td>2100</td>
<td>0.30</td>
<td>27</td>
<td>2100</td>
</tr>
<tr>
<td>2200</td>
<td>1.45</td>
<td>42</td>
<td>2200</td>
</tr>
<tr>
<td>2300</td>
<td>0.71</td>
<td>48</td>
<td>2300</td>
</tr>
<tr>
<td>2400</td>
<td>0.76</td>
<td>50</td>
<td>2400</td>
</tr>
<tr>
<td>2500</td>
<td>0.93</td>
<td>52</td>
<td>2500</td>
</tr>
<tr>
<td>2600</td>
<td>0.74</td>
<td>53</td>
<td>2600</td>
</tr>
<tr>
<td>2700</td>
<td>0.96</td>
<td>53</td>
<td>2700</td>
</tr>
<tr>
<td>2800</td>
<td>0.94</td>
<td>53</td>
<td>2800</td>
</tr>
<tr>
<td>2900</td>
<td>0.94</td>
<td>54</td>
<td>2900</td>
</tr>
<tr>
<td>3000</td>
<td>0.94</td>
<td>55</td>
<td>3000</td>
</tr>
<tr>
<td>3100</td>
<td>0.90</td>
<td>57</td>
<td>3100</td>
</tr>
<tr>
<td>3200</td>
<td>0.72</td>
<td>59</td>
<td>3200</td>
</tr>
<tr>
<td>3300</td>
<td>0.60</td>
<td>61</td>
<td>3300</td>
</tr>
<tr>
<td>3400</td>
<td>0.54</td>
<td>62</td>
<td>3400</td>
</tr>
<tr>
<td>3500</td>
<td>0.79</td>
<td>24</td>
<td>3500</td>
</tr>
<tr>
<td>3600</td>
<td>1.16</td>
<td>52</td>
<td>3600</td>
</tr>
<tr>
<td>3700</td>
<td>1.21</td>
<td>53</td>
<td>3700</td>
</tr>
<tr>
<td>3800</td>
<td>1.01</td>
<td>53</td>
<td>3800</td>
</tr>
<tr>
<td>3900</td>
<td>0.89</td>
<td>53</td>
<td>3900</td>
</tr>
<tr>
<td>4000</td>
<td>0.81</td>
<td>55</td>
<td>4000</td>
</tr>
<tr>
<td>4100</td>
<td>0.97</td>
<td>57</td>
<td>4100</td>
</tr>
<tr>
<td>4200</td>
<td>0.98</td>
<td>58</td>
<td>4200</td>
</tr>
<tr>
<td>4300</td>
<td>1.00</td>
<td>58</td>
<td>4300</td>
</tr>
<tr>
<td>4400</td>
<td>0.96</td>
<td>59</td>
<td>4400</td>
</tr>
<tr>
<td>4500</td>
<td>0.85</td>
<td>59</td>
<td>4500</td>
</tr>
<tr>
<td>4600</td>
<td>0.92</td>
<td>60</td>
<td>4600</td>
</tr>
<tr>
<td>4700</td>
<td>0.84</td>
<td>61</td>
<td>4700</td>
</tr>
<tr>
<td>4800</td>
<td>0.74</td>
<td>62</td>
<td>4800</td>
</tr>
<tr>
<td>4900</td>
<td>0.69</td>
<td>65</td>
<td>4900</td>
</tr>
<tr>
<td>5000</td>
<td>0.60</td>
<td>66</td>
<td>5000</td>
</tr>
<tr>
<td>5100</td>
<td>0.60</td>
<td>68</td>
<td>5100</td>
</tr>
<tr>
<td>5200</td>
<td>0.42</td>
<td>65</td>
<td>5200</td>
</tr>
<tr>
<td>5300</td>
<td>0.93</td>
<td>64</td>
<td>5300</td>
</tr>
<tr>
<td>5400</td>
<td>0.71</td>
<td>64</td>
<td>5400</td>
</tr>
<tr>
<td>5500</td>
<td>0.87</td>
<td>65</td>
<td>5500</td>
</tr>
<tr>
<td>5600</td>
<td>0.66</td>
<td>131</td>
<td>5600</td>
</tr>
<tr>
<td>5700</td>
<td>0.33</td>
<td>66</td>
<td>5700</td>
</tr>
<tr>
<td>5800</td>
<td>0.32</td>
<td>62</td>
<td>5800</td>
</tr>
</tbody>
</table>
4.3 Redundancy Removal

As described earlier, the APC system is based on the removal of two kinds of redundancy: short term redundancy caused by vocal tract filter and long-term redundancy caused by pitch frequency. Once the pitch period T and gain parameter $\beta$ are determined, reduced speech is formed as

$$v(k) = s(k) - \beta s(k \mid k-T) \tag{4.8}$$

where $s(k \mid k-T)$ represents the reconstructed speech sample. Figure 4.2 shows the plot of reduced and original speech. It is easy to notice the energy reduction achieved in the almost periodic voiced portion of the speech. The amount of energy reduction achieved is expressed by SER (Signal Energy Reduction) which is calculated as

$$\text{SER} = -10 \log_{10} \frac{\sum v^2(k)}{\sum s^2(k)} \tag{4.9}$$

As the value of signal energy reduction is increased the dynamic range of the input signal to quantizer is reduced; hence the reduced speech signal can be represented by the lower quantizer levels thus requiring fewer bits for transmission. The SER can be increased by accurately picking pitch period T and choosing the block size such that effect of transition of $\beta$ from block to block is minimum. The parameters $\beta$ and T are associated with every block. For smaller block sizes, the number of parameters to be transmitted per second is increased. However, for smaller block size, the amplitude variations are closely represented by $\beta$ and transition of $\beta$ from block to block is smooth. These factors contribute to improve SER.

Table 4.2 shows the effect of block-size variation on SER. The following performance measures were also computed:
Original Speech

Reduced Speech

Sample Number

Fig. 4.2b Sentence II, Female Speaker
### Table 4.2

Effect of Block Size on Various Performance Measures

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Block Size</th>
<th>SER (db)</th>
<th>Overall SNR (db)</th>
<th>Inloop SNR (db)</th>
<th>Entropy bits/sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>8.47</td>
<td>20.81</td>
<td>12.34</td>
<td>1.53</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>7.07</td>
<td>20.24</td>
<td>13.17</td>
<td>1.48</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>5.63</td>
<td>19.08</td>
<td>13.45</td>
<td>1.46</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>5.44</td>
<td>18.97</td>
<td>13.53</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>5.15</td>
<td>18.85</td>
<td>13.70</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>140</td>
<td>4.70</td>
<td>18.26</td>
<td>13.56</td>
<td>1.44</td>
</tr>
<tr>
<td></td>
<td>160</td>
<td>4.56</td>
<td>18.33</td>
<td>13.77</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td>180</td>
<td>4.41</td>
<td>18.31</td>
<td>13.90</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>4.07</td>
<td>18.20</td>
<td>14.13</td>
<td>1.43</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
<td>11.19</td>
<td>22.78</td>
<td>11.59</td>
<td>1.48</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>9.81</td>
<td>22.44</td>
<td>12.63</td>
<td>1.44</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>8.48</td>
<td>21.72</td>
<td>13.24</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>8.04</td>
<td>21.65</td>
<td>13.61</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td>140</td>
<td>8.20</td>
<td>21.57</td>
<td>13.37</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td>160</td>
<td>7.92</td>
<td>21.59</td>
<td>13.67</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>180</td>
<td>8.16</td>
<td>21.64</td>
<td>13.48</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>8.33</td>
<td>21.58</td>
<td>13.25</td>
<td>1.44</td>
</tr>
</tbody>
</table>
Overall SNR = \(10 \log_{10} \frac{\sum s^2(k)}{\sum [s(k) - \hat{s}(k)]^2}\)  

Inloop SNR = \(10 \log_{10} \frac{\sum v^2(k)}{\sum [v(k) - \hat{v}(k)]^2}\)  

where \(v(k) = \hat{v}(k) + q(k)\)

Entropy \(H = - \sum \frac{1}{l} p_i \log_2 p_i\)

where \(p_i\) = Probability of occurrence of \(i^{th}\) quantizer level.

As SER improves overall performance also improves. It is interesting to note that the SNR (overall) may increase even if the SNR (inloop) decreases because of the improvement in SER. The speech signal spectrum becomes flatter because of pitch extraction thus adversely affecting the performance of predictor in INLOOP. However, because of smaller dynamic range of reduced speech, the quantizer noise is decreased which more than compensates for the poor performance of predictor. Hence the overall performance improves.

The searching range \((T_{\text{max}} - T_{\text{min}})\) for pitch extraction also affects the redundancy removal. It was observed that a longer searching range gives better SER while a small value of searching range decreases SER by as much as 2 db. A searching range of the order of twice the maximum pitch period appears to be sufficient. However, the longer searching range also means more computations and hence more CPU time. In the real-time simulations on the MAP, timing is critical and therefore the number of computations and memory transfers need to be reduced. In such cases the searching range must be reduced to achieve a compromise between the number of computations.
and the reduction in SER that can be tolerated.

It was noticed that the pitch extraction algorithm sometimes picks double or triple pitch periods. This fact has only a modest effect redundancy removal. However, the transmission of double or triple pitch period values may require the allocation of more bits for transmission of $T$. Again, it is desirable to limit the search range.

In Section 4.2, the estimation algorithm used to compute the pitch period $T$ and long term gain is based solely on the original speech. In fact, as seen by examining Fig. 2.10 the long-term redundancy removal operation actually subtracts the reconstructed speech from original speech. In an attempt to compensate for this fact, the $\beta$ obtained from Eq. (4.4) was modified by multiplying by scalar $\alpha$ as

$$\beta^* = \alpha \beta$$

(4.13)

Here $\alpha$ can be expressed [3] as

$$\alpha = \frac{1}{1 + \alpha}$$

(4.14)

where $\alpha$ is the inverse of signal to noise ratio. The parameter $\alpha$ was varied between 0 and 1.2 with no significant improvement was noticed. See Table 4.3.
TABLE 4.3

The Effect of $a$ on SER and Overall SNR

<table>
<thead>
<tr>
<th>$a$</th>
<th>SER</th>
<th>Overall SNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 db</td>
<td>15.61 db</td>
</tr>
<tr>
<td>0.9</td>
<td>5.45 db</td>
<td>19.23 db</td>
</tr>
<tr>
<td>0.94</td>
<td>5.55 db</td>
<td>19.29 db</td>
</tr>
<tr>
<td>0.96</td>
<td>5.57 db</td>
<td>19.24 db</td>
</tr>
<tr>
<td>0.98</td>
<td>5.55 db</td>
<td>19.40 db</td>
</tr>
<tr>
<td>0.995</td>
<td>5.52 db</td>
<td>19.28 db</td>
</tr>
<tr>
<td>1.0</td>
<td>5.55 db</td>
<td>19.31 db</td>
</tr>
<tr>
<td>1.04</td>
<td>5.48 db</td>
<td>19.16 db</td>
</tr>
<tr>
<td>1.1</td>
<td>5.37 db</td>
<td>19.21 db</td>
</tr>
<tr>
<td>1.2</td>
<td>5.07 db</td>
<td>18.79 db</td>
</tr>
</tbody>
</table>
4.4 References


CHAPTER 5
TREE CODING

5.1 Introduction

The concept known as tree coding was investigated as part of this study to evaluate its ability to improve performance. One particular algorithm, the (M,L) algorithm, was the basis for most of the investigation. A modified version of the (M,L) algorithm called adaptive tree searching was developed and investigated. Simulations indicated that adaptive tree searching marginally improved the performance of the PARC algorithm.
5.2 The (M,L) Algorithm

5.2.1 Description

The (M,L) algorithm is one of a number of algorithms which perform what is known as tree coding. The main idea of tree coding is to defer making a decision, in this case, which quantizer level should be used for a given sample, until a later time when it can be made in light of that which follows. Tree coding is useful in predictive or backward adaptive quantization systems, because the selection of a quantizer level affects the selection of quantizer levels in the future. This effect can be represented graphically in the form of a tree, where a node represents the "state" of the system as a result of selecting the sequence of quantizer levels leading to that "state", with a branch connecting the node to the node representing the previous "state". The tree is rooted by an arbitrary "state" at an arbitrary time, and evolves in time, with a new level of nodes added at each sample time.

The (M,L) algorithm operates in the following way, and is illustrated in Figure 5.1. At a given time, let us say that there are n nodes in the outermost level of the tree, and a new sample is to be quantized. A new level of nodes are then "grown" from the outermost level, representing all of the new possible "states". Thus, if there are k quantizer levels, there will be kn nodes in the new level. These new nodes are then ranked by some performance criterion, such as quantization noise. Nodes are next pruned from the tree. This is a key step, because it is this that prevents the task from growing exponentially. In order to insure that a quantizer level is selected for a sample in a finite amount of time, a
quantizer level decision is forced, if necessary, for the sample \( L \) time units ago, by picking the predecessor node, in the level \( L \)-levels in, of the best of the new nodes. All branches which do not stem from that predecessor are pruned. If more than \( M \) new nodes remain, the \( M \) best nodes are kept, and the rest are pruned. The process then starts over.
5.2.2 Results

The (M,L) algorithm has been studied by a number of investigators in connection with simpler quantization schemes, such as adaptive delta modulation (ADM) and adaptive differential pulse code modulation (ADPCM). For example, Jayant and Christensen reported a 3dB improvement in the signal-to-noise ratio (SNR) using the (M,L) algorithm with simple ADM and ADPCM schemes, with M=4 and L=7. In contrast, our investigation concentrated on the use of the (M,L) algorithm in connection with adaptive residual coding (ARC). It was originally envisioned that the resulting algorithm would be embedded within a pitch extraction loop. Some preliminary work was done, however, without the pitch extraction loop in order to verify the tree coding software and to gain some insight into its operation. For example, by using a four level fixed quantizer and a third order fixed predictor in the ARC algorithm, an improvement of 3dB in the SNR was achieved for M=4 and L=7. Unfortunately, the results obtained using the standard ARC algorithm did not show as great an improvement in SNR.

In order to make the following results more understandable though, a comment is necessary here. One of the key elements of the ARC algorithm is the source coding, which translates quantizer levels into bit patterns. The selection of a good source code is dependent, however, on the statistics of that which is to be encoded. As a result, the performance of the (M,L) algorithm was evaluated without use of a source coder. Instead, the entropy of the quantizer levels was computed, which allows for a fair comparison between the possibilities. In general, a higher entropy allows better performance.

Some of the first results obtained were for the first second of Sentence
"Cats and dogs each hate each other". In this series of runs, a five level quantizer was used in the standard ARC algorithm, M was set to five, and L was varied. The results are shown in Table 5.1. These results illustrate the typical effect of tree coding: increasing L improves performance by increasing the SNR and decreasing the entropy simultaneously. The results also illustrate another phenomenon of tree coding: that tree coding becomes less effective in improving performance as the performance of the base algorithm improves.

More results were obtained for the first second of Sentence 1 using a 19 level quantizer. The results are shown in Table 5.2. These results again show that, in general, increasing L or M (or both) improves performance, but that the amount of improvement is smaller than that achievable by the poorer-performing five level quantizer. There are, however, several instances where it appears that increasing L or M has decreased performance. This can occur because tree coding can make a suboptimal decision because the set of possible decisions is deliberately limited. As a result, tree coding can occasionally get "fooled".

Simulations were also performed with the tree coding ARC algorithm embedded within a pitch extraction loop. Some typical results are shown in Table 5.3. These results were obtained from Sentence 1, using an 11 level quantizer and M=5. Performance is again generally improved by tree coding, but by only a small amount.
Table 5.1. Results for a five level quantizer, $M=5$.

<table>
<thead>
<tr>
<th>L</th>
<th>SNR (dB)</th>
<th>ENTROPY (BITS/SAMPLE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.03</td>
<td>1.423</td>
</tr>
<tr>
<td>2</td>
<td>14.60</td>
<td>1.421</td>
</tr>
<tr>
<td>4</td>
<td>14.66</td>
<td>1.407</td>
</tr>
<tr>
<td>8</td>
<td>14.98</td>
<td>1.403</td>
</tr>
<tr>
<td>16</td>
<td>15.08</td>
<td>1.393</td>
</tr>
</tbody>
</table>

Table 5.2. Results for a 19 level quantizer.

<table>
<thead>
<tr>
<th>M</th>
<th>L</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21.30</td>
<td>21.30</td>
<td>21.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.263</td>
<td>2.262</td>
<td>2.262</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.263</td>
<td>2.258</td>
<td>2.250</td>
<td>2.250</td>
<td>2.265</td>
<td>2.244</td>
<td>2.248</td>
<td>2.245</td>
<td>2.246</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.263</td>
<td>2.258</td>
<td>2.250</td>
<td>2.246</td>
<td>2.249</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.258</td>
<td>2.248</td>
<td>2.248</td>
<td>2.244</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>21.44</td>
<td>21.64</td>
<td>21.75</td>
<td>21.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.257</td>
<td>2.253</td>
<td>2.244</td>
<td>2.243</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key: upper number is SNR (dB), lower number is entropy (bits/sample)
Table 5.3. Results for an 11 level quantizer, with M=5, embedded in a pitch removal loop.

<table>
<thead>
<tr>
<th>L</th>
<th>SNR inside pitch loop (dB)</th>
<th>SNR overall (dB)</th>
<th>Entropy (bits/sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13.75</td>
<td>18.41</td>
<td>1.372</td>
</tr>
<tr>
<td>2</td>
<td>14.01</td>
<td>18.63</td>
<td>1.355</td>
</tr>
<tr>
<td>3</td>
<td>14.29</td>
<td>18.89</td>
<td>1.369</td>
</tr>
<tr>
<td>4</td>
<td>14.36</td>
<td>18.97</td>
<td>1.353</td>
</tr>
<tr>
<td>5</td>
<td>14.22</td>
<td>18.83</td>
<td>1.350</td>
</tr>
<tr>
<td>6</td>
<td>14.42</td>
<td>19.05</td>
<td>1.353</td>
</tr>
</tbody>
</table>
5.3 Adaptive Tree Coding

5.3.1 Description

Adaptive tree coding was developed as an attempt to gain the improvement in performance of the (M,L) algorithm, without using as many computations. It was felt that the performance improvement was desirable, but the large number of computations was a costly trade-off, and, more importantly, unable to be done in real time. This lead to the search for some way of improving the performance/computation ratio.

Adaptive tree coding was the result of that search. It is based on the fact that the receiver does not need to know to what extent, if any, that tree coding is being performed at the transmitter. The receiver acts only upon the quantizer levels it receives; it does not matter how those quantizer levels were arrived at. So the basic concept of adaptive tree coding is to use tree coding only when it appears to make sense.

Two strategies for adaptive tree coding were developed from this basic concept. The first strategy was to perform additional tree pruning, so that a node would have to have a reasonable chance of being selected to be kept. Specifically, a node would be pruned if the value of the criterion for it were worse than the value of the criterion for the best node, multiplied by an arbitrary factor. In this way, when growing the next level of nodes, time would not be spent growing nodes which had a high probability of being pruned eventually.

The second strategy was to "turn off" the coding (by setting M equal to 1) when the system was performing well, and "turn it back on" when it was not performing well. The idea behind this strategy was that if the system were
performing well, there was little to be gained from tree coding. This strategy was implemented by checking the value of the criterion for the best new node was better than some arbitrary threshold. If, on the other hand, tree coding was not in use, then tree coding would not be used until the value of the criterion for the best new node was worse than some second arbitrary threshold.

Both of these strategies were employed in our simulation of adaptive tree coding, because they are somewhat complementary in nature: the second strategy provides "course tuning", and the first strategy provides "fine tuning".
5.3.2 Results

Results were first obtained for adaptive tree coding using only the second ("threshold") strategy. Some typical results from these runs are shown in Table 5.4. Tree coding was used for 26% of the samples in Sentence 1, and for 14% of the samples in Sentence 11. The results indicate the ability of adaptive tree coding to improve performance with a small increase in computation.

Results were next obtained for adaptive tree coding using only the first ("factor") strategy. A sample of the results are shown in Table 5.5. On these runs, Sentence 1 was used as the input, and M was set to 5. A measure of the decrease in computations is "effective M", which is the average number of nodes retained for each new sample. It can clearly be seen that the number of calculations can be decreased dramatically while still retaining much of the increased performance. In fact, it can be seen from the data that the performance was improved by adaptive tree coding. One possible explanation for this phenomenon would be that for those cases where adaptive tree coding increased performance, that the additional pruning eliminated nodes which appeared to be good, but were not in the long run.

Finally, a series of simulation runs were made which utilized both adaptive strategies simultaneously. The results indicated that the two strategies were complementary, in that use of both was better than the use of either alone. Some representative results are shown in Table 5.6. It can be seen that about the same performance is obtained by adaptive tree coding as with tree coding, with about a 66% reduction in computations.
Table 5.4 Results using the "threshold" strategy for adaptive tree coding.

Sentence 1 (male speaker):

<table>
<thead>
<tr>
<th></th>
<th>PARC</th>
<th>Adaptive tree coding</th>
<th>Tree coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNR inside pitch loop (dB)</td>
<td>13.77</td>
<td>14.13</td>
<td>14.42</td>
</tr>
<tr>
<td>SNR overall (dB)</td>
<td>18.42</td>
<td>18.75</td>
<td>19.04</td>
</tr>
<tr>
<td>Entropy (bits/sample)</td>
<td>1.374</td>
<td>1.374</td>
<td>1.356</td>
</tr>
</tbody>
</table>

Sentence 11 (female speaker):

<table>
<thead>
<tr>
<th></th>
<th>PARC</th>
<th>Adaptive tree coding</th>
<th>Tree coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNR inside pitch loop (dB)</td>
<td>12.84</td>
<td>13.35</td>
<td>13.64</td>
</tr>
<tr>
<td>SNR overall (dB)</td>
<td>20.88</td>
<td>21.40</td>
<td>21.70</td>
</tr>
<tr>
<td>Entropy (bits/sample)</td>
<td>1.386</td>
<td>1.374</td>
<td>1.338</td>
</tr>
</tbody>
</table>
### Table 5.5

Results using the "factor" strategy for adaptive tree coding.

<table>
<thead>
<tr>
<th>FACTOR</th>
<th>Overall SNR(dB)</th>
<th>Entropy (bits/sample)</th>
<th>Effective M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.E6 (60dB)</td>
<td>18.92</td>
<td>1.347</td>
<td>5.00</td>
</tr>
<tr>
<td>10.00(10dB)</td>
<td>18.92</td>
<td>1.347</td>
<td>4.46</td>
</tr>
<tr>
<td>7.94(9dB)</td>
<td>18.92</td>
<td>1.348</td>
<td>4.43</td>
</tr>
<tr>
<td>6.31(8dB)</td>
<td>18.92</td>
<td>1.348</td>
<td>4.39</td>
</tr>
<tr>
<td>5.01(7dB)</td>
<td>18.92</td>
<td>1.348</td>
<td>4.33</td>
</tr>
<tr>
<td>3.98(6dB)</td>
<td>18.92</td>
<td>1.348</td>
<td>4.24</td>
</tr>
<tr>
<td>3.16(5dB)</td>
<td>18.94</td>
<td>1.349</td>
<td>4.10</td>
</tr>
<tr>
<td>2.51(4dB)</td>
<td>19.09</td>
<td>1.343</td>
<td>3.84</td>
</tr>
<tr>
<td>2.00(3dB)</td>
<td>19.08</td>
<td>1.348</td>
<td>3.41</td>
</tr>
<tr>
<td>1.58(2dB)</td>
<td>19.07</td>
<td>1.356</td>
<td>2.62</td>
</tr>
<tr>
<td>1.50(1.75dB)</td>
<td>19.05</td>
<td>1.354</td>
<td>2.40</td>
</tr>
<tr>
<td>1.41(1.50dB)</td>
<td>18.94</td>
<td>1.360</td>
<td>2.10</td>
</tr>
<tr>
<td>1.33(1.25dB)</td>
<td>18.94</td>
<td>1.347</td>
<td>1.85</td>
</tr>
<tr>
<td>1.26(1dB)</td>
<td>18.82</td>
<td>1.354</td>
<td>1.58</td>
</tr>
<tr>
<td>1.00(0dB)</td>
<td>18.42</td>
<td>1.374</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Table 5.6 Results using both strategies for adaptive tree searching.

<table>
<thead>
<tr>
<th></th>
<th>Overall SNR (dB)</th>
<th>Entropy (bits/sample)</th>
<th>Effective</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARC</td>
<td>18.42</td>
<td>1.374</td>
<td>1.00</td>
</tr>
<tr>
<td>Adaptive tree coding</td>
<td>18.92</td>
<td>1.360</td>
<td>1.51</td>
</tr>
<tr>
<td>HITHR = 2416 (15dB), LOTHR = 304.1 (24dB), FACTOR = 1.50 (1.75dB)</td>
<td>18.92</td>
<td>1.347</td>
<td>5.00</td>
</tr>
<tr>
<td>Tree coding</td>
<td>18.92</td>
<td>1.347</td>
<td>5.00</td>
</tr>
</tbody>
</table>
3.4 Conclusions and suggestions for further research

It would appear that on the basis of the results obtained, tree coding is an effective way to increase the performance of a quantization system. The major question though, is not whether it is effective, but how effective it is relative to what it costs. It would appear that tree coding may not be effective in this sense, but that some form of adaptive tree coding may be. The question was moot for this project, because there was not sufficient real time to perform a tree coding version of PARC.

The reason why tree coding appears to be relatively ineffective with PARC may be that the ARC algorithm has been overly adapted for PARC — that is a set of parameters optimized for PARC will most probably be a poor choice to be used with tree coding. This makes a lot of sense if you think about how tree coding can help a system. For the PARC algorithm to work well, the parameters are chosen so that typically only one quantizer level results in a reasonable amount of error — to do otherwise would be suboptimal for PARC, since only one quantizer level can be selected, and having more than one reasonable quantizer level would simply reduce the dynamic range of the quantizer or increase the average quantization error. In contrast, for tree coding to be effective, several quantizer levels should be reasonable for each sample.

In order to modify the ARC parameters to make tree coding more effective then, it would appear that it would be desirable to select parameters which would decrease granular noise and let the tree coding reduce slope overload noise. Specifically, it would seem to be desirable to "move in" the output and scaling factors of the quantizer, decrease the updating gains, and reduce the "time constants" of the system. In this way, there would be a
richer selection of quantizer levels for the tree code.

There are also several other areas to be researched. In our simulations, the tree coding took place entirely within the pitch loop. It might be interesting, therefore, to investigate the effect of basing the pruning criterion on the reconstructed speech rather than the reconstructed depitched speech. Doing this might aid in the smooth transition from one pitch block to the next. Another idea which could be investigated would be delayed updating of the ARC algorithm. In delayed updating, the algorithm would be updated on the basis of the quantizer level just decided upon (corresponding to the sample L samples earlier), so that the updating could be done once for all nodes, eliminating many computations.

Tree coding might also be made more effective by making changes in the tree coding algorithm itself. One possible path for investigation would be in the area of variable symbol release, as developed by Goris. Another possible path would involve investigating different forms of the pruning criterion. For example, it might improve performance to weigh the contribution from earlier samples more heavily than that from recent samples, because it would appear that the "soft" decisions become "harder" as time passes.
5.5 References


CHAPTER 6
BACKWARD PITCH EXTRACTION ADAPTIVE RESIDUAL CODER (BPARC)

6.1 Introduction

Many of the practical systems for digitizing speech are variants of differential pulse code modulation (DPCM). The speech coder developed for the 9.6 Kbs bit rate uses this structure augmented by a pitch extraction loop. This algorithm is called Pitch extraction Adaptive Residual Coder (PARC).

The system described in the chapter is identical to PARC except for the method of pitch extraction. In PARC, pitch is extracted block by block from raw speech. Once the correlation coefficient $\beta$ and pitch period $T$ are known, reduced speech is formed, processed by the coder and transmitted. The $\beta$'s and $T$'s also need to be transmitted; the number of bits required to transmit these parameters depends on the number of parameters available (this depends on block length), type of coding employed and the way they are transmitted. In addition to the number of bits required for transmission, the transmission of these parameters necessitates a framing of the bit stream and an associated frame synchronization problem.

In order to avoid this framing problem, a backward adaptive approach was investigated which would not require transmission of $\beta$'s and $T$'s. Since the values of $\beta$ and $T$ change very little in a given voiced region, these parameters can be calculated for a previous block of speech and used for the current block to reconstruct speech. As noted in an earlier chapter, short pitch blocks provide the best performance. However, these short blocks require the transmission of a large amount of side information,
and T's, making them impractical. The use of the backward identification of β and T eliminates this problem.

In following sections, this approach referred to as Backward PARC (BPARC) is described along with various computer simulation studies comparing it with the standard PARC algorithm. A complete listing of the source program and a flow chart for the simulation are also included in this chapter.
6.2 System Structure

Figure 6.1 shows the block diagram of the backward adaptive pitch extraction residual coder. A comparison of this figure with Fig. 2.1 reveals the obvious difference that \( \hat{\beta} \) and \( T \) are now computed from \( \hat{s}(k) \)'s rather than \( s(k) \)'s. As a result of this change, it is no longer necessary to transmit \( \hat{\beta} \) and \( T \) since, in case of no transmission errors, the receiver can carry out the same computation. Note that the receiver must now be more complex since it must be capable of computing \( \hat{\beta} \) and \( T \).

The use of \( \hat{s}(k) \) to compute \( \hat{\beta} \) and \( T \) causes another less obvious change in the transmitter. In order to use \( \hat{s}(k) \)'s, the computation of \( \hat{\beta} \) and \( T \) must be based only on past speech estimates. In the forward adaptive case, the computation of \( \beta \) and \( T \) is based on both future and past speech samples. The basic approach for BPARC is to compute \( \beta \) and \( T \) on a block of \( \hat{s}(k) \), \( k = k_o, k_o-1, \ldots, k_o-K_c \) and then to use this value of \( \beta \) and \( T \) to form reduced speech \( v(k) \) for \( k = k_o+1, k_o+2, \ldots, k_o+K_u \). Here \( K_c \) is called the computation block size and \( K_u \) is the use block size; these two values are not necessarily equal.

The basic philosophy of the BPARC approach is that in voiced segments, where pitch redundancy removal is most effective, \( \beta \) and \( T \) does not change rapidly. Table 6.1 illustrates this fact with a listing of \( \beta \) and \( T \) for a typical segment of voiced speech. The use of \( \beta \) and \( T \), computed for one block, in the next block will not have much effect on the performance in such a voiced segment.

There will, however, be rapid changes in \( \beta \) and \( T \) during transitions from voiced to unvoiced or unvoiced to voiced speech. During these transitions, significant performance degradation can be expected in BPARC.
Table 6.1

Values of $\beta$ and $T$ for a Segment of Voiced Speech

<table>
<thead>
<tr>
<th>Sample Number</th>
<th>$\beta$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5420</td>
<td>0.91</td>
<td>30</td>
</tr>
<tr>
<td>5520</td>
<td>0.86</td>
<td>30</td>
</tr>
<tr>
<td>5550</td>
<td>0.92</td>
<td>30</td>
</tr>
<tr>
<td>5610</td>
<td>0.99</td>
<td>30</td>
</tr>
<tr>
<td>5640</td>
<td>1.04</td>
<td>30</td>
</tr>
<tr>
<td>5700</td>
<td>0.98</td>
<td>30</td>
</tr>
<tr>
<td>5730</td>
<td>0.98</td>
<td>30</td>
</tr>
<tr>
<td>5760</td>
<td>1.02</td>
<td>30</td>
</tr>
<tr>
<td>5790</td>
<td>1.03</td>
<td>30</td>
</tr>
<tr>
<td>5810</td>
<td>1.00</td>
<td>30</td>
</tr>
<tr>
<td>5850</td>
<td>1.02</td>
<td>30</td>
</tr>
<tr>
<td>5880</td>
<td>1.00</td>
<td>30</td>
</tr>
<tr>
<td>5910</td>
<td>1.06</td>
<td>30</td>
</tr>
<tr>
<td>5940</td>
<td>1.06</td>
<td>30</td>
</tr>
<tr>
<td>5970</td>
<td>1.04</td>
<td>30</td>
</tr>
<tr>
<td>6000</td>
<td>1.09</td>
<td>30</td>
</tr>
<tr>
<td>6030</td>
<td>1.08</td>
<td>30</td>
</tr>
<tr>
<td>6060</td>
<td>1.09</td>
<td>30</td>
</tr>
<tr>
<td>6090</td>
<td>1.05</td>
<td>30</td>
</tr>
<tr>
<td>6120</td>
<td>1.06</td>
<td>30</td>
</tr>
<tr>
<td>6150</td>
<td>1.06</td>
<td>30</td>
</tr>
<tr>
<td>6180</td>
<td>1.06</td>
<td>30</td>
</tr>
<tr>
<td>6210</td>
<td>1.06</td>
<td>30</td>
</tr>
<tr>
<td>6240</td>
<td>1.06</td>
<td>30</td>
</tr>
<tr>
<td>6270</td>
<td>1.03</td>
<td>30</td>
</tr>
<tr>
<td>6300</td>
<td>1.02</td>
<td>30</td>
</tr>
<tr>
<td>6330</td>
<td>1.02</td>
<td>30</td>
</tr>
<tr>
<td>6360</td>
<td>1.00</td>
<td>30</td>
</tr>
<tr>
<td>6390</td>
<td>0.98</td>
<td>30</td>
</tr>
<tr>
<td>6420</td>
<td>0.99</td>
<td>30</td>
</tr>
<tr>
<td>6450</td>
<td>0.97</td>
<td>30</td>
</tr>
<tr>
<td>6480</td>
<td>0.95</td>
<td>30</td>
</tr>
<tr>
<td>6510</td>
<td>0.93</td>
<td>30</td>
</tr>
<tr>
<td>6540</td>
<td>0.94</td>
<td>30</td>
</tr>
<tr>
<td>6570</td>
<td>0.94</td>
<td>30</td>
</tr>
<tr>
<td>6600</td>
<td>0.87</td>
<td>30</td>
</tr>
<tr>
<td>6630</td>
<td>0.89</td>
<td>30</td>
</tr>
<tr>
<td>6660</td>
<td>0.85</td>
<td>30</td>
</tr>
<tr>
<td>6690</td>
<td>0.84</td>
<td>30</td>
</tr>
</tbody>
</table>
Fig. 6.2 Backward Pitch extraction Adaptive Residual Coder (BPARC)

1. Start
2. Define "Computation" and "Use" blocks KBLKR, KBLKT
3. Read speech samples s(k)
4. Compute total number of blocks NBLK
5. Compute block number I
6. Calculate first sample of new block IBNDRY
7. Is sample number equal to first sample of new block?
   - yes: calculate $\hat{S}$ and $T$ for this block, but use them for current block
   - no: calculate $\hat{S}$ and $T$ for this block, but use them for current block
8. Call subroutine INLOOP which returns $\hat{v}(k)$
9. Construct reduced speech $\hat{v}(k)$
10. Reconstruct speech $\hat{s}(k)$
11. Calculate signal energy, noise energy, and reduced speech energy
12. Are all samples done?
   - yes: Compute SNR and SER
   - no: Call subroutine INEND which returns Inloop SNR Entropy
13. Stop
over the PARC algorithm. The hope was that the decrease in quantization noise caused by removing the $\beta$ and $T$ transmission would offset this degradation. The advantage of not requiring framing for $\beta$ and $T$ is also obvious.

Exactly the same method, namely, Eqs. (2.6) and (2.7) are used to compute $\beta$ and $T$ for the BPARC as for the standard PARC except $\hat{s}$ is used in place of $s$ and all $\hat{s}$'s are past values. Hence $T$ at stage $k$ is the value of $\tau$ which minimizes the AMDF function given by

$$A(\tau) = \frac{1}{K_c} \left[ \sum_{j=k-K_c}^{k} |\hat{s}(j) - \hat{s}(j-\tau)| \right]$$  \hspace{1cm} (6.1)

$$\tau = 20, 21, \ldots, T_{\text{max}}$$

Once $T$ is known, $\beta$ is determined from

$$\beta = \frac{\sum_{j=k-K_c}^{k} \hat{s}(j)\hat{s}(j-T)}{\sum_{j=k-K_c}^{k} \hat{s}(j-T)\hat{s}(j-T)}$$  \hspace{1cm} (6.2)

A block diagram for the complete BPARC algorithm is given in Fig. 6.2.

This algorithm assumes that a complete sentence is read in and then processed. This algorithm was programmed on the PDP-11/60 in order to determine its performance characteristics.
6.3 Performance Evaluation and Parametric Studies

As the first step in evaluating the performance of the BPARC algorithm, a comparison of $\beta$, $T$ and SER using the normal (forward) PARC and the BPARC algorithms was made. Table 6.2 shows a typical portion of the results from this study. Note that SER is significantly degraded in the transition regions at the beginning and end. There is some decrease (-1 db) in SER in the voiced segment. A computation block length of 30 samples was used for both cases. This degradation in SER is understandable since we use $\beta$ and $T$ which were calculated from previously processed speech are used for the present block of samples. In transition regions such as V/UV, UV/S, S/V, the previously processed speech block can be much different than the current block of speech to be processed.

A study of the effect of different lengths of "computation block" and "use block" on SNR was conducted next. In general for very large "computation" or "use" block SNR goes down while for smaller block lengths increase in performance was noticed. However, there is not a monotonic increase or decrease in performance noticed with a decrease or increase in block lengths. The results are tabulated in Table 6.3.

From the very basic idea of BPARC, it is clear that this approach should work better for voiced speech. Exactly the same thing was observed when the range of possible $\beta$'s was limited to correspond to voiced speech. Table 6.4 shows the improvement in SNR by restricting $\beta$.

Table 6.4

Effect of Limit Range for $\beta$
(Sent 11, Female Speaker, $K_u = K_c = 30$)

<table>
<thead>
<tr>
<th></th>
<th>$-2 &lt; \beta &lt; 2$</th>
<th>$0.74 &lt; \beta &lt; 1.14$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNR</td>
<td>19.96 db</td>
<td>20.69 db</td>
</tr>
<tr>
<td>SNR(inloop)</td>
<td>15.00 db</td>
<td>14.28 db</td>
</tr>
<tr>
<td>SER</td>
<td>4.96 db</td>
<td>6.41 db</td>
</tr>
<tr>
<td>H</td>
<td>1.51 b/sample</td>
<td>1.46 b/sample</td>
</tr>
</tbody>
</table>
Table 6.2
Comparison of $\beta, T$ and SER for PARC and BPARC (Sent 11)

<table>
<thead>
<tr>
<th>Sample number</th>
<th>PARC $\beta$</th>
<th>PARC $T$</th>
<th>PARC Signal Energy Reduction</th>
<th>BPARC $\beta$</th>
<th>BPARC $T$</th>
<th>BPARC Signal Energy Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>5310</td>
<td>8.37</td>
<td>29</td>
<td>1.11 DB</td>
<td>0.88</td>
<td>23</td>
<td>0.08 DB</td>
</tr>
<tr>
<td>5320</td>
<td>2.00</td>
<td>26</td>
<td>4.36 DB</td>
<td>0.89</td>
<td>22</td>
<td>0.08 DB</td>
</tr>
<tr>
<td>5376</td>
<td>1.32</td>
<td>26</td>
<td>7.71 DB</td>
<td>0.80</td>
<td>26</td>
<td>0.08 DB</td>
</tr>
<tr>
<td>5400</td>
<td>8.91</td>
<td>20</td>
<td>6.90 DB</td>
<td>0.89</td>
<td>27</td>
<td>2.54 DB</td>
</tr>
<tr>
<td>5430</td>
<td>7.93</td>
<td>29</td>
<td>13.54 DB</td>
<td>0.85</td>
<td>29</td>
<td>3.67 DB</td>
</tr>
<tr>
<td>5460</td>
<td>8.90</td>
<td>20</td>
<td>11.00 DB</td>
<td>0.89</td>
<td>29</td>
<td>5.05 DB</td>
</tr>
<tr>
<td>5476</td>
<td>0.91</td>
<td>30</td>
<td>16.55 DB</td>
<td>0.80</td>
<td>30</td>
<td>14.72 DB</td>
</tr>
<tr>
<td>5520</td>
<td>8.80</td>
<td>30</td>
<td>15.54 DB</td>
<td>0.87</td>
<td>30</td>
<td>10.94 DB</td>
</tr>
<tr>
<td>5550</td>
<td>8.86</td>
<td>30</td>
<td>10.35 DB</td>
<td>0.89</td>
<td>30</td>
<td>14.98 DB</td>
</tr>
<tr>
<td>5590</td>
<td>8.92</td>
<td>30</td>
<td>14.98 DB</td>
<td>0.90</td>
<td>30</td>
<td>15.57 DB</td>
</tr>
<tr>
<td>5610</td>
<td>0.98</td>
<td>30</td>
<td>15.70 DB</td>
<td>0.91</td>
<td>30</td>
<td>14.82 DB</td>
</tr>
<tr>
<td>5640</td>
<td>0.95</td>
<td>30</td>
<td>14.46 DB</td>
<td>0.95</td>
<td>30</td>
<td>16.37 DB</td>
</tr>
<tr>
<td>5670</td>
<td>1.04</td>
<td>30</td>
<td>18.42 DB</td>
<td>0.96</td>
<td>30</td>
<td>19.16 DB</td>
</tr>
<tr>
<td>5700</td>
<td>0.98</td>
<td>30</td>
<td>16.69 DB</td>
<td>0.97</td>
<td>30</td>
<td>19.33 DB</td>
</tr>
<tr>
<td>5730</td>
<td>0.96</td>
<td>30</td>
<td>18.80 DB</td>
<td>0.99</td>
<td>30</td>
<td>20.15 DB</td>
</tr>
<tr>
<td>5760</td>
<td>1.02</td>
<td>30</td>
<td>17.09 DB</td>
<td>1.00</td>
<td>30</td>
<td>16.39 DB</td>
</tr>
<tr>
<td>5790</td>
<td>1.03</td>
<td>30</td>
<td>17.40 DB</td>
<td>1.01</td>
<td>30</td>
<td>19.69 DB</td>
</tr>
<tr>
<td>5920</td>
<td>1.00</td>
<td>30</td>
<td>17.38 DB</td>
<td>1.02</td>
<td>30</td>
<td>19.72 DB</td>
</tr>
<tr>
<td>5950</td>
<td>0.96</td>
<td>30</td>
<td>20.45 DB</td>
<td>0.99</td>
<td>30</td>
<td>18.55 DB</td>
</tr>
<tr>
<td>5980</td>
<td>0.96</td>
<td>30</td>
<td>18.70 DB</td>
<td>1.00</td>
<td>30</td>
<td>18.96 DB</td>
</tr>
<tr>
<td>6000</td>
<td>1.06</td>
<td>30</td>
<td>17.62 DB</td>
<td>1.02</td>
<td>30</td>
<td>17.08 DB</td>
</tr>
<tr>
<td>6030</td>
<td>1.06</td>
<td>30</td>
<td>18.61 DB</td>
<td>1.02</td>
<td>30</td>
<td>16.07 DB</td>
</tr>
<tr>
<td>6060</td>
<td>1.09</td>
<td>30</td>
<td>16.66 DB</td>
<td>1.07</td>
<td>30</td>
<td>19.13 DB</td>
</tr>
<tr>
<td>6090</td>
<td>1.05</td>
<td>30</td>
<td>18.04 DB</td>
<td>1.09</td>
<td>30</td>
<td>15.74 DB</td>
</tr>
<tr>
<td>6120</td>
<td>1.06</td>
<td>30</td>
<td>17.41 DB</td>
<td>1.06</td>
<td>30</td>
<td>14.31 DB</td>
</tr>
<tr>
<td>6150</td>
<td>1.06</td>
<td>30</td>
<td>11.71 DB</td>
<td>1.09</td>
<td>30</td>
<td>12.02 DB</td>
</tr>
<tr>
<td>6180</td>
<td>1.06</td>
<td>30</td>
<td>13.62 DB</td>
<td>1.06</td>
<td>30</td>
<td>13.72 DB</td>
</tr>
<tr>
<td>6210</td>
<td>1.06</td>
<td>30</td>
<td>16.46 DB</td>
<td>1.08</td>
<td>30</td>
<td>14.62 DB</td>
</tr>
<tr>
<td>6240</td>
<td>1.06</td>
<td>30</td>
<td>14.30 DB</td>
<td>1.08</td>
<td>30</td>
<td>14.39 DB</td>
</tr>
<tr>
<td>6270</td>
<td>1.03</td>
<td>30</td>
<td>14.53 DB</td>
<td>1.05</td>
<td>30</td>
<td>14.83 DB</td>
</tr>
<tr>
<td>6300</td>
<td>1.02</td>
<td>30</td>
<td>16.95 DB</td>
<td>1.05</td>
<td>30</td>
<td>17.52 DB</td>
</tr>
<tr>
<td>6330</td>
<td>1.02</td>
<td>30</td>
<td>17.48 DB</td>
<td>1.03</td>
<td>30</td>
<td>16.48 DB</td>
</tr>
<tr>
<td>6360</td>
<td>1.00</td>
<td>30</td>
<td>17.05 DB</td>
<td>1.01</td>
<td>30</td>
<td>17.34 DB</td>
</tr>
<tr>
<td>6390</td>
<td>1.00</td>
<td>30</td>
<td>21.70 DB</td>
<td>1.01</td>
<td>30</td>
<td>21.29 DB</td>
</tr>
<tr>
<td>6420</td>
<td>0.90</td>
<td>30</td>
<td>17.56 DB</td>
<td>1.02</td>
<td>30</td>
<td>18.34 DB</td>
</tr>
<tr>
<td>6450</td>
<td>0.99</td>
<td>30</td>
<td>20.76 DB</td>
<td>0.99</td>
<td>30</td>
<td>19.43 DB</td>
</tr>
<tr>
<td>6480</td>
<td>0.97</td>
<td>30</td>
<td>18.49 DB</td>
<td>0.98</td>
<td>30</td>
<td>18.26 DB</td>
</tr>
<tr>
<td>6510</td>
<td>0.95</td>
<td>30</td>
<td>18.09 DB</td>
<td>0.97</td>
<td>30</td>
<td>19.37 DB</td>
</tr>
<tr>
<td>6540</td>
<td>0.93</td>
<td>30</td>
<td>13.91 DB</td>
<td>0.96</td>
<td>30</td>
<td>12.93 DB</td>
</tr>
<tr>
<td>6570</td>
<td>0.94</td>
<td>30</td>
<td>12.66 DB</td>
<td>0.94</td>
<td>30</td>
<td>13.74 DB</td>
</tr>
<tr>
<td>6600</td>
<td>0.94</td>
<td>30</td>
<td>12.80 DB</td>
<td>0.92</td>
<td>30</td>
<td>12.60 DB</td>
</tr>
<tr>
<td>6630</td>
<td>0.97</td>
<td>30</td>
<td>13.60 DB</td>
<td>0.95</td>
<td>30</td>
<td>12.54 DB</td>
</tr>
<tr>
<td>6660</td>
<td>0.99</td>
<td>30</td>
<td>12.98 DB</td>
<td>0.89</td>
<td>30</td>
<td>14.57 DB</td>
</tr>
<tr>
<td>6690</td>
<td>0.05</td>
<td>30</td>
<td>10.01 DB</td>
<td>0.88</td>
<td>30</td>
<td>10.66 DB</td>
</tr>
<tr>
<td>6720</td>
<td>0.04</td>
<td>30</td>
<td>14.74 DB</td>
<td>0.87</td>
<td>30</td>
<td>14.67 DB</td>
</tr>
<tr>
<td>6750</td>
<td>0.04</td>
<td>30</td>
<td>9.99 DB</td>
<td>0.87</td>
<td>30</td>
<td>14.67 DB</td>
</tr>
</tbody>
</table>
Table 6.3
Variation of Use and Computation Block Lengths

USE BLOCK LENGTH $K_u$ (Samples)

<table>
<thead>
<tr>
<th>USE BLOCK LENGTH $K_u$ (Samples)</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>19.06*</td>
<td>20.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.05</td>
<td>14.10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.01</td>
<td>5.96</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.53</td>
<td>1.51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>19.23</td>
<td>19.96</td>
<td>18.80</td>
<td></td>
</tr>
<tr>
<td>14.62</td>
<td>15.00</td>
<td>15.59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.61</td>
<td>4.96</td>
<td>3.22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.51</td>
<td>1.51</td>
<td>1.50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>19.25</td>
<td>19.43</td>
<td>19.15</td>
<td>19.39</td>
</tr>
<tr>
<td>15.11</td>
<td>14.82</td>
<td>15.27</td>
<td>14.76</td>
<td></td>
</tr>
<tr>
<td>4.15</td>
<td>4.61</td>
<td>3.88</td>
<td>4.62</td>
<td></td>
</tr>
<tr>
<td>1.48</td>
<td>1.49</td>
<td>1.50</td>
<td>1.49</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>19.46</td>
<td>20.06</td>
<td>19.22</td>
<td>18.34</td>
</tr>
<tr>
<td>14.91</td>
<td>14.12</td>
<td>15.02</td>
<td>14.97</td>
<td></td>
</tr>
<tr>
<td>4.55</td>
<td>5.94</td>
<td>4.21</td>
<td>3.37</td>
<td></td>
</tr>
<tr>
<td>1.49</td>
<td>1.50</td>
<td>1.50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>19.19</td>
<td>19.64</td>
<td>19.17</td>
<td>18.89</td>
</tr>
<tr>
<td>14.98</td>
<td>14.56</td>
<td>15.05</td>
<td>15.13</td>
<td></td>
</tr>
<tr>
<td>4.21</td>
<td>5.08</td>
<td>4.11</td>
<td>3.75</td>
<td></td>
</tr>
<tr>
<td>1.49</td>
<td>1.49</td>
<td>1.51</td>
<td>1.50</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>19.24</td>
<td>19.53</td>
<td>19.05</td>
<td>18.43</td>
</tr>
<tr>
<td>14.90</td>
<td>14.93</td>
<td>15.61</td>
<td>15.00</td>
<td></td>
</tr>
<tr>
<td>4.35</td>
<td>4.60</td>
<td>3.44</td>
<td>3.43</td>
<td></td>
</tr>
<tr>
<td>1.51</td>
<td>1.48</td>
<td>1.48</td>
<td>1.51</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>19.80</td>
<td>18.61</td>
<td>18.61</td>
<td>18.12</td>
</tr>
<tr>
<td>14.84</td>
<td>15.06</td>
<td>14.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.96</td>
<td>3.55</td>
<td>3.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.49</td>
<td>1.52</td>
<td>1.51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>18.90</td>
<td>18.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15.09</td>
<td>15.29</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.81</td>
<td>3.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.49</td>
<td>1.51</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Each block lists SNR (overall), SNR (inloop), SER and $\mu$. 
In the above approach, $f$ was set zero if it was not in the allowed range. It appeared that the resulting sharp changes in $f$ would degrade performance. To eliminate this problem a limit was set on the maximum percentage changes in $f$ so that $f$ would vary more smoothly in transition regions. This approach helped to increase SER as shown in Table 6.5 but not the performance as SNR inloop decreased.

Table 6.5
Effect of Limiting $\Delta f$
(Sent 1, male speaker, $K_c = 45$, $K_u = 30$, $0.65 < B < 1.25$)

| $\Delta f$ = 0 | $|\Delta f| < 10\%$ | $|\Delta f| < 20\%$ |
|----------------|-----------------|-----------------|
| SNR           | 17.78 db        | 17.57 db        | 17.70 db        |
| SNR(inloop)   | 14.84 db        | 14.45 db        | 14.67 db        |
| SER           | 2.94 db         | 3.12 db         | 3.02 db         |
| $H$           | 1.49 b/sample   | 1.50 b/sample   | 1.49 b/sample   |

To calculate the pitch parameter $T$ the function

$$A(T) = \sum_{j=1}^{K} |\hat{s}(j) - \hat{s}(j-T)|$$

was minimized with respect to $T$ for $20 \leq T \leq L$ where $L = K_c$ computation block length = 30. Different values of $L$ were tried. Some improvement in performance was noticed (see Table 6.6) if search range was decreased up to a certain point. For a long search range, the effect of errors that were made while reconstructing previous samples becomes more severe while a small search range may not be enough to detect correct period $T$.

Table 6.6
Effect of Search Range for $T$
(Sent 1 : male speaker, $K_u = K_c = 30$)

<table>
<thead>
<tr>
<th></th>
<th>$20 &lt; T &lt; 100$</th>
<th>$20 &lt; T &lt; 80$</th>
<th>$20 &lt; T &lt; 70$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNR</td>
<td>17.67 db</td>
<td>17.78 db</td>
<td>17.78 db</td>
</tr>
<tr>
<td>SNR(inloop)</td>
<td>14.87 db</td>
<td>14.84 db</td>
<td>14.75 db</td>
</tr>
<tr>
<td>SER</td>
<td>2.80 db</td>
<td>2.94 db</td>
<td>3.03 db</td>
</tr>
<tr>
<td>$H$</td>
<td>1.49 b/sample</td>
<td>1.49 b/sample</td>
<td>1.49 b/sample</td>
</tr>
</tbody>
</table>
As discussed earlier for the normal PARC algorithm, it is necessary to transmit β's and T's along with quantized residual reduced speech. This, of course, requires a few bits per sample. If a block size of 100 is used and β and T require 13 bits, this becomes 0.13 bits/sample. That leaves 1.37 bits/sample for transmission of other information. Hence, to make a fair comparison between the two algorithms, H was limited to 1.37 for PARC and 1.5 bits/sample for BPARC. Even though BPARC is not a clear winner (see Table 6.7), it is a very attractive solution to problems of transmitting β and T.

Table 6.7
Comparison of PARC and BPARC Algorithms

<table>
<thead>
<tr>
<th></th>
<th>SNR</th>
<th>SNR</th>
<th>SER</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>inloop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentence 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARC</td>
<td>19.11 db</td>
<td>13.63 db</td>
<td>5.48 db</td>
<td>1.37 b/sample</td>
</tr>
<tr>
<td>BPARC</td>
<td>17.78 db</td>
<td>14.84 db</td>
<td>2.94 db</td>
<td>1.49 b/sample</td>
</tr>
<tr>
<td>Sentence 11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARC</td>
<td>21.40 db</td>
<td>12.95 db</td>
<td>8.45 db</td>
<td>1.36 b/sample</td>
</tr>
<tr>
<td>BPARC</td>
<td>20.53 db</td>
<td>14.21 db</td>
<td>6.31 db</td>
<td>1.47 b/sample</td>
</tr>
</tbody>
</table>
6.4 Transmission Error Studies

No speech encoding algorithm is good unless it can tolerate with channel errors. To study the effects of channel errors, first step is to determine the effect of errors in quantizer levels. The errors that are introduced in quantizer levels do not exactly correspond to errors caused due to bit reversal but nevertheless they are a good measure of the algorithm's susceptibility to channel errors.

The following procedure was adopted to study effect of channel errors in the BPARC algorithm:

(i) The receiver program was extracted from transmitter program.

(ii) A quantizer output file was created with desired transmission errors.

(iii) This file was read by receiver and a reconstructed speech \( \hat{s}(k) \) file was created.

(iv) SNR was calculated between original speech and received speech.

The algorithm tolerated one transmission error (for male speaker SNR goes down from 19.52 to 19.48) but becomes unstable with additional errors. It was suspected that wrong \( \hat{s}'s \) lead to wrong \( \hat{e}'s \) and possibly wrong \( \hat{T}'s \) which in turn make \( \hat{s}'s \) wrong. To investigate the cause, the BPARC program was run with no pitch removal. The algorithm works very well (see Table 6.8) even with 1% transmission error rate thus confirming above doubt.

| Table 6.8 |
| Effect of Transmission Errors on BPARC with no Pitch Extractor |
| (Sentence 1: Male speaker, SNR with no transmission error = 18.83 db, \( H=2.18 \)) |

<table>
<thead>
<tr>
<th>Transmission error rate</th>
<th>SNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01%</td>
<td>18.82 db</td>
</tr>
<tr>
<td>0.1%</td>
<td>18.37 db</td>
</tr>
<tr>
<td>1%</td>
<td>4.25 db</td>
</tr>
</tbody>
</table>
A fixed predictor instead of adaptive predictor was also tried in the system. It was found that fixed predictor minimizes the effect of transmission errors to some extent.

Table 6.9 lists $\beta$'s and $T$'s with and without transmission error. Note that $\beta$ changes immediately after error has been introduced. The plots in Figs. 6.3 and 6.4 compare the effect of transmission error on local SNR.
Table 6.9
Effect of Transmission Error (Sentence 11: Female speaker)

Without transmission error

<table>
<thead>
<tr>
<th>Sample Number</th>
<th>( \beta )</th>
<th>( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1298</td>
<td>0.88</td>
<td>27</td>
</tr>
<tr>
<td>1328</td>
<td>0.84</td>
<td>27</td>
</tr>
<tr>
<td>1358</td>
<td>0.97</td>
<td>55</td>
</tr>
<tr>
<td>1388</td>
<td>0.98</td>
<td>27</td>
</tr>
<tr>
<td>1418</td>
<td>0.98</td>
<td>27</td>
</tr>
<tr>
<td>1448</td>
<td>0.95</td>
<td>27</td>
</tr>
<tr>
<td>1478</td>
<td>0.99</td>
<td>27</td>
</tr>
<tr>
<td>1508</td>
<td>0.99</td>
<td>27</td>
</tr>
<tr>
<td>1538</td>
<td>0.92</td>
<td>27</td>
</tr>
<tr>
<td>1568</td>
<td>0.94</td>
<td>27</td>
</tr>
<tr>
<td>1598</td>
<td>0.94</td>
<td>27</td>
</tr>
<tr>
<td>1628</td>
<td>0.97</td>
<td>55</td>
</tr>
<tr>
<td>1658</td>
<td>0.98</td>
<td>55</td>
</tr>
<tr>
<td>1688</td>
<td>1.01</td>
<td>55</td>
</tr>
<tr>
<td>1718</td>
<td>1.00</td>
<td>55</td>
</tr>
<tr>
<td>1748</td>
<td>1.02</td>
<td>55</td>
</tr>
<tr>
<td>1778</td>
<td>1.02</td>
<td>55</td>
</tr>
<tr>
<td>1808</td>
<td>1.08</td>
<td>55</td>
</tr>
<tr>
<td>1838</td>
<td>0.93</td>
<td>55</td>
</tr>
<tr>
<td>1868</td>
<td>0.99</td>
<td>55</td>
</tr>
<tr>
<td>1898</td>
<td>0.81</td>
<td>27</td>
</tr>
<tr>
<td>1928</td>
<td>0.83</td>
<td>27</td>
</tr>
<tr>
<td>1958</td>
<td>0.87</td>
<td>27</td>
</tr>
<tr>
<td>1988</td>
<td>0.91</td>
<td>27</td>
</tr>
<tr>
<td>2018</td>
<td>0.91</td>
<td>27</td>
</tr>
<tr>
<td>2048</td>
<td>0.98</td>
<td>27</td>
</tr>
<tr>
<td>2078</td>
<td>0.98</td>
<td>27</td>
</tr>
<tr>
<td>2108</td>
<td>0.97</td>
<td>27</td>
</tr>
<tr>
<td>2138</td>
<td>0.98</td>
<td>27</td>
</tr>
<tr>
<td>2168</td>
<td>0.78</td>
<td>57</td>
</tr>
<tr>
<td>2198</td>
<td>0.78</td>
<td>28</td>
</tr>
<tr>
<td>2228</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2258</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2288</td>
<td>0.88</td>
<td>25</td>
</tr>
<tr>
<td>2318</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2348</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2378</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2408</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2438</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2468</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2498</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2528</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2558</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2588</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2618</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2648</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2678</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2708</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2738</td>
<td>0.88</td>
<td>28</td>
</tr>
</tbody>
</table>

With transmission error at sample number 1674, 7075

<table>
<thead>
<tr>
<th>Sample Number</th>
<th>( \beta )</th>
<th>( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1298</td>
<td>0.88</td>
<td>27</td>
</tr>
<tr>
<td>1328</td>
<td>0.94</td>
<td>27</td>
</tr>
<tr>
<td>1358</td>
<td>0.97</td>
<td>55</td>
</tr>
<tr>
<td>1388</td>
<td>0.98</td>
<td>27</td>
</tr>
<tr>
<td>1418</td>
<td>0.98</td>
<td>27</td>
</tr>
<tr>
<td>1448</td>
<td>0.95</td>
<td>27</td>
</tr>
<tr>
<td>1478</td>
<td>0.99</td>
<td>27</td>
</tr>
<tr>
<td>1508</td>
<td>0.99</td>
<td>27</td>
</tr>
<tr>
<td>1538</td>
<td>0.92</td>
<td>27</td>
</tr>
<tr>
<td>1568</td>
<td>0.94</td>
<td>27</td>
</tr>
<tr>
<td>1598</td>
<td>0.94</td>
<td>27</td>
</tr>
<tr>
<td>1628</td>
<td>0.97</td>
<td>55</td>
</tr>
<tr>
<td>1658</td>
<td>0.98</td>
<td>55</td>
</tr>
<tr>
<td>1688</td>
<td>1.01</td>
<td>55</td>
</tr>
<tr>
<td>1718</td>
<td>1.00</td>
<td>55</td>
</tr>
<tr>
<td>1748</td>
<td>1.02</td>
<td>55</td>
</tr>
<tr>
<td>1778</td>
<td>1.02</td>
<td>55</td>
</tr>
<tr>
<td>1808</td>
<td>1.08</td>
<td>55</td>
</tr>
<tr>
<td>1838</td>
<td>0.93</td>
<td>55</td>
</tr>
<tr>
<td>1868</td>
<td>0.99</td>
<td>55</td>
</tr>
<tr>
<td>1898</td>
<td>0.81</td>
<td>27</td>
</tr>
<tr>
<td>1928</td>
<td>0.83</td>
<td>27</td>
</tr>
<tr>
<td>1958</td>
<td>0.87</td>
<td>27</td>
</tr>
<tr>
<td>1988</td>
<td>0.91</td>
<td>27</td>
</tr>
<tr>
<td>2018</td>
<td>0.91</td>
<td>27</td>
</tr>
<tr>
<td>2048</td>
<td>0.98</td>
<td>27</td>
</tr>
<tr>
<td>2078</td>
<td>0.98</td>
<td>27</td>
</tr>
<tr>
<td>2108</td>
<td>0.97</td>
<td>27</td>
</tr>
<tr>
<td>2138</td>
<td>0.98</td>
<td>27</td>
</tr>
<tr>
<td>2168</td>
<td>0.78</td>
<td>57</td>
</tr>
<tr>
<td>2198</td>
<td>0.78</td>
<td>28</td>
</tr>
<tr>
<td>2228</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2258</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2288</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2318</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2348</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2378</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2408</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2438</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2468</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2498</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2528</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2558</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2588</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2618</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2648</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2678</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2708</td>
<td>0.88</td>
<td>28</td>
</tr>
<tr>
<td>2738</td>
<td>0.88</td>
<td>28</td>
</tr>
</tbody>
</table>
LOCAL SNR (DB) VS. LOCAL SIGNAL (DB)
SENT 11 FEMALE SPEAKER, NO TRANS ERROR.
SNR = 21.53  SNRSEG = 16.15
LOCAL SNR (DB) VS. LOCAL SIGNAL (DB)
SENT11: FEMALE SPEAKER, 2 TRANS ERRORS.
SNR = -13.68  SNRSEG = -3.11
6.5 Conclusions

The study presented in this chapter of the BPARC algorithm shows that it is an attractive approach for a fairly error-free channel but not suitable for channel with at least 0.1% error rate. The following changes in algorithm might help to make it more robust.

1. Currents $S$ and $T$ are calculated for computation block (using received speech) and then used for the current block to construct speech. It is now possible to use the new received speech samples for current block and re-calculate $S$ and $T$ using those samples. These values of $S$ and $T$ should be closer to true values for that block.

2. Maximum changes in $S$ could be limited thus reducing effect of channel error.

3. Errors in pitch periods (in voiced region) due to channel errors could be known by looking at $T$'s of previous blocks and then could be corrected.

Unfortunately time did not permit an examination of these ideas.
CHAPTER 7
TANDEM OPERATION

7.1 Introduction

One of the requirements of the speech coding algorithm is that it should perform satisfactorily in tandem with a CVSD speech coder operating at a data rate of 16 Kb/s and this tandem configuration should provide speech intelligibility with minimal degradation compared with a single link of CVSD operating at 16 Kb/s. The simulation of this tandem operation was done and results are discussed in Sec. 7.2.

Another requirement of speech coding algorithm is that it should produce intelligible speech under acoustic background noise. The simulation of background noise and the performance of PARC are discussed in Sec. 7.3.
7.2 PARC in Tandem with CVSD

The performance of the PARC system in tandem with CVSD system was studied. This study included passing raw speech through CVSD algorithm to create a file of CVSD output speech and using this speech file as input to the PARC algorithm. The output of the PARC algorithm becomes the output of the tandem system CVSD-PARC as shown in Fig. 7.1a. Similarly the PARC-CVSD tandem connection shown in Fig. 7.1b was also studied. The CVSD algorithm used for the study of tandem operation is described in Appendix D.

The CVSD algorithm used in this study operates on input speech sampled at the rate of 16K samples per second. This means the sampling rates at the input and the output of CVSD must be modified in order to make the tandem connections with PARC which operates on speech sampled at 6.4 KHz. The resampling can be done by using resampling programs listed in Appendix D. The CVSD program has incorporated this resampling program which makes simulation less time consuming.

The performance of these tandem connections are judged by Signal to quantization Noise Ratio and the subjective criterion. Sentence 1, "Cats and Dogs each hate the other", spoken by a male speaker, was used for simulation. Results are reported in Table 7.1 and 7.2.

TABLE 7.1 SNR for PARC, CVSD and their interconnections.

<table>
<thead>
<tr>
<th></th>
<th>PARC</th>
<th>CVSD</th>
<th>PARC-CVSD tandem</th>
<th>CVSD-PARC tandem</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNR</td>
<td>18.31</td>
<td>11.78</td>
<td>10.80</td>
<td>10.68</td>
</tr>
</tbody>
</table>


Fig. 7.1a CVSD in Tandem with PARC

Fig. 7.1b PARC in Tandem with CVSD
The results in Table 7.1 show that there is little degradation in SNR due to tandem operations. In fact, the speech quality of CVSD-PARC tandem seems to be better than CVSD alone in terms of perception. This study indicated that PARC acts as a filter for the granular noise in the CVSD output. The performance of these tandem operations could be improved by redesigning various parameters used in PARC. However, at this point the purpose of the study was to make sure that PARC algorithm performs reasonably well in tandem with CVSD.

It can be seen from Table 7.2 that the SNR decreased by less than 2 db by inputting the CVSD speech instead of raw speech. This decrease in SNR could be attributed to the decrease in the predictor performance as a result of high frequency contents of CVSD speech and poor signal energy reduction due to decrease in correlation between CVSD speech samples.

However, this decrease in SNR is surprisingly low. This shows that the PARC algorithm works very well for noisy input speech except for some increase in entropy. The entropy increase is a rather serious problem but solvable by using buffer control techniques which are discussed in details in Chapter 10.
7.3 Effect of Background Speakers on PARC Performance

It has been observed that PARC algorithm performs satisfactorily well in tandem with CVSD. However, the noise in CVSD speech is white. It is also important to study the effects of correlated background noise such as officenoise.

One of the speech coding algorithm requirements is that the speech coder shall produce intelligible speech under conditions of acoustic background noise, 60 db referenced to 20 $\mu$ Newtons/meter$^2$. This statement, though technically precise, gives little feeling about the loudness of noise. Figure 7.2 [1] gives comparative intensities of variety of common sounds. The noise level described above is similar to quiet office noise. Regarding sound energy, Alexander Woods'[2] quotation gives the whole picture. He, in his book, "Physics of Music", points out that sound energy generated by shouting of the crowd throughout an exciting game (say, 50,000 people at a 90 minute football match between Notre Dame and USC is just about enough to warm one cup of coffee.

The effect of background noise, consisting of typewriter noise, conversation, music and so forth was studied on the real-time system. The algorithm performs with no difficulty.

The study of background noise is very simple after the algorithm has been implemented in real time. It is just a matter of talking into handset with noise in the background. The output could be heard through headphone. However, in the FORTRAN simulation, the task is not so straightforward. There is a need for digital speech file with background noise. It was thought that periodic background noise would be the worst kind of noise for PARC algorithm. Therefore, it was decided to study the performance of the algorithm for multispeaker files. Multispeaker files were created by adding two digital speech files with appropriate weight.
In the simulation, the multispeaker file was generated by adding sentence 11 (female speaker) to male speaker, sentence 7, as shown in Eq. (7.1).

\[ s_{\text{composite}} = s_{11} + k s_1 \]  

(7.1)

where \( k \) takes values from 0 to 1 thus having varying degree of background noise. It was noticed that pitch extraction loop picks pitch for both the speakers and algorithm performs very well as can be seen from the results in Table 7.3.

<table>
<thead>
<tr>
<th>Multispeaker file</th>
<th>SNR</th>
<th>Inloop SNR</th>
<th>SER</th>
<th>Entropy H Bits/sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_{11} + 0 S_1 )</td>
<td>21.11 db</td>
<td>13.01 db</td>
<td>8.10 db</td>
<td>1.39</td>
</tr>
<tr>
<td>( S_{11} + .25 S_1 )</td>
<td>20.16 db</td>
<td>13.03 db</td>
<td>7.13 db</td>
<td>1.57</td>
</tr>
<tr>
<td>( S_{11} + .5 S_1 )</td>
<td>19.44 db</td>
<td>13.39 db</td>
<td>6.05 db</td>
<td>1.66</td>
</tr>
<tr>
<td>( S_{11} + 1 S_1 )</td>
<td>18.92 db</td>
<td>13.95 db</td>
<td>4.97 db</td>
<td>1.76</td>
</tr>
</tbody>
</table>
Intensity (Watts per square meter)

Thunderclap
Rocket
Airplane
Machine
Street
Noisy Office
Automobile
Conversation
Home
Quiet Office
Whisper
Zero Loudness

Intensity (decibels)

Fig 7.2 Comparative intensities of a variety of common sounds from bottom to top in order of increasing sound pressure.
7.4 References


CHAPTER 3
TRANSMISSION ERRORS

8.1 Introduction

Many toll-quality speech links maintain bit-error rates (BER) which are too small (less than $10^{-5}$) to affect the quantizer and hence the coder performance. However, a BER of one tenth of a percent is not uncommon and for bad channels this rate could be as high as one percent. In such cases, SNR degradation may be severe unless special precautions are taken. It is important to determine the extent of SNR degradation and if possible how to minimize it.

The study outlined in this chapter is an attempt to answer the question posed above. Section 8.2 describes the method of introducing random transmission errors. This method was simulated on digital computer such that BER could be changed at run time. With the introduction of transmission errors, the effect of various parameters such as predictor order, coarseness of the quantizer and various decay constants on SNR was observed. Simulation results are discussed in Section 8.3.
8.2 Simulation of Transmission Errors:

It is assumed that transmission errors that occur in the digital channel as shown in Fig. 8.1 are random in nature. Hence in the simulation it is important to insure that errors do not occur in bursts. Similarly it is assumed that errors are made by bit reversal and not by bit addition or deletion. From the Fig. 8.1, it may appear that the effect of errors introduced in quantizer levels is similar to errors introduced in bit stream representing quantizer levels provided encoding and decoding operations are carried out correctly. However, it must be kept in mind that samples are not gained or lost if the errors are introduced in quantizer levels while they may be if errors are introduced in bit streams. The advantage of introducing errors in the quantizer levels is that the effect of transmission errors can be measured in terms of degradation of SNR so that the effect of various parameters on transmission errors can be easily evaluated. With errors in the bit stream, SNR looses its meaning since samples may no longer be synchronized. Of course, the transmission errors do affect the bit stream. The following procedure was adopted for simulation of transmission error.

1. Separate programs for PARC transmitter and receiver were written. Program asks for the bit error rate and transmitter produces a file of quantizer levels represented by integer numbers from 1 to 11. Receiver program reads quantizer levels from this file and produces file of reconstructed speech samples.

2. Randomized transmission errors were introduced by using RANDU function available in PDP 11/60 library. Care should be taken to make the seeds large enough for this function so that bursts of errors do not occur in the beginning.
Fig. 8.1 Speech Coder
3. Original speech file and reconstructed speech file is compared and SNR is calculated. This procedure is repeated for different values of parameters.
8.3 Minimizing Transmission Error Effect

Because of error in quantizer level \( q(k) \) reconstructed reduced speech \( \hat{v}(k) \) and hence reconstructed speech sample \( \hat{s}(k) \) is erroneous. This makes \( e(k) \) also erroneous which in turn causes \( a_1 \)'s (predictor parameter) to be incorrect. The effect of \( e(k) \) on \( a_1 \)'s can be observed from equation 2, which is repeated here for convenience.

\[
a_1(k+1) = a_1(k) + \frac{g \hat{v}(k-1) e(k)}{[(1-g) \sum \hat{v}(k-j) + \text{RMSMIN}])}
\]

Effect of erroneous \( \hat{e}(k) \) on updating \( a_1 \)'s can be minimized by increasing RMSMIN and decreasing \( g \). This can be seen from Table 8.1 and 8.2.

Table 8.1
Male speaker, sentence 1:
Bit Error Rate 1 in 100

<table>
<thead>
<tr>
<th>( g )</th>
<th>SNR without error</th>
<th>SNR with error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.015</td>
<td>19.25 db</td>
<td>5.83 db</td>
</tr>
<tr>
<td>0.01</td>
<td>19.07 db</td>
<td>6.49 db</td>
</tr>
</tbody>
</table>

Table 8.2
Male speaker, sentence 1:
Bit Error Rate 1 in 100

<table>
<thead>
<tr>
<th>RMSMIN</th>
<th>SNR without error</th>
<th>SNR with error</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>18.96 db</td>
<td>7.18 db</td>
</tr>
<tr>
<td>65</td>
<td>19.05 db</td>
<td>6.95 db</td>
</tr>
<tr>
<td>55</td>
<td>19.30 db</td>
<td>8.04 db</td>
</tr>
<tr>
<td>52</td>
<td>19.29 db</td>
<td>7.73 db</td>
</tr>
<tr>
<td>30</td>
<td>19.83 db</td>
<td>3.66 db</td>
</tr>
</tbody>
</table>
Predictor output is linear combination of past \( \hat{v} \)'s and is given by

\[
p(k) = \sum_{i=1}^{N} a_i(k) \hat{v}(k-i)
\]

where \( N \) is the order of predictor.

For high predictor order effect of transmission error is more since comparatively larger number of incorrect predictor coefficients and larger number of previous incorrect samples contribute to predictor output. Since \( a_i \)'s and \( \hat{v} \)'s both are used in constructing \( p(k) \), effect of transmission errors for increase in predictor order is rather serious. It was observed that by decreasing predictor order from 8 to 4 SNR (for BER of 1 in 100) improved by 3 db.

Various decay constants such as \( \alpha \), exponential decay for RMS value calculation and \( \delta \), decay constant for updating predictor parameters do have effect on performance of system with transmission errors. Choice of \( \alpha \) controls the effective interval that contributes to the Rms estimate. This interval is larger for syllabic system while it is smaller for instantaneous system. For both extremes, such as large \( \alpha \) (syllabic) and small \( \alpha \) (instantaneous), SNR decreased. (Ref. Table 8.3).

Decay constant \( \delta \) is used to update predictor parameters to prevent the transmission errors to propagate. Larger values of \( \delta \) improve the system performance with error, as can be seen from Table 8.3.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>SNR with no error</th>
<th>SNR with 1% error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha = 0.97 )</td>
<td>19.28 db</td>
<td>6.13 db</td>
</tr>
<tr>
<td>( = 0.9 )</td>
<td>18.96 db</td>
<td>7.18 db</td>
</tr>
<tr>
<td>( = 0.8 )</td>
<td>18.89 db</td>
<td>5.01 db</td>
</tr>
<tr>
<td>(*\delta = 0.02 )</td>
<td>19.46 db</td>
<td>overflow</td>
</tr>
<tr>
<td>( = 0.04 )</td>
<td>19.21 db</td>
<td>-0.78 db</td>
</tr>
<tr>
<td>( = 0.06 )</td>
<td>19.18 db</td>
<td>1.63 db</td>
</tr>
</tbody>
</table>

*With all other parameters optimized for good error performance, degradation is not so severe with above values of \( \delta \).
Quantizer noise is also an important factor in affecting the performance of the system with transmission errors. Closer the quantizer levels less is the quantization noise and hence less is the effect of transmission errors. Table 8.4 shows that by taking output levels apart SNR has decreased.

Table 8.4
Symmetric quantizer.
Bit error rate 1 in 100.

<table>
<thead>
<tr>
<th>Output levels</th>
<th>SNR without error</th>
<th>SNR with error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1.8 4.25 6.5 8.12</td>
<td>20.13 db</td>
<td>10.18 db</td>
</tr>
<tr>
<td>0 1.9 4.5 7.5 10.12</td>
<td>19.42 db</td>
<td>7.65 db</td>
</tr>
</tbody>
</table>

To see the effect of various error rates and the effect of errors in different segments of speech, random errors with 1% and 0.1% error rates were added using different random sequences. The study has shown that 0.1% error rate causes little degradation while it is significant for 1% error rate. However, output speech was found to be intelligible in spite of BER of 1%. It was also noticed that if the error occurs in silence segment of speech its effect is negligible. Considering the fact that 40 to 60% of the speech is silence, effect of small BER is not significant as can be seen from Table 8.5
Table 8.5

Case 1: SNR at transmitter = 20.13 db

<table>
<thead>
<tr>
<th>Random Sequence</th>
<th>Error Rate 1%</th>
<th>Error Rate 0.1%</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>7.13 db</td>
<td>15.14 (15 errors)</td>
</tr>
<tr>
<td>#2</td>
<td>8.57 db</td>
<td>20.00 (7 errors)</td>
</tr>
<tr>
<td>#3</td>
<td>9.78 db</td>
<td>14.34 (12 errors)</td>
</tr>
</tbody>
</table>

Case 2: SNR at transmitter = 18.63 db

<table>
<thead>
<tr>
<th>Random Sequence</th>
<th>Error Rate 1%</th>
<th>Error Rate 0.1%</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>6.99 db</td>
<td>16.41 db</td>
</tr>
<tr>
<td>#2</td>
<td>6.00 db</td>
<td>18.48 db</td>
</tr>
<tr>
<td>#3</td>
<td>8.53 db</td>
<td>13.72 db</td>
</tr>
</tbody>
</table>
8.4 Conclusion

PARC algorithm found to perform well in presence of random channel errors as high as one percent error rate. The degree of adaptation in the algorithm and error performance seems to be related. In general, higher adaptation of parameters and more complex system leads to poor performance in presence of channel errors. For example, reducing the predictor order from 8 to 4 improved the error performance significantly.

PARC employs DPCM quantizer, hence it is more tolerant to randomly occurring bit errors from perceptual point of view than systems which employ PCM quantizers (1),(2). This is because error spikes caused in the reconstruction of a PCM waveform (due to wrongly received bit) can have maximum amplitudes which are in the order of peak of input signal while corresponding spike magnitudes in DPCM decoding really related to the peak value of first difference in the input. The consequent greater magnitude of a typical PCM error spike makes it more annoying in spite of the fact that it does not propagate in time. The effect of channel error on the synchronization has not been investigated.
8.5 References


CHAPTER 9
FILTERING

9.1 Introduction

Subjective listening tests have indicated that the most objectionable aspect of the speech generated at the PARC receiver is its granular noise due to quantization errors. Since the spectrum of the quantization noise \( n_q(k) \) will, in general, be whiter than the speech signal \( s(k) \), it appears reasonable to develop a filter for removing part of the quantization noise from \( s(k) \) and hence improving the speech.

Section 9.3 describes the pre-emphasis of speech to improve the speech quality in terms of perception. Various choices of filters are outlined in Sec. 9.3 and the results are presented in Sec. 9.4.

In the study of pre-emphasis of speech it was observed that low pass filtering does have considerable effect on entropy. This effect is discussed in Sec. 9.5 and simulation results are presented in Sec. 9.6. Design of low-pass Butterworth filter is outlined in the Appendix.
9.2 Evaluation of Pre-emphasis

For auto-correlated signals, such as speech, predictive coding [1.2] is an efficient method of encoding the signal into digital form. In predictive coders, the quantization noise depends on prediction errors; hence, efficient prediction minimizes quantization error. However, in some segments of speech, such as unvoiced speech, the degree of correlation is small and prediction is therefore poor, resulting in more quantization noise. When the amplitude of this noise is comparable to the speech signal it mars the quality of the received speech. To reduce this problem, it appeared that some sort of pre-emphasis of speech would be helpful.

The basic concept of a pre-emphasis filter as shown in Fig. 9.1 is to spread energy in the input signal over the full bandwidth of the processor. Since most of the energy in speech is in the lower end of the spectrum, the filter is a high-pass filter; and hence, the de-emphasis filter is a low pass filter. As pre-emphasis filter is a high frequency filter, the unvoiced segment of speech gets emphasized. However, the filter design is such that overall energy gain for typical phonetically balanced sentence is approximately to unity.

From the figure, it is clear that pitch extraction is to be carried out on high pass filtered speech to get B's and T's. However, it was observed that it makes little difference if the B's and T's are obtained by pitch extraction on original speech.

It might be possible to manipulate the block diagram of PARC by moving filters inside to get the equivalent system. This is of no immediate interest and hence not covered here. However, PARC algorithm
could be modified to include adaptive noise spectral shaping as proposed by Makhoul & Berouti [3].
Fig. 9.1 Pre and De-emphasis Filters with PARC Algorithm
9.3 Filter Selection

As mentioned earlier, the pre-emphasis filter is a high-pass filter and de-emphasis is just the inverse of pre-emphasis filter.

In order to minimize complexity, the order of filter is kept small. It was decided to use a 3rd-order filter with general form

\[ s_f(k) = \frac{1}{K} s(k) + \frac{b}{K} s(k-1) + \frac{c}{K} s(k-2) \]  

(9.1)

The Z transform of this filter is

\[ H(z) = \frac{1 + az^{-1} + bz^{-2} + cz^{-3}}{K} \]  

(9.2)

Then, using the fact that Z is just \( e^{j\omega T} \) where \( T \) is the sampling interval the system function becomes

\[ H(j\omega) = \frac{1 + ae^{-j\omega T} + be^{-j2\omega T} + c e^{-j3\omega T}}{K} \]  

(9.3)

and the magnitude of the system function is

\[ |H(e^{j\omega t})|^2 = 1 + a^2 + b^2 + c^2 + [2a + 2b (a + c)] \cos \omega T \]
\[ + 2(b + ac) \cos 2\omega T + 2c \cos 3\omega T \]  

(9.4)

The four filters whose performance will be described in details in the next section are shown in Table 9.1.

<table>
<thead>
<tr>
<th>Filter Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>( K )</td>
</tr>
<tr>
<td>filter 1</td>
</tr>
<tr>
<td>filter 2</td>
</tr>
<tr>
<td>filter 3</td>
</tr>
<tr>
<td>filter 4</td>
</tr>
</tbody>
</table>
The frequency response of these filters is shown in Fig. 9.2 for a sampling rate of 6.4 kHz.

The de-emphasis filter is just the inverse of the pre-emphasis filter. Therefore, for filter of Eq. (9.3) the inverse is

\[ \hat{s}(k) = K s_f(k) - a\hat{s}(k-1) - b\hat{s}(k-2) = c\hat{s}(k-3) \]  (9.5)
9.4 Results

The four pre-emphasis filters shown in Fig. 9.2 were evaluated on a PARC operating in the 9.6 kbs mode. Sentence 1, "Cats and Dogs each hate the other" was used for the simulation. Following parameters were computed to evaluate the performance of the filters.

\[
\text{SNR(PARC)} = \frac{\sum s_f^2(k)}{\sum (s_f(k) - \hat{s}_f(k))^2} \quad (9.6)
\]

\[
\text{SNR(overall)} = \frac{\sum s^2(k)}{\sum (s(k) - \hat{s}(k))^2} \quad (9.7)
\]

\[
\text{SEGSNR} = \frac{\sum_{i=1}^{n} \text{SNR}_i}{n} \quad (9.8)
\]

where \(n\) - number of blocks of block length 120 samples in this case.

\[
\text{SER (Signal Energy Reduction)} = -10 \log \frac{\sum v_f^2(k)}{\sum s_f^2(k)} \quad (9.9)
\]

where \(v_f(k) = s_f(k) - \beta \hat{s}_f(k-T)\)

It was noticed that by pre-emphasizing speech, the output speech is perceptually better than without pre-emphasis. However, there is an increase in entropy value. This is due to the fact that the increased amplitude of high frequency speech generates more upper levels of quantizer thus generating more bits. All results are reported in Table 9.2.
Table 9.2
Signal to Noise Ratios with Various Pre-emphasis Filters

<table>
<thead>
<tr>
<th>Filter Gain</th>
<th>PARC</th>
<th>SER</th>
<th>Whole SNR</th>
<th>System SEGSNR</th>
<th>Entropy H bits/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>no filters</td>
<td>-</td>
<td>18.54 db</td>
<td>5.43 db</td>
<td>18.54 db</td>
<td>11.02 db</td>
</tr>
<tr>
<td>filter 1</td>
<td>1.003</td>
<td>15.63 db</td>
<td>2.71 db</td>
<td>16.73 db</td>
<td>9.68 db</td>
</tr>
<tr>
<td>filter 2</td>
<td>0.994</td>
<td>16.44 db</td>
<td>3.58 db</td>
<td>18.64 db</td>
<td>11.02 db</td>
</tr>
<tr>
<td>filter 3</td>
<td>0.999</td>
<td>17.41 db</td>
<td>4.44 db</td>
<td>19.36 db</td>
<td>11.65 db</td>
</tr>
<tr>
<td>filter 4</td>
<td>0.997</td>
<td>17.69 db</td>
<td>4.73 db</td>
<td>19.20 db</td>
<td>11.55 db</td>
</tr>
</tbody>
</table>
9.5 Low-pass Filtering vs. Entropy

For the 9.6 Kbs transmission rate and 6.4 KHz sampling frequency, the number of bits per sample is 1.5. Transmission of parameters such as \( \beta \) and \( T \) take a few bits per sample. Therefore, the entropy in the simulation must be maintained at the value less than 1.5. In the previous section, it was observed that pre-emphasis makes the output speech perceptually better; however, it also increases entropy which is unacceptable. To overcome this problem one could use coarser quantization to make the entropy small to begin with and then employ pre- and de-emphasis filters. Unfortunately, the improvement in speech quality due to pre-emphasis operation is not significant enough to consider the above approach.

Another method for achieving good speech quality while controlling the bit rate would be to select parameters such that the speech quality is excellent disregarding the increase in entropy and using the buffer control to check the bit rate. How this buffer control works is discussed in details in the next chapter. The use of a filtering operation to control the bit rate is discussed here.

If a low pass filter is used instead of high pass filter as a pre-emphasis filter, energy reduction is improved and as a result entropy drops. Therefore, the buffer will fill at a slower rate and buffer control would be used infrequently; consequently, there would be less degradation caused by the use of buffer control. However, low pass filtering with bandwidth less than 3200 Hz causes some loss of speech naturalness. It was noticed that during high energy, voiced segment, of speech that the bit rate is higher and hence the buffer fills faster. If the bit rate is brought down by employing low-pass filtering after buffer content is greater than a particular threshold, a double purpose is served. One, the buffer filling
operation is slowed down thus avoiding or delaying the drastic buffer control operation. Second, high frequency components in low energy unvoiced segments of speech are not filtered out since filter is in operation only after particular threshold thus preserving natural quality. This threshold was found by plotting the buffer content against time for a typical sentence and noting the value of buffer content for voiced speech. For the bit buffer in the FORTRAN simulation of the PARC algorithm, 500 bits appeared to be reasonable a threshold value.

As the cut-off frequency of low-pass filter is decreased, the performance of the predictor improves thus decreasing bit rate further. Thus, the filter cut-off frequency can be decreased depending how full the buffer gets. Hence the pre-emphasis filter is now an adaptive low-pass filter, adaptation of the cut-off frequency depending on the buffer contents.

The low-pass filter used is a simple 3rd order Butterworth filter. Its design and frequency plots are given in an appendix to this chapter.
9.6 Results and Conclusion

The FORTRAN simulation of PARC was modified to include the adaptive low-pass filter concept. To insure that different filters are not employed for every sample of speech when the buffer contents are close to the threshold, a hysterisis structure was utilized. Once a particular filter is selected, it employed in the algorithm for next block of 100 samples. The buffer content is compared with the thresholds only after the block of speech samples is processed. This is shown in the flow chart in Fig. 9.3. The simulation was carried out for Sentence 1: male speaker. The effects of low-pass filtering on entropy and signal energy reduction are tabulated in Table 9.5 while effects on buffer content and bit rates are reported in Table 9.6.

Table 9.5
Effects of Low-Pass Filtering

<table>
<thead>
<tr>
<th></th>
<th>SNR</th>
<th>SER</th>
<th>Entropy H</th>
</tr>
</thead>
<tbody>
<tr>
<td>no filter</td>
<td>14.85 db</td>
<td>4.1 db</td>
<td>1.37 bits/sample</td>
</tr>
<tr>
<td>LPF with cut-off 1400 Hz</td>
<td>16.43 db</td>
<td>5.1 db</td>
<td>1.95 bits/sample</td>
</tr>
</tbody>
</table>

Table 9.6
Effect of LPF on Bit Rate

<table>
<thead>
<tr>
<th></th>
<th>Sample Number</th>
<th>Change in Buffer Content</th>
<th>Increase Bit/Sample</th>
<th># Bit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>no filter</td>
<td>300 - 1000</td>
<td>800</td>
<td>1.14</td>
<td>2.49</td>
</tr>
<tr>
<td>LPF with Cut-off 1400 Hz</td>
<td>300 - 1000</td>
<td>680</td>
<td>0.97</td>
<td>2.32</td>
</tr>
</tbody>
</table>

*Bit rate is obtained by adding 1.35 bits/sample to the bit/sample increase in buffer. This is because on an average of 1.35 bits/samples are transmitted on digital channel.
i = 0

is it a new block?

yes

i = 100 (Block size)

Use the same filter as for the previous sample

Select the filter cut off frequency according to buffer content

i = i-1

Fig. 9.3 Hysterisis Structure of Adaptive Low Pass Filter
The results in Table 9.5 and 9.6 indicate that low-pass filtering can be used to control bit rate. In the real-time simulation on the MAP, the structure of programming was such that a varying number of speech samples would be processed each time to get exact predetermined number of bits. This could cause double or triple filtering of the same speech samples. This situation occurs in voiced segments of speech.

The all-pole 3rd order filter could not be used since double and triple filtering causes frequency response to have peaks at the 3 db frequency. This is undesirable and hence there is a need to find new methods of employing adaptive low filtering for buffer control.

It was observed that multiple filtering reduces the 3 db frequency. As mentioned earlier more bits are generated in voiced speech and filter cut-off frequency has to be reduced to cut down the bit rate. Since multiple filter case happens in voiced speech and since it is the region where low-pass cut-off frequency needs to be decreased, the design of single filter would be enough. Thus, there would be no need to change the filter as buffer gets closer to being full. This happens automatically by multiple filtering when buffer gets closer to being full. The filter which gives required change in 3 db frequency upon repetition was designed and design and frequency response is outlined in the appendix.
9.7 References


9.A Appendix: Derivation of Filters

9.A.1 Butterworth (Maximally flat) Filters

A Butterworth filter is designed to be maximally flat at the origin of the magnitude of the frequency response, i.e., the filter is forced to have as many zero derivatives at the origin of the magnitude response as possible.

The normalized squared magnitude response of the Butterworth filters is

\[ |H(\omega)|^2 = \frac{1}{1 + \omega_p^{2n}} \]  \tag{9.8}

where \( \omega_p = \omega/\omega_c \) is the normalized frequency for an nth order and \( \omega_c \) is the desired 3 dB cut-off frequency of the nth order filter.

In the design of the desired filter frequency response the poles of the transfer function \( H(s) \) are needed, then

\[
H(s)H(-s) = \frac{1}{1 + (-s^2)^n} \]

where

\[
\begin{align*}
&\frac{1}{1 + s^{2n}} & & \text{even} \\
&\frac{1}{1 - s^{2n}} & & \text{odd}
\end{align*}
\]  \tag{9.9}

Thus, the \( 2n \) roots of \( +1 \) are desired depending on the oddness or evenness of the order of the desired filter. Consider the third-order Butterworth filter; for \( n=3 \), the poles are

\[
\begin{align*}
s_0 &= \omega_c \angle 0^\circ \\
s + 1 &= \omega_c \angle 60^\circ \\
s + 2 &= \omega_c \angle 120^\circ \\
s_3 &= \omega_c \angle 180^\circ
\end{align*}
\]

These poles are plotted in Fig. 9.4.
Fig. 9.4 Pole Location of 3rd Order Butterworth Filter
Since the poles of the magnitude squared transfer function are symmetrically placed, the poles which fall in the left half plane are assigned to $H(s)$ for physical realizability.

Thus

$$H(s) = \frac{1}{\prod_{i=1}^{3} (s - s_i)}$$

(9.10)

where $s_i$ are the left half plane poles of the magnitude squared transfer function. In the third order case

$$s_1 = (\frac{1}{2} - j\frac{\sqrt{3}}{2})\omega_c$$

$$s_2 = (\frac{1}{2} + j\frac{\sqrt{3}}{2})\omega_c$$

$$s_3 = -\omega_c$$

The equivalent transfer function for the digital filter becomes

$$H(z) = \frac{Kz^3}{(z - p_1)(z - p_2)(z - p_3)}$$

(9.11)

where

$$p_1 = e^{s_1 T} = \exp \left[ (\frac{1}{2} - j\frac{\sqrt{3}}{2})\omega_c T \right]$$

$$p_2 = e^{s_2 T} = \exp \left[ (\frac{1}{2} + j\frac{\sqrt{3}}{2})\omega_c T \right]$$

$$p_3 = e^{s_3 T} = \exp \left[ -\omega_c T \right]$$
\( H(z) \) can also be written as

\[
H(z) = \frac{K}{1 + az^{-1} + bz^{-2} + cz^{-3}}
\]  

(9.12)

Comparing (9.11) and (9.12)

\[
a = -e^{\frac{-\pi f_c}{f_s}} \left( e^{\frac{-\pi f_c}{f_s}} \frac{\sqrt{3}\pi f_c}{f_s} \right)
\]

\[
b = -e^{\frac{-2\pi f_c}{f_s}} \left( 1 + 2 e^{\frac{-\pi f_c}{f_s}} \frac{\sqrt{3}\pi f_c}{f_s} \right)
\]

\[
c = -e^{\frac{-4\pi f_c}{f_s}}
\]

Where \( f_c \) - Cut off frequency in Hz.

\( f_s \) - Sampling frequency in Hz.

Frequency response of various Butterworth filters is plotted in Fig. 9.5.
9.A.2 Design of the low-pass filter used in the algorithm

As mentioned in the earlier sections of this chapter, low pass filtering was seen as a method of softly degrading the speech to avoid overflowing the speech buffer. This low pass filter had to be designed, however, to operate in the MAP.

The first design of the necessary digital filter was done using the impulse invariant transformation on a third order Butterworth filter. Unfortunately, the resulting digital filter had a relatively large amount of ripple which was not desirable, especially because multiple filtering was desired. To assure monotonicity, then, the digital filter was redesigned, using the conformal bilinear transform. Also, it appeared that for ease of implementation in the MAP that the digital filter has only one zero in the z-plane. Thus, the general form of the transfer function of the digital filter was

\[ H(z) = \frac{\omega_{CA} + \omega_{CA} z^{-1}}{(\omega_{CA} +1) + (\omega_{CA} -1) z^{-1}} \]  \hspace{1cm} (9.13)

where

\[ \omega_{CA} = \tan \left( \frac{\pi}{6400} f_C \right), \]

\[ f_C = \text{filter cutoff frequency (Hz)}. \]

The frequency response of such a filter with an 1800 Hz cutoff is shown in Fig. 9.6. The frequency response for double filtering is also shown, and it can be seen that that cutoff frequency is about 1350 Hz.
Figure 9.6 Frequency response of the filter and the "double filter".
10.1 Introduction

Even with the use of adaptive low pass filtering, as described in the previous chapter, to decrease bit-rate generation when the speech buffer approaches overflow, there still remains the ultimate problem of deciding what to do to prevent the buffer from overflowing. Similarly, there is also the problem of deciding what to do to prevent the buffer from underflowing. These two problems are considered in this chapter on buffer control.
10.2 Overflow Control

To prevent buffer overflow, a method had to be found to sharply limit the bit generation rate occasionally which did not cause an unreasonable amount of distortion. Because the quantizer employs feedback and is backward adaptive, it is possible to obtain buffer control by denying the use of certain quantizer levels (or by selectively permitting the use of additional quantizer levels), or by varying the decision thresholds for the quantizer levels. A number of simulation runs were made of both of these techniques without a great deal of success. It appeared that dropping or adding levels was too crude to be an effective control. It was found that either produced a very small change in bit rate generation, or a very pronounced change in bit rate generation. Varying the decision threshold did not appear to be very useful, either, because it typically caused too much distortion.

As a result of these investigations, the problem was approached again from a different angle. Analysis of simulation results showed that the speech buffer was most prone to overflow during instances of voiced speech. This suggested that pitched repetition might be a useful solution.

Pitched repetition relies on the large amount of correlations between pitch periods of voiced speech. In pitched repetition, samples are generated by duplicating the samples from one pitch period earlier. Due to the large amount of correlation, pitched repetition can typically be carried out for short periods of time during voiced speech, without greatly affecting the subjective quality of speech.

Details of the implementation of pitched repetition are given in Chapter 2.
10.3 Underflow Control

It is just as important to prevent the transmitter sample buffer from underflowing as it is to prevent it from overflowing. This is because when the transmitter sample buffer underflows, the receiver sample buffer overflows. Thus, some method was needed to prevent the transmitter sample buffer from underflowing.

A relatively easy solution to this problem was found by the use of "null" quantizer levels. This technique involves the transmission of a specified bit pattern, just like a normal quantizer level, except that it causes nothing to happen and is discarded at the receiver, with nothing being placed in the receiver sample buffer. Null quantizer levels are used as necessary, then, to prevent the transmitter sample buffer from underflowing.
10.4 Special Considerations at the Receiver

In the error-free condition, it is possible to control both the transmitter and receiver sample buffers by controlling just the transmitter sample buffer. In the presence of errors, however, this is no longer the case. For example, due to an error, it would be possible for the receiver sample buffer to underflow without the transmitter sample buffer overflowing. Some simple rules were developed to handle this situation and to resynchronize the buffers. If the receiver sample buffer overflows, the most recent sample is discarded, since it probably represents silence or near silence. This seems to cause the least distortion, and resynchronizes the buffers. If the receiver sample buffer underflows, a quantizer level "l" is inserted. This again causes a minimum of distortion and resynchronizes the buffers.
Conclusions and Suggestions for Further Research

It is necessary to provide buffer control to prevent potentially catastrophic conditions due to overflow or underflow. Some relatively simple, but effective, strategies for buffer control have been developed to this end.

There is, of course, room for improvement. For example, the null quantizer levels could also be used to force resynchronization of the transmitter and receiver sample buffers. Even more basic questions exist about the problem of buffer control itself, because it would seem that the quantization system is not as efficient as it could be if it regularly runs into overflow and underflow. A related question is why pitched repetition, which takes few bits to transmit, is so effective at a time when the quantizer is operating at a high bit generation rate.
CHAPTER 11
SOURCE AND ERROR CONTROL CODING

11.1. Introduction

Source and error control coding are the interface between the internal variables of the system and the communications channel. The noiseless source coder performs the first step in generating the bits to be transmitted. Its goal is to try to convey all the necessary information using a minimum of bits. The error control coding is then used to increase the probability that these bits will be received without error. Due to differences in quantity and importance, though, the quantizer levels and the side information are handled in different ways by the source and error control coders.
11.2. Source Coding

11.2.1. Quantizer Levels

In most common quantization systems, a very simple source coding procedure is used: typically there are $2^N$ quantizer levels used by the system, and each quantizer level is coded as a $N$-bit binary number. In contrast, PARC uses a more sophisticated source coding procedure in order to increase performance. The source code used to encode the 11 quantizer levels and the "null" level use a variable number of bits to represent the levels, with fewer bits being used for the more common levels. In this way, the quantizer level information can be conveyed very efficiently.

The first attempts at designing a simple variable length source code, however, proved frustrating. Simulations showed that a simple variable length source code tended to cause the sample buffer to fill rapidly during segments of voiced speech. Analysis of the situation showed that the problems appeared to stem from the fact that the quantizer levels were not stationary or independent. Instead, analysis showed that the quantizer levels were better represented by a model where the levels were generated by switching between two sources, one representing the quantizer behavior during voiced segments, and another representing the quantizer behavior during unvoiced and silent segments.

In order to take advantage of this phenomenon, then, a new variable length source code was developed. This new source code was what is known as an overfull source code. Overfull codes are characterized by an ability to encode some sequence in more than one way. For example, in the final source code, a sequence of 14 level-1's could be encoded in either of two ways. This
redundancy would appear, at first glance, to decrease the efficiency of the code, but it, in fact, increases the efficiency of the code. In particular, this redundancy is what allows the source code to perform well with a bimodal source. This was accomplished by designing most of the code using the high-entropy (voiced) statistics, and then adding the long-run codeword based on the low-entropy statistics. In this way, the overfull code performs better than a non-overfull code could.
11.2.2. **Side Information**

Besides encoding the quantizer levels, the source coder must also encode the side information used in the PARC system. This information consists primarily of the pitch extraction coefficient $\beta$ and the pitch period $T$. (Pitched repetition is also signaled by the use of a false $\beta$.) This encoding is performed for every frame of samples.

The encoding used is fairly straightforward. The pitch period can only be one of 64 possible integers between 20 and 83, so that it can be represented exactly by 6 bits. The encoding used for the pitch extraction coefficient $\beta$, however, is slightly more complicated. The first complication is that the value of $\beta$ must be quantized because it is a real number. It was determined by simulation, though, that the system was relatively insensitive to the quantization of $\beta$, and that using 97 quantization levels, evenly distributed between $-2$ and 2, appeared to have a negligible effect. The other complication was the signaling of pitched repetition through the use of a false $\beta$. The signaling itself could be handled easily by simply assigning it an unused $\beta$ value. It was felt, however, that it was important that this signal not be mistaken. The encoding used for $\beta$, then, used 7 bits, with the all zero pattern reserved for pitched repetition. Error suppression was then provided by not assigning any $\beta$ quantizer levels to the patterns containing one or two 1's. This left 99 patterns for $\beta$ quantizer levels, while protecting the pitched repetition signal from single bit errors.
11.3. Error Control Coding

In order to maintain system performance when using a communications channel with a relatively high bit error rate, error control coding is provided for each block of bits. There were several constraints, though, which dictated what kind of error control coding could be used. The blocks were required to be about 200 bits long by the pitch information. The block length also was constrained by synchronization requirements; it was also constrained by the lengths which simple coding schemes require. All of the constraints were satisfied by performing the coding over partial blocks, rather than an entire block at a time. The 189 bit block is divided into three 63 bit frames, so that a single-error-correcting (57, 63) Hamming code may be used. As a result, up to 3 bit errors per block can be corrected, greatly improving the performance of the system in a severe environment.
11.4. Conclusions and Suggestions for Further Research

The source and error control coding described in this chapter allows the system to perform efficiently and cope with severe communications environments. The schemes described were rather simple, as appeared to be required for this implementation. There are, of course, more sophisticated methods which could be studied which might improve performance even further. A major question is how to develop quantization schemes which allocate bits efficiently according to subjective criteria. Another major issue is how to design error protection systems which perform well over a large range of bit error rates.
APPENDIX A
FORTRAN SIMULATION OF ALGORITHM

This appendix presents a listing of the FORTRAN simulation of PARC algorithm. This simulation differs from the real-time algorithm in two ways. First, this algorithm operates on a block containing a fixed number of samples rather than a fixed number of bits. Second, the algorithm described here does not have the adaptive filtering of the input sample sequence.
PROGRAM REPEAT

PURPOSE: TO SIMULATE THE REPEAT ALGORITHM.

V.R DATE NAME COMMENTS
1.0 16-JUL-79 J.M. KRESSE EXISTING SOFTWARE
1.1 17-JUL-79 J.M. KRESSE BACKED UP
1.2 22-JUL-79 J.M. KRESSE OUTPUT BUFFER COUNTER
1.1# 31-OCT-79 J.M. KRESSE REMAINING SAMPLES BUG

THIS IS A SPEECH CODING ALGORITHM FOR DIGITAL TRANSMISSION OF SPEECH AT A RATE OF 96# BITS PER SECOND. THE ALGORITHM COMBINES PITCH EXTRACTION LOOP, PITCH COMPENSATING ADAPTIVE QUANTIZER, SEQUENTIALLY ADAPTIVE PREDICTOR, MULTIPATH TREE SEARCHING AND ADAPTIVE SOURCE CODING. ADAPTIVE SOURCE CODING PART OF THE ALGORITHM WILL BE ADDED TO THIS LATER.

SPEECH SIGNAL IS HIGHLY REDUNDANT. THIS REDUNDANCY IS REMOVED BY EXTRACTING PITCH BLOCK BY BLOCK AND PITCH REDUCED SIGNAL IS FORMED. A SEQUENTIALLY ADAPTIVE PREDICTOR USING BACKWARD ADAPTAION IS USED TO FORM AN ESTIMATE OF PITCH REDUCED SIGNAL. THE ERROR IN THIS ESTIMATE IS QUANTIZED BY PITCH COMPENSATING ADAPTIVE QUANTIZER. THE QUANTIZER HOWEVER USES THE MULTIPATH TREE SEARCHING ALGORITHM TO DETERMINE ITS OUTPUT. THE RESULTING QUANTIZED OUTPUT IS CODED USING AN ADAPTIVE SOURCE CODING PROCEDURE. SOURCE CODE ALSO PERMITS TRANSMISSION OF PITCH INFORMATION AND SYNCHRONIZING SIGNALS.

FOLLOWING IS THE LIST OF VARIABLES.
S - ARRAY OF ORIGINAL SPEECH SAMPLES.
V - ARRAY OF REDUCED SPEECH SAMPLES.
VHAT - ARRAY OF ESTIMATED PITCH REDUCED SPEECH.
SHAT - ARRAY OF RECONSTRUCTED SPEECH SAMPLES.
T - PITCH PERIOD.
BETA - CORRELATION COEFFICIENTS.
KBLK - BLOCK SIZE.
NBBLK - NUMBER OF BLOCKS.
L - DEPTH OF TREE IN TREE SEARCHING ALGORITHM.
```fortran
C
PARAMETER BUFFL=16,IBUF1=256,IBUF2=512,KBLK=15# 1-1.7

PARAMETER BUFFL=16,IBUF1=256,IBUF2=512,KBLK=15# 11.7
PARAMETER LUNBUF=7,LUNPRT=6,LUNSH=2,LUNTI=5,MXBLCK=255 11.2
PARAMETER NAMLEN=33,NO=.FALSE.,NUL=",YES=",TRUE. 1.1.2
PARAMETER LUNPRT=6,LUNSH=2,MXBLCK=255 11.2
C
BYTE NAME(NAMLEN),FNAME1(NAMLEN),FNAME2(NAMLEN),FNAME3(NAMLEN) 11.2
BYTE NAME(NAMLEN),H(4#),ICOMM(4#) 11.2
C
BYTE NAME(33),FNAME1(33),FNAME2(33),FNAME3(33),H(4#),ICOMM(4#) 1-1.2
BYTE IDATE(16),ITIME(9),QPLOT 11.2
C
INTEGER NCHAR,PREFIX 11.2
C
INTEGER QSY,TS(MXBLCK) 11.1
C
INTEGER QSY,TS(255) 11.1
C
LOGICAL BUFNL 11.2
C
REAL BUFCT 11.2
C
DIMENSION BETA(MXBLCK),SHAT(IBUF1),V(IBUF1),WATT(IBUF1) 11.1
DIMENSION BETA(255),SHAT(IBUF1),V(IBUF1),WATT(IBUF1) 1-1.1
C
COMMON S(IBUF2) 11.1
C
COMMON /ICT/ ICT 11.2
C
COMMON /MBUFF/ MBUFF(BUFFL) 11.2
C
COMMON /QPLOT/ QPLOT 11.2
C
MOD81(INDX1)=1AND(INDX1-1,IBUF1-1)+1 11.1
MOD82(INDX2)=1AND(INDX2-1,IBUF2-1)+1 11.1
C
C
ASK FOR THE FILENAMES, COMMENTS
C
WRITE(S,519) 11.1
C
FORMAT( ' ENTER THE INPUT FILENAME: ' ) WRITE(S,519)
C
READ(S,520)NAME 11.1
C
FORMAT(33A1) 11.1
C
WRITE(S,520)NAME 11.1
C
FORMAT( ' ENTER THE PARAMETER POINTER FILENAME: ' ) WRITE(S,520)
C
READ(S,549)NAME1 11.1
C
FORMAT(33A1) 11.1
C
WRITE(S,549)NAME1 11.1
C
FORMAT( ' ENTER THE OUTPUT FILENAME: ' ) WRITE(S,555)
C
READ(S,555)NAME2 11.1
C
FORMAT(33A1) 11.1
C
WRITE(S,555)NAME2 11.1
C
FORMAT( ' ENTER THE OUTPUT HEADER COMMENT: ' ) WRITE(S,559)
C
READ(S,559)ICOMM 11.1
C
FORMAT(48A1) 11.1
C
C
DO UNTIL Y OR N
C
C
CONTINUE
WRITE(S,591) 11.1
C
FORMAT( ' DO YOU WANT AN EFFECTIVE M PLOT? (Y OR N): ' ) WRITE(S,591)
C
READ(S,592)QPLOT 11.1
C
FORMAT(A1) 11.1
C
IF( NOT( (QPLOT.EQ.' Y').OR.(QPLOT.EQ.' N') ) ) GO TO 594
C
CONTINUE
C
IF( QPLOT.EQ. 'Y' ) GO TO 594
C
WRITE(S,593) 11.1
C
FORMAT( ' ENTER THE EFFECTIVE M FILENAME: ' ) WRITE(S,593)
C
READ(S,559)NAME3
```
I~
I.I!
ICI
0
z
N
i
I-z
V)
u
ui
o
!Z
w
U
I-L
0
0
Zy
oal
..hi
P
JK-W
Nw
:O-
Keg
z
.3
4 <
110x130
I2
o
iAn.hZf
216x130
a

I 15:47:32  15-APR-88  PAGE 3
FORTRAN IV-PLUS  V#2-51
REPEAT.FTM /TR:BLOCKS/WR
     594 CONTINUE
     595 WRITE(LUNIT,595)
     596 FORMAT(' IF YOU WANT A FILE OF BUFFER COUNTS, ENTER THE',
           ' FILENAME.'/
           ' OTHERWISE, HIT RETURN. ')
     597 READ(LUNIT,596)NCHAR,FNAME4
     598 FORMAT(9,9,NAMES)
     599 IF(NCHAR.EQ.0)GO TO 597
     600       BUFFIL=YES
     601       FNAME4(NAMLEN)=NUL
     602 OPEN(UNIT=LUNBUF,NAMES=FNAME4,TYPE='NEW',
           CARRIAGECONTROL='LIST')
     603       1
     604       GO TO 598
     605       CONTINUE
     606       BUFFIL=NO
     607       GO TO 598
     608       CONTINUE
     609       CONTINUE
     610       CONTINUE
     611       CONTINUE
     612       CONTINUE
     613       CONTINUE

C C INSERT END-OF-STRING MARK.
     614 FNAME(33)='B
     615 FNAME1(33)='B
     616 FNAME2(33)='B
     617 FNAME3(33)='B
OPEN THE INPUT FILE.

OPEN(UNIT=1, TYPE='OLD', READERLY, NAME=FNAME, SHARED)

READ THE HEADER ON SPEECH FILE.

READ(1,1#)NSENT, IRATE, NSAMP, IUPPR, ILOWR, NTERMS, H

FORMAT(615, 1#X, 4#A1)

PRINT THE HEADER.

CALL DATE(IDATE)

CALL TIME(ITIME)

WRITE(6,11) IDATE, ITIME

WRITE(6, 12)

WRITE(6,28) NSENT, IRATE, NSAMP, IUPPR, ILOWR, NTERMS, H

WRITE(6,78) XBLK, NSAMP

WRITE(6,78) FORMAT('FRAME SIZE=', 1#, ' TOTAL SAMPLES=', 1#)

WRITE(6,78)

FORMA(T 'ENTER THE VALUE OF THE MULTIPLYING FACTOR ALPHA:')

READ(5, 72) ALPHA

WRITE(G18.7)

WRITE(6,73) ALPHA

WRITE(5.71)

FORMAT('ALPHA=', G12.4)
INITIALIZE \$, BETA(1), T(1), IFRST, IBLK1, NREM, AND NREAD

DO 201 I=Buf1+1, IBUF2

CONTINUE

BETA(1)=B.
T(1)=1

IFIRST POINTS TO THE FIRST SAMPLE IN THE BLOCK.

IBLK1 IS THE BLOCK NUMBER.

IBLK1=1

NREM IS THE NUMBER OF SAMPLES REMAINING IN THE BUFFER.

NREM=#

NREAD IS THE NUMBER OF "COMPLETE BUFFER" READS.

NREAD=NSAMP/IBUF1

WRITE(6,74)

FORMAT('',''BLOCK'',5X,''BETA'',6X,''T'')
FORTRAN IV-PLUS V#2-61
15:47:32 15-APR-68
REPEAT.HTN /TR:BLOCKS/WR

##94 IF(NREAD.LE.9)GO TO 21
##95 DO 22 NREAD=1,NREAD
##96 IF((IREAD IS EVEN))GO TO 221
##97 READ(I,23)(S(1ITME2),ITME2=1,IBUF1)
##98 FORMAT(1615)
##99 DO WHILE(MODB2(IFRST+KBLK-1).LE.IBUF1)
##100 IF(MODB2(IFRST+KBLK-1).GT.IBUF1)
##101 GO TO 24
##102 CALL PITCH(BETA(IBLK1),T(IBLK1),MODB2(IFRST))
##103 WRITE(6,699)IBLK1,BETA(IBLK1),T(IBLK1)
##104 FORMAT(2X,13,2X,GIN,14)
##105 IFRST=MODB2(IFRST+KBLK)
##106 GO TO 25
##107 CONTINUE
##108 GO TO 26

GO TO 26

GO TO 25

GO TO 24

GO TO 23

GO TO 22

GO TO 21

GO TO 20

GO TO 19

GO TO 18

GO TO 17

GO TO 16

GO TO 15

GO TO 14

GO TO 13

GO TO 12

GO TO 11

GO TO 10

GO TO 9

GO TO 8

GO TO 7

GO TO 6

GO TO 5

GO TO 4

GO TO 3

GO TO 2

GO TO 1

GO TO 0
READ(I,23)(S(ITME3),ITME3=IBUF1+1,IBUF2)
DO WHILE(IFIRST+KBLK-1.LE.IBUF2)
IF(IFIRST+KBLK-1.GT.IBUF2)GO TO 270
CALL PITCH(BETA(IBLK1),T(IBLK1),IFIRST)
WRITE(6,6)(IBLK1,BETA(IBLK1),
T(IBLK1)
IBLK1=IBLK1+1
IFIRST=IFIRST+KBLK
GO TO 280
CONTINUE
GO TO 260
CONTINUE
CONTINUE
CONTINUE
CONTINUE
NREM=IBUF1-MOD81(IFIRST)+1
NREM=IAND(IBUF1-MOD81(IFIRST)+1,IBUF1-1)
GO TO 210
CONTINUE
NREM1=IAND(NSAMP,IBUF1-1)

IF(TOTAL SAMPLES REMAINING.LT.KBLK)GO TO 29#

IF(NREM1+NREM.LT.KBLK)GO TO 29#

IF((NREAD+1)IS EVEN) GO TO 33#

IF(IAND(NREAD,1).EQ.1)GO TO 33#

READ(1,23#)(S(ITME4),ITME4=NREM1)

DO WHILE(MODB2(IFRST+KBLK-1).LE.IBUF1)

IF(MODB2(IFRST+KBLK-1).GT.IBUF1)GO TO 31#

CALL PITCH(BETA(IBLKI),T(IBLKI),
MODB2(IFRST))

WRITE(6,6#)(IBLKI,BETA(IBLKI),
T(IBLKI)

IFRST=MODB2(IFRST+KBLK)

GO TO 32#

CONTINUE

GO TO 33#
READ(1,230)(S(I),I=1,IBU1)  
DO WHILE (IFRST+KBLK-1.LE.IBUF2)  
IF (IFRST+KBLK-1.GT.IBUF2) GO TO 340  
CALL PITCH(BETA(IBM1),T(IBM1),IFRST)  
WRITE(6,330)IBM1,BETA(IBM1),T(IBM1)  
IBM1=IBM1+1  
IFRST=IFRST+KBLK  
GO TO 350  
CONTINUE  
GO TO 330  
CONTINUE  
GO TO 290  
CONTINUE  
IF MODB1(IFRST)=1, THERE ARE NO LEFTOVER SAMPLES.  
IF MODB1(IFRST).EQ.1) GO TO 360  
SET BETA=-T=1, FOR LEFTOVER SAMPLES.  
BETA(IBM1)=0  
T(IBM1)=1  
WRITE(6,330)IBM1,BETA(IBM1),T(IBM1)  
GO TO 360  
CONTINUE
CLOSE THE SPEECH FILE

CLOSE(UNIT=1)

OPEN THE POINTER FILE FOR PARAMETERS.

OPEN(UNIT=3,NAME=FNAME1,TYPE='OLD',READONLY,SHARE)

READ(3,37)NRUNS

TYPE *= .', IF YOU WISH TO RUN AGAIN*, TYPE I'.

ACCEPT *, IRUN

IF(IRUN.NE.1)GOTO 110

DO 76 IRUNS=1,NRUNS

CALL SUBROUTINE INSTR WHICH RETURNS VALUE OF L.

CALL INSTR'T(L)

REOPEN THE SPEECH FILE AND SKIP OVER THE HEADER.

OPEN(UNIT=1,NAME=FNAME,TYPE='OLD',READONLY,SHARE)

READ(1,48)

FORMAT(8#X)

OPEN THE OUTPUT (SHAT) FILE AND WRITE THE HEADER

OPEN(UNIT=2,NAME=FNAME2,TYPE='NEW',CARRIAGECONTROL='LIST')

WRITE(2,481)NSENT,IRATE,NSAMP,IUPPR,ILOWR,NTERMS,ICOMM

FORMAT(65,1#X,4#A1)

IF(NPLOT.NE.'V')GO TO 487

OPEN(UNIT=4,NAME=FNAME3,TYPE='NEW',
CARRIAGECONTROL='LIST')

WRITE(4,481)NSENT,IRATE,NSAMP,IUPPR,ILOWR,NTERMS,ICOMM

ICNT=1

CONTINUE
initialize es, en, shat, vhat, s

169 es=e.
170 en=e.
171 do 41 itmi=1, ibuf1
172 shat(itmi)=e.
173 vhat(itmi)=e.
174 s(ibuf1+itmi)=e.
41 continue
176 write(6, 62)
177 62 format('asample', 5x, 'es', 10x, 'en')
178 bufcnt=e.
179 prefix=e.
160

FORTRAN IV-PLUS V#2-B1  15:47:32  15-APR-88  PAGE 12
REPEAT.FTN /TR:BLOCKS/WR

#169
#181
#182
   IF(NREAD2.LE.9)GO TO 428
   DO 43# IREAD2=1,NREAD2
      C
      IF(IREAD2 IS EVEN)GO TO 431
      IF(IAND(IREAD2,1).EQ.0)GO TO 431
      READ(1,44#)S(IITM2),ITM2=1,IBUF1
      FORMAT(16(S))
      DO 45# ITM3=1,IBUF1
      ITM4=(IREAD2-1)*IBUF1+ITM3
      CALL LPPROC(ALPHA,BETA,BUFCT,  11.2
      BUFIN,EN,ES,IBUF1,ITM3,ITM4,  11.2
      ITM5,ITM7,KBLK,L,LUMBUF,LUMPR,T,  11.2
      LUNSH,MXBLCK,PREFIX,SHAT,T,V,  11.2
      WHAT)
      CALL LPPROC(ALPHA,BETA,EN,ES,  1-1.2
      IBUF1,ITM3,ITM4,ITM5,KBLK,L,  1-1.2
      LUNPR,T,LUNSH,MXBLCK,SHAT,T,V,  1-1.2
      WHAT)
      ITM5=(ITM4-L+1  1-1.1
      ITM6=MODB1(ITM4)  1-1.1
      ITM7=MODB1(ITM5)  1-1.1
      IBLK2=(ITM4-1)/KBLK+1  1-1.1
      IBLK3=(ITM5-1)/KBLK+1  1-1.1
      CONSTRUCT THE PITCH-REDUCED
      SPEECH.
      V(ITM6)=FLOAT(S(IITM3))-ALPHA  1-1.1
      *BETA(IBLK2)=SHAT(MODB1(ITM3)
      -T(IBLK2))  1-1.1
      CALL INLOOP TO GET Q, WHAT  1-1.1
      CALL INLOOP(V(ITM6),Q,WHAT  1-1.1
      (ITM7))  1-1.1
RECONSTRUCT THE SPEECH.

IF(ITM5.LE.8)GO TO 468
   BTA=BETA(IBLK3)
   IT1=T(IBLK3)
   GO TO 478
   BTA=8.
   IT1=1
   GO TO 478
CONTINUE
   SHAT(ITM7)=WHAT(ITM7)+ALPHA *BTA*SHAT(MOD11(ITM7-IT1))
   SHAT(ITM7)=AMAX1(SHAT(ITM7), 2*SHAT(11))
   SHAT(ITM7)=AMIN1(SHAT(ITM7), 2*SHAT(11))
   IF((ITM7.NE.IBUF1).OR.(ITM5 .LE.8))GO TO 473
   WRITE(2,474)(NINT(SHAT((Y(I)),I=1,IBUF1)) FORMAT(16I5))
GO TO 473
CONTINUE
C
C
C
C
C
FS1=FLOAT(S(MODB2(ITM5)))
ES=ES+FS1**2
EN=EN+(FS1-SHAT(ITM7))**2
IF(MOD(ITM5,1000).NE.0)GO TO 63#
WRITE(6,61#)ITM5,ES,EN
C 61#
C 63#
GO TO 63#
CONTINUE
#189 45#
#19# GO TO 432
READ(1,448),(S(ITM2),ITM2=IBUF1+1,IBUF2)
DO 471 ITM3=IBUF1+1,IBUF2
CALL LPROC(ALPHA,BETA,BUFMT,)
BUFF,EN,ES,IBUF1,ITM3,ITM4,
ITM5,ITM7,KBLK,L,LUNBUF,LUNPRT,
LUNSH,MBLK,C,PREFIX,SHAT,T,V,

CALL LPROC(ALPHA,BETA,EN,ES,
IBUF1,ITM3,ITM4,ITM5,KBLK,L,
LUNPRT,LUNSH,MBLK,C,PREFIX,SHAT,T,V,

ITM4=(ITM3-L)-1
ITM5=MODB1(ITM4)
ITM7=MODB1(ITM5)
IBLK2=(ITM4-1)/KBLK+1
IBLK3=(ITM5-1)/KBLK+1
V(ITM5)=FLOAT(S(ITM3))-ALPHA
*SHAT(ITM7)+ALPHA
*SHAT(ITM7)+ALPHA
SHAT(ITM7)=SHAT(ITM7)

IF(ITM7.EQ.IBUF1)WRITE(2,474)

EN=EN+(FS1-SHAT(ITM7))**2
IF(MOD(ITM5,100)).EQ.0)WRITE(6,1)

CONTINUE
GO TO 420
CONTINUE
GO TO 420
CONTINUE
NREAD3 = IAND(NSAMP, IBUF1-1)
IF(NREAD3 .EQ. 0) GO TO 401

C
IF(NREAD2+1 IS EVEN) GO TO 472

C
IF(IAND(NREAD2, 1) .EQ. 1) GO TO 472

READ(1, 448) (S(ITM2), ITM2 = 1, NREAD3)
DO 488 ITM3 = 1, NREAD3
ITM4 = NREAD2 + IBUF1 + ITM3
CALL LPPROC(ALPHA, BETA, BUFCHT, BUFIN, EN, ES, IBUF1, ITM3, ITM4, ITHM, KBLK, L, LUNBUF, LUNPRT)
LOUH, MXBLCK, PREFIX, SHAT, TV, WHAT)

CALL LPPROC(ALPHA, BETA, EN, ES, IBUF1, ITM3, ITM4, ITHM, KBLK, L, LUNPRT, LUNSH, MXBLCK, SHAT, TV, WHAT)

LTHM = ITM4 - L+1
ITHM = MODB1(ITM4)
ITHM = MODB1(ITM5)
IBLK2 = ITM4 - I/KBLK + 1
IBLK3 = ITM5 - I/KBLK + 1
V(ITHM) = FLOAT(S(ITM3) - ALPHA*BETA(IBLK2)*SHAT

MODB1(ITM3 - T(IBLK2)))
CALL INLOOP(V(ITHM), Q, WHAT(ITHM))

IF(ITHM .EQ. 0) GO TO 498
BTA = BETA(IBLK3)
IT1 = T(IBLK3)
GO TO 588

488

588

CONTINUE
SHAT(ITHM) = SHAT(ITHM) + ALPHA*BTA*SHAT(MODB1(ITM7) - IT1)

SHAT(ITHM) = ANAD1(SHAT(ITHM), -2848.)
SHAT(ITHM) = ANAD1(SHAT(ITHM), 2847.)

GO TO 588

498

BTA = 0.
IT1 = 1
GO TO 588

588

CONTINUE
```
C IF((ITM7.EQ.1BUF1).AND.((ITM5.GT.3))WRITE(2,474) I-1.1
C (MINT(SHAT(I3)),I3=1,1BUF1) I-1.1
C FS1=FLOAT(S(MODB2(ITM5))) I-1.1
C ES=ES+FS1**2 I-1.1
C EN=EN+(FS1-SHAT(ITM7))**2 I-1.1
C IF(MOD(ITM5,18))EQ.0)WRITE(6,61#)ITM5,ES,EN I-1.1
#269 48# CONTINUE
#270 GO TO 491
#271 READ(1,44#)(S(ITM2),ITM2=IBUF1+1,IBUF1+NREAD3)
#272 DO 482 ITM3=IBUF1+1,IBUF1+NREAD3
#273 ITM4=(NREAD2-1)*IBUF1+ITM3
#274 CALL LPPROC(ALPHA,BETA,BUCNT,BUFFIL,EN,ES,
#275 1 IBUF1,ITM3,ITM4,ITM5,ITM7,KBLK,L,LUNBUF,LUNPRT,11.2
#276 2 LUNSH,MXBLCK,PREFIX,SHAT,T,V,VHAT) I-1.2
#277 CALL LPPROC(ALPHA,BETA,EN,ES,IBUF1,ITM3,ITM4,
#278 1 ITM5,KBLK,L,LUNPRT,LUNSH,MXBLCK,SHAT,T,V,VHAT) I-1.2
#279 ITM5=ITM4-L[
#280 ITM6=MODB1(ITM4)
#281 ITM7=MODB1(ITM5)
#282 IBLK2=(ITM4-1)/KBLK+1 I-1.1
#283 IBLK2=ITM5-I/KBLK+1 I-1.1
#284 V(ITM6)=FLOAT(S(ITM3)-ALPHA*BETA(IBLK2)
#285 1 *SHAT(MODB1(ITM3-TIBLK2))) I-1.1
#286 CALL INLOOP(V(ITM6),Q,VHAT(ITM7)) I-1.1
#287 SHAT(ITM7)=VHAT(ITM7)+ALPHA*BETA(IBLK3)
#288 1 *SHAT(MODB1(ITM7-TIBLK3))) I-1.1
#289 SHAT(ITM7)=AMAX1(SHAT(ITM7),-2#48.) I-1.1
#290 SHAT(ITM7)=AMIN1(SHAT(ITM7),2#47.) I-1.1
#291 IF(ITM7.EQ.1BUF1)WRITE(2,474)(MINT(SHAT(I4)),
#292 1 IX4=1,IBUF1) I-1.1
#293 FS1=FLOAT(S(MODB2(ITM5))) I-1.1
#294 ES=ES+FS1**2 I-1.1
#295 EN=EN+(FS1-SHAT(ITM7))**2 I-1.1
#296 IF(MOD(ITM5,18#)).EQ.0)WRITE(6,61#)ITM5,ES,EN I-1.1
#297 482 CONTINUE
```

CLOSE THE SPEECH FILE.

CLOSE(UNIT=1)

FINISH AND CLOSE THE OUTPUT FILE

IF(L.EQ.1) GO TO 484

DO 485 IX5=1,L-1

ITMB=MODB1(ITMB+IX5)

SHAT(ITMB)=#.

IF(ITMB.EQ.IBUF1)WRITE(2,474)(NINT(SHAT(IX6)),

IX6=1,IBUF1)

FORMAT(1615)

CONTINUE

GO TO 486

ITMB=ITM7

GO TO 486

CONTINUE

IF(ITMB.NE.IBUF1)WRITE(2,474)(NINT(SHAT(IX7)),IX7=1,ITMB)

CLOSE(UNIT=2)

IF(.NOT.(QPLOT.EQ.'Y'.AND.(ICNT.NE.4)))GO TO 488

WRITE(4,487)(MBUFF(IX8),IX8=1,ICNT)

CONTINUE

IF(QPLOT.NE.'Y')GO TO 489

CLOSE(UNIT=4)

CONTINUE

IF(BUFFIL.EQV.NO)GO TO 499

CLOSE(UNIT=LUMBUF)

CONTINUE

GO TO 499

CONTINUE

COMPUTE SIGNAL TO NOISE RATIO.

WRITE(6,610)NSAMP,ES,EN

FORMAT(17,2G12.4)

SNR=1#.*ALOG10#(ES/EN)

WRITE(6,98) SNR

FORMAT(8SNR OVERALL= ',G11.4,' DB')

WRITE(5,98)SNR

CALL IREN
SUBROUTINE PITCH(BETA,T,IJ)

PURPOSE: TO CALCULATE THE PITCH PERIOD T AND THE PREDICTION COEFFICIENT BETA.

V.R. DATE NAME COMMENTS
#1 16-JUL-79 J.M. KRESSE EXISTING SOFTWARE
#2 17-JUL-79 J.M. KRESSE BACKED UP

THIS IS A PITCH EXTRACTION PROGRAM.
BASIC PURPOSE OF PITCH EXTRACTION LOOP IN OUR ALGORITHM IS TO REMOVE PITCH REDUNDANCY OF SPEECH SIGNAL. HERE THE BLOCK ADAPTATION IS ASSUMED WITH BLOCK LENGTH K.
K VARIES BETWEEN 8# AND 2##.

PARAMETER 1BUF2=512, ITL=28, ITU=158, KBLK=158

PARAMETER 1BUF2=512, ITL=28, ITU=158, KBLK=188

INTEGER S,T

COMMON (IBUF2)

DIMENSION A(ITU), B(ITU)
MODB2(INDX1)=IAND(INDX1-1,IBUF2-1)+1

CALCULATE CORRELATION BETWEEN SAMPLES.

BIG=#.
IK=I3+KBLK-1
DO 6# IT=ITL,ITU
SUM1=#.
SUM2=#.
DO 5# J=1J,IK
M=S(MODB2(J))
MM=S(MODB2(J-IT))
F=FLOAT(M)
FF=FLOAT(MM)
SUM1=SUM1+F*FF
SUM2=SUM2+FF*FF
5# CONTINUE
IF(SUM2.LT..#)GOTO 51
B(IT)=SUM1/SUM2
A(IT)=B(IT)*SUM1
IF(BIG.LT.A(IT))BIG=A(IT)
GOTO 58
51 B(IT)=#.
A(IT)=#.
6# CONTINUE
SUBROUTINE INSTRT(I1)

PURPOSE: TO INITIALIZE THE INLOOP SYSTEM.

V.R.    DATE       NAME             COMMENTS
1. 16-JUL-79       J. M. KRESSE     EXISTING SOFTWARE
1. 17-JUL-79       J. M. KRESSE     BACKED UP
1.3A 19-SEP-79      J. M. KRESSE     ADD BUFFER CONTROL
1.4 26-SEP-79      J. M. KRESSE     ADD BUFFER OCCUPANCY

SUBROUTINE INSTRT IS USED TO READ IN PARAMETER VALUES AND
INITIALIZE QUANTITIES FOR SUBROUTINE INLOOP.

PARAMETER MXOM=11, MXXN=6, MXXM=1, MXXL=18, MXXLN=18
PARAMETER MINOS=FALSE, LNOIS=TRUE
PARAMETER NHRANGE=18
BYTE NAME(33), IDATE(18), ITIME(9)
INTEGER BUFACC
INTEGER OVLX, OMAX, ON, QQ, RANK
LOGICAL LEVEL
LOGICAL GAPED
REAL LOSNR, NOISE
REAL BFFCAR, RANGE, T
COMMON /ALPHA/ ALPHA(MXOM)
COMMON /AN/ AN(MXOM*MXXM, MXXN)
COMMON /AO/ AO(MXOM, MXMN)
COMMON /B/ B(MXXM)
COMMON /BFFCAR/ BFFCAR(NHRANGE)
COMMON /BUFACC/ BUFACC(NHRANGE)
COMMON /CN/ CN(MXOM*MXXM)
COMMON /EASQN/ EASQ(MXOM*MXXM)
COMMON /EASQ/ EASQ(MXOM)
COMMON /EASQ/ EASQ(MXOM)
COMMON /F/ F(MXOM)
COMMON /GAPED/ GAPED
COMMON /INBU/ INBU(MXLM)
COMMON /LEFDIS/ LEFDIS(MXLM)
COMMON /MADAP/ FACTOR, HISNR, LOSNR
COMMON /MEFDIS/ MEFDIS(MXLM)
COMMON /MVARY/ LEVEL, M1, M1LO, NHINO
COMMON /NNODE/ NNODE
COMMON /NNODE/ NNODEO
COMMON /PARAM/ M, L, N, OMAX, MXVHM, DELTA, ALPH, EASQ, EASQ, E
COMMON /OLVL/ OLVL(MXOM)
COMMON /ON/ ON(MXOM*MXXM, MXXN)
COMMON /OO/ OO(MXOM, MXXM, MXXN)
COMMON /RANGE/ RANGE(NHRANGE+1)
COMMON /RANK1/ RANK(MXOM*MXXM)
COMMON /SIGMN/ SIGMN(MXOM*MXXM)
COMMON /SIGMO/ SIGMO(MXOM)
COMMON /STATS/ NSAMP, VSRQ, NOISE
COMMON /T/ T(MXOM, 2)
COMMON /VHAT/ VHAT(MXOM*MXXM, MXXN)
COMMON /VHATO/ VHATO(MXOM, MXXN-1)
READ IN FILENAME OF FILE CONTAINING PARAMETERS, INITIALIZATION DATA.

TYPE = ', ' ENTER FILENAME OF PARAMETER FILE: '

ACCEPT 10,NLETTR,(FNAME(ILETTR),ILETTR=1,NLETTR)

READ(3,28)FNAME

FORMAT(33A1)

INSERT END-OF-STRING MARK.

FNAME(33)='

WRITE(6,30)FNAME

FORMAT('PARAMETER FILENAME: ',33A1)

OPEN (UNIT=8,NAME=FNAME,TYPE='OLD',READONLY)

READ IN SCALAR PARAMETERS

READ(8,48)L,M,N,QMAX,DELTA,ALPH,EASND,EASNMI,G

FORMAT(413,5G15.7)

MXVHN=MAX0(L-1,N)

WRITE(6,50)L,M,N,QMAX,MXVHN,DELTA,ALPH,EASND,EASNMI,G

FORMAT('SCALAR PARAMETERS: '/L=','I3,' M=','I3,' N=','I3,' 1' QMAX=','I3,' MXVHN=','I3/ DELTA=','G12.4,' ALPH=','G12.4,' 2' EASND=','G12.4,' EASNMI=','G12.4,' G=','G12.4)

LI=L
READ IN ARRAY PARAMETERS

READ(8,60) (F(IQ1),IQ1=1,QMAX), (ALPHA(IQ2),IQ2=1,QMAX),
   I(B,ICOEF), ICOEF=1,N

FORMAT(6/IQ1,F7.2/6/IQ2,F7.2/6/N>F7.2)
WRITE(6,70) (F(IQ3),IQ3=1,QMAX), (ALPHA(IQ4),IQ4=1,QMAX),
   I(B,ICOEF), ICOEF=1,N

FORMAT('ARRAY PARAMETERS: '/6/F10.4,6/X,6/QMAX/6/F7.2'/6/ALPHA=',
   6/QMAX/F7.2'/8='.4X,6/N>F7.2')

READ IN INITIAL VALUES FOR SIGMA, COEFFICIENTS

READ(8,80) SIGMA(1), (AO(1,ICOEF2), ICOEF2=1,N)
FORMAT(F7.2/6/N>F7.2)
WRITE(6,90) SIGMA(1), (AO(1,ICOEF3), ICOEF3=1,N)
FORMAT('INITIAL CONDITIONS: '/6/SIGMA=':.4X,F7.2'/6/PREDICTOR=',
   6/N>F7.2')

READ IN ADAPTIVE M PARAMETERS

READ(8,*) FACTOR, HSNR, LOSNR
WRITE(6,95) FACTOR, HSNR, LOSNR
FORMAT('ADAPTIVE M PARAMETERS: '/
   2 ' FACTOR=',G12.4,'; HSNR=',G12.4,'; LOSNR=',G12.4)
READ(8,*) ((T(I3,I4),I3=1,1,QMAX)
   11.3A
READ(8,*) RANGE
11.3A
READ(8,*) BFCAR
11.3A
WRITE(6,90) T:((T(I3,I4),I3=1,1,QMAX)
   11.3A
WRITE(6,90) RANGE: RANGE
11.3A
WRITE(6,90) BFCAR: BFCAR
11.3A
CLOSE FILE

CLOSE (UNIT=8)
COMPLETE INITIALIZATION OF THE TREE.

**76  NNODE=1**
**77  EAASO(1)=0.**
**78  DO 100 IBCK=1,MXVNM**
**79  VHAAT0(1,IBCK)=0.**
**80  100 CONTINUE**
**81  IF(L.LT.1)GO TO 100**
**82  DO 110 IBCK=1,L-1**
**83  QD(1,IBCK)=1**
**84  VN(IBCK)=0.**
**85  110 CONTINUE**
**86  120 CONTINUE**

INITIALIZE THE STATISTICAL COUNTERS.

**88  NSAMP=0**
**89  VSQR=0.**
**90  NOISE=0.**
**91  DO 130 IQ3=1,QMAX**
**92  QLVL(IQ3)=0**
**93  130 CONTINUE**
**94  LEVEL=MINOS**
**95  NLOD=0**
**96  NHNO=0**
**97  M1=0**
**98  DO 140 IX1=1,L**
**99  LEFDIS(IX1)=0**
**100  140 CONTINUE**
**101  DO 150 IX2=1,M**
**102  MEDIS(IX2)=0**
**103  150 CONTINUE**
**104  DO 160 IX5=1,NRANGE**
**105  BUFOCC(IX5)=0**
**106  160 CONTINUE**
**107  GAPPED=.FALSE.**
**108  RETURN**
**109  END**
SUBROUTINE LPPROC(ALPHA,BETA,BUFCT,_BUFIL,EN,ES,IBUFL,ITM3,ITM4,ITM5,ITM6,ITM7,IBLKL,LUNPR,LUNSH,MXBLCK,PREFIX)

2 SHAT,T,V,VMAT)

SUBROUTINE LPPROC(ALPHA,BETA,EN,ES,IBUFL,ITM3,ITM4,ITM5,ITM6,ITM7,IBLKL,1.LUNPR,LUNSH,MXBLCK,SHAT,T,V,VMAT)

1

POURSE: TO PERFORM THE LOOP PROCESSING.

<table>
<thead>
<tr>
<th>V.R</th>
<th>DATE</th>
<th>NAME</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>19-JUL-79</td>
<td>J. M. KRESSE</td>
<td>ORIGINATION</td>
</tr>
<tr>
<td>1.2</td>
<td>22-JUL-79</td>
<td>J. M. KRESSE</td>
<td>OUTPUT BUFFER COUNTER</td>
</tr>
<tr>
<td>1.3A</td>
<td>19-SEP-79</td>
<td>J. M. KRESSE</td>
<td>ADD BUFFER CONTROL</td>
</tr>
</tbody>
</table>

-----DECLARATIONS-----

PARAMETER IBUFL=512,MAXVAL=2047.,MINVAL=-2048.
PARAMETER BUFTHR=888,MXOM=11
INTEGER IBLKL,IBLKL,IBUFL,ITM3,ITM4,ITM5,ITM6,ITM7,ITM1,IXI,KBLKL
INTEGER LUNPR,LUNSH,MXBLCK,PREFIX,QS,TS(MXBLCK)
LOGICAL BUFIL
LOGICAL GAPPED
REAL ALPHA,BETA(MXBLCK),BTA,BUFCT,EN,ES,FS1,SHAT(IBUFL)
REAL VIUFL),VMAT(IBUFL)
REAL ALPHM(BETA(MXBLCK),BTA,EN,ES,FS1,SHAT(IBUFL),V(IBUFL)
REAL VMAT(IBUFL)
REAL BTASAV,THS,THRS,SHRS(MXOM,2)
COMMON SIUFL)
COMMON /GAPPED/ GAPPED
COMMON /T/ THS(MXOM,2)

-----DEFINITIONS-----

ITM5=ITM4-L+1
ITM6=MODB(ITEM4,IBUFL)
ITM7=MODB(ITEM5,IBUFL)
IBLKL=(((ITM4-1)/KBLKL)+1
IBLKL=(((ITM5-1)/KBLKL)+1

-----PROCEDURE-----

IF(NOT.(MOD(ITEM4,KBLKL).EQ.1).AND.(BUFCT.GE.BUFTHR))GO TO 7
GAPPED=YES
GAPPED=NO
BASAV=BETA(IBLKL)
BASAV=ALPHA(IBLKL)
DO 6 IX2=1,MXOM
DO 5 IX3=1,2
THRS(IK2,IX3)=THRS(IK2,IX3)
THRS(IK2,IX3)=1.E3#
CONTINUE
GO TO 7
CONTINUE
V(ITEM6)=FLOAT(S(ITEM3))-ALPHA*BETA(IBLKL)-

FORTRAN IV-PLUS
V82-51
LPPROC.FTN
/PR:BLOCS/WR

15:34:16  15-APR-8#  PAGE 2

1 SHAT(MODB((ITM3-T(IBM2)),IBUF1))

CALL INLOOP(V(ITM6),Q,VHAT(ITM7),BUFCT)

CALL INLOOP(V(ITM6),Q,VHAT(ITM7))

CALL OUTBUF(BUFCT,PREFIX,Q)

IF(ITMS.LE.9)GO TO 19
    BTA=BETA(IBM3)
    IT1=T(IBM3)
GO TO 29
19 CONTINUE

BTA=0.

GO TO 29

20 CONTINUE

SHAT(ITM7)=VHAT(ITM7)+ALPHA*BTA*SHAT(MODB((ITM7-IT1),IBUF1))

IF(.NOT.((MOD(ITM4,KBK).EQ.0).AND.((GAPPED.EQV.YES)))GO TO 27

GAPPED=NO

BETA(IBM2)=BTASAV

DO 26 IX2=1,MKOM

DO 26 IX3=1,2

IF(IX2,IX3)=THRSAVE(IX2,IX3)

CONTINUE

GO TO 27

CONTINUE

SHAT(ITM7)=AMAX1(MINVAL,(AMIN1(MAXVAL,SHAT(ITM7))))

IF(.NOT.((ITM7.EQ.IBUF1).AND.(ITMS.GT.9)))GO TO 48

WRITE(LUNSH,38)(MINT(SHAT(IX1),IX1=1,IBUF1)

FORMAT(16IS)

GO TO 48

CONTINUE

FS1=FLOAT(S(MODB(ITM5,IBUF2)))

IF(BUFIL.EQV.NO)GO TO 42

WRITE(LUNBUF,41)FS1,BUFCT

FORMAT(7.8,F10.1)

GO TO 42

CONTINUE

ES=ES+FS1**2

EN=EN+(FS1-SHAT(ITM7))**2

IF(MOD(ITMS,1000).NE.0)GO TO 68

WRITE(LUNPRT,58)(ITM5,ES,EN

FORMAT(17,2G12.4)

GO TO 68

CONTINUE

RETURN

END
INTEGER FUNCTION MODB(INT1,INT2)

PURPOSE: TO CALCULATE INT1 MOD INT2,
WHERE INT2=2**N, FOR SOME INTEGER N,
EXCEPT MODB(INT2,INT2)=INT2.

V.R  DATE  NAME  COMMENTS
1.1  19-JUL-79  J.M. KRESSE  ORIGINATION

------DECLARATIONS

C

INTEGER INT1,INT2

------PROCEDURE

C

MODB=IAND(INT1-1,INT2-1)+1
RETURN
END
SUBROUTINE INLOOP(V,Q,WHAT,BUF_CNT)
SUBROUTINE INLOOP(V,Q,WHAT)

PURPOSE: TO SIMULATE THE INLOOP SYSTEM.

V.R DATE NAME COMMENTS
#1 16-JUL-79 J.M. KRESSE EXISTING SOFTWARE
1.1 17-JUL-79 J.M. KRESSE BACKED UP
1.3A 19-SEP-79 J.M. KRESSE ADD BUFFER CONTROL

INLOOP TAKES THE REDUCED SPEECH, V, AT TIME N, AND RETURNS THE
QUANTIZER LEVEL, Q, AND THE ESTIMATE OF THE REDUCED SPEECH,
WHAT, FOR THE TIME N-L+1 (A DELAY OF L-1 UNITS).

PARAMETER MXL=18
INTEGER QMAX,Q
REAL BUF_CNT
COMMON /PARAM/M,L,N,QMAX
COMMON /INBUF/VN(MXL)
COMMON /NNODEO/NNODE
COMMON /NNDEN/NNDEN

SAVE V IN VN

IF(L.LE.1)GO TO 18
DO 28 IBCK=1,L-1
28 VN(L-IBCK+1)=VN(L-IBCK)
CONTINUE

NUMBER OF NEW NODES=NUMBER OF OLD NODES*NUMBER OF QUANTIZER
LEVELS

NNDEN=NNODEO*QMAX
GROW NEW NODES
CALL GROW4(BUF_CNT)
CALL GROW3
RANK NEW NODES
CALL RANKS3
SUBROUTINE GROW4(BUFCNT)

SUBROUTINE GROW3

PURPOSE: TO GROW THE NEW NODES.

V.R  DATE     NAME  COMMENTS
#1  16-JUL-79  J.M. KRESSE  EXISTING SOFTWARE
1.0  17-JUL-79  J.M. KRESSE  BACKED UP
1.3A  19-SEP-79  J.M. KRESSE  ADD BUFFER CONTROL

SUBROUTINE GROW GROWS QMAX NEW NODES FROM EACH OLD NODE.

PARAMETER MXQM=11,MXM=9,MXH=1,MXH=1,MXN=1

INTEGER QMAX,QN,QO

REAL BUFCNT,BUFA,ENORM,T

COMMON /PARAM/M.L.N,QMAX,MXVH,DELTA,ALPHA,EASNM,EASNM,G

COMMON /F(F(MXQM),F/B(MXN)/ALPHA/ALPHA(MXQM)

COMMON /NUDEO/NODEO/WHAT/WHATQ(MXM,MXN-1)/QQ/QQQ(MXM,MXN-1)

COMMON /EASO/EASO(MXM)

COMMON /AO/AO(MXM,MXH)/SIGMO/SIGMO(MXM)

COMMON /NODE/NOH/(CN-CN(MXQM=MXM)/WHATN/WHATN(MXQM=MXM,MXN)

COMMON /QQ/QQ(MQM=MXM,MXN)

COMMON /EASN/EASNW(MXM=MXM)/AN/AN(MXM=MXM)

COMMON /SIGM/SIGM(MXM=MXM)

COMMON /INBUF/INBUF(MXM)

COMMON /T/(MXQM,2)

WRITE(6,100)

FORMAT('NEW NODES:','X','Q1 2','X','WHAT1','X','WHAT2','X','CN',
18X,'EASN','X','SIGM','X','AO1','X','AO2','X','SIGMN','X','AN1',
29X,'AN2'/100)

DO FOR EACH OLD NODE

DO 10 INO=1,NODEO

GROW QMAX NEW NODES

DO 20 IQ=1,QMAX

COMPUTE NEW NODE INDEX

INN=(INO-1)*QMAX+IQ

COMPUTE THE ESTIMATE OF THE ERROR

EAT=F(IQ)+SIGMO(IQ)
COMPUTE ESTIMATE OF REDUCED SPEECH

```
VHATN(INN,1)=EHAT
DO 38 ICOEF=1,N
   VHATN(INN,1)=VHATN(INN,1)+AD(INO,ICOEF)*
   VHATO(INO,ICOEF)
CONTINUE
```

UPDATE Q

```
QN(INN,1)=IQ
```

BRING FORWARD OTHER VHAT, Q FROM OLD

```
NODE
```

```
DO 88 ITBK2=1,MXVHN
   VHATN(INN,ITBK2+1)=VHATO(INO,ITBK2)
CONTINUE
```

```
IF(L.LE.1)GO TO 78
   DO 48 ITBK=1,L-1
   QN(INN,ITBK+1)=QO(INO,ITBK)
   CONTINUE
```

```
PRINT 98,INN,VHATN(INN,1),VHATN(INN,L),
   QN(INN,1),QN(INN,L)
FORMAT(17,2G15.7,2I7)
```

COMPUTE CRITERION

```
CALL BUFCON(BUCNT,BUFFAC)
BUFFAC=1.
ENORM=(VN(1)-VHATN(INN,1)+EHAT)/(SIGMO(INO)*
   BUFFAC)
   IF(.NOT.(TIQ,1).LT.ENORM).AND.((ENORM.LE.
   T(IQ,2))))GO TO 75
   CN(INN)=8.
   GO TO 76
CONTINUE
```

```
   CN(INN)=-1.
   GO TO 76
CONTINUE
```

```
   CN(INN)=8.
   DO 58 ITBK1=1,L
   CN(INN)*CN(INN)-(VN(ITBK1)-VHATN(INN,
   ITBK1))**2
   CONTINUE
```
COMPUTE EXPONENTIAL AVERAGE OF ABSOLUTE VALUE OF VHAT

\[ EAASN(INN) = (1.0 - ALPHA) * ABS(VHAT(INN, 1)) * ALPHA \]
\[ EAASO(INO) \]

UPDATE PREDICTOR COEFFICIENTS

\[ DO 68 ICOF1 = 1, N \]
\[ CORRFA = (G * VHATN(INN, ICOF1 + 1) * EHAT) / \]
\[ (EAASN(INN) + EAASN)**2 \]
\[ AN(INN, ICOF1) = DELTA * (ICOF1 + (1.0 - DELTA) * \]
\[ (AO(INO, ICOF1) + CORRFA) \]

CONTINUE

UPDATE SIGMA

\[ SIGMN(INN) = MAX1(ALPHA(IQ) * SIGMO(INO), \]
\[ (EAASN(INN) + EAASN)**2 / EAASN) \]
\[ WRITE(6, 118) INN, QN(INN, 1), QN(INN, 2), \]
\[ VHATN(INN, 1), VHAT(INN, 2), CN(INN), EAASN(INN), \]
\[ SIGMO(INO), AO(INO, 1), AO(INO, 2), SIGMN(INN), \]
\[ AN(INN, 1), AN(INN, 2) \]
\[ FORMAT(13, 21Z, 16G12.4) \]

CONTINUE

CONTINUE

RETURN

END
DESIGN AND IMPLEMENTATION OF A SPEECH CODING ALGORITHM AT 9600 ETC(U)
APR 80 J L MELSA, D L Cohn, A ARORA
DCA100-79-C-0005

UNCLASSIFIED
SUBROUTINE BUFCON(BUFCNT,BUFFAC)

PURPOSE: TO COMPUTE THE BUFFER CONTROL FACTOR.

V.R DATE NAME COMMENTS
1.3A 19-SEP-79 J.M. KRESSE ADD BUFFER CONTROL
1.4 26-SEP-79 J.M. KRESSE ADD BUFFER OCCUPANCY

---DECLARATIONS---

PARAMETER NRANGE=1#
INTEGER BUFOCC
INTEGER IX
REAL BUFCAR,BUFCNT,BUFFAC,RANGE
COMMON /RANGE/ RANGE(NRANGE+1)
COMMON /BUFCAR/ BUFCAR(NRANGE)
COMMON /BUFOCC/ BUFOCC(NRANGE)

---PROCEDURE---

DO 1# IX=1,NRANGE
   IF(.NOT.((RANGE(IX).LE.BUFCNT).AND.(BUFCNT.LT.
      (RANGE(IX+1)))))) GO TO 5
   BUFCAR=BUFCAR(IX)
   BUFOCC(IX)=BUFOCC(IX)+1
   GO TO 6
   CONTINUE
   BUFCAR=BUFCAR(IX)
C   CONTINUE
   RETURN
1# CONTINUE
RETURN
END
SUBROUTINE RANKS3

PURPOSE: TO RANK THE NEW NODES.

V.R DATE NAME COMMENTS
V.1 16-JUL-79 J.M. KRESSE EXISTING SOFTWARE
V.2 17-JUL-79 J.M. KRESSE BACKED UP

SUBROUTINE RANKS RANKS THE NEW NODES IN DESCENDING ORDER OF THE CRITERION. THE ARRAY RANK STORES NODE INDICES: E.G., RANK(I)= NODE INDEX OF NEW NODE WITH HIGHEST VALUE OF CRITERION.

PARAMETER MXQM=11, MXM=1

INTEGER RANK, TEMP

COMMON /NNODEN/NNODEN/CH/CH(MXQM*MXM)
COMMON /RANK1/RANK(MXQM*MXM)

INITIALIZE RANKINGS

DO 1# MINDX=1, NNODEN

INITIAL RANK IS NODE INDEX

RANK(MINDX)=MINDX

1# CONTINUE
PERFORM BUBBLE SORT

DO (NNODEN-1) PASSES

DO 2# NPASS=1,NNODEN-1

DO (NNODEN-NPASS) COMPARISONS ON THE PASS

DO 3# NCOMP=1,NNODEN-NPASS

THE COMPARISON:

IF(CM(RANK(NCOMP)).GE.CM(RANK(NCOMP+1))) GO TO 40

ORDER REVERSED: REVERSE THEM

TEMP=RANK(NCOMP)
RANK(NCOMP)=RANK(NCOMP+1)
RANK(NCOMP+1)=TEMP

CONTINUE

WRITE(6,50) (RANK(NINDX1),NINDX1=1,NNODEN)

FORMAT(/"THE RANKING:/<(NNODEN)13//")

RETURN
END
SUBROUTINE PRUNE3(Q,WHAT)

PURPOSE: TO PRUNE THE NEW NODES.

V.R DATE NAME COMMENTS
#1 16-JUL-79 J.M. KRESSE EXISTING SOFTWARE
#2 17-JUL-79 J.M. KRESSE BACKED UP

SUBROUTINE PRUNE ANNOUNCES THE QUANTIZER LEVEL (AND THE ASSOCIATED WHAT), COLLECTS STATISTICS, AND PRUNES THE TREE IN PREPARATION FOR THE NEXT SPEECH SAMPLE.

PARAMETER MXL=15, MXHN=15, MXXM=1, MXN=8, MXQM=11
PARAMETER MNOIS=.FALSE.
PARAMETER LNOIS=.TRUE.
BYTE OPLT
INTEGER Q,QLVL,QMAX,QN,RANK
LOGICAL LEVEL
REAL LOSNR,NOISE
COMMON /AO/ AQ(MXXM,MXX)
COMMON /CN/ CN(MXQM*MXK)
COMMON /INBUF/ VN(MXK)
COMMON /MADAPT/ FACTOR,HISNR,LOSNR
COMMON /MEFDIS/ MEFDIS(MXXM)
COMMON /MVARY/ LEVEL,N1,NONO,NHINO
COMMON /MNODEN/ MNODEN
COMMON /MNODEO/ MNODEO
COMMON /PARAM/ M,L,N,QMAX,MXVHN
COMMON /QLVL/ QLVL(MXQM)
COMMON /QN/ QN(MXQM*MXK,MXL)
COMMON /QPLT/ QPLT
COMMON /RANK1/ RANK(MXQM*MXK)
COMMON /SIGMO/ SIGMO(MXK)
COMMON /STATS/ NSAMP,VSQR,NOISE
COMMON /VHATN/ VHATN(MXQM*MXK,MXLN)

ANNOUNCE DECISIONS
Q=QN(RANK1,L)
WHAT=VHATN(RANK1,L)

COLLECT SOME STATISTICS

INCREMENT SAMPLE COUNTER
NSAMP=NSAMP+1
IF(MOD(NSAMP,100).EQ.0) WRITE(6,18)(SIGMO(I),(AO(I,IX1),IX1=1,N))
1-CM(RANK1),MEFDIS

COLLECT SNR DATA
VSQR=VSQR+VN(L)**2
NOISE=NOISE+(VN(L)-WHAT)**2

COLLECT QUANTIZER LEVEL STATISTICS
COLLECT EFFECTIVE L STATISTICS

CALL LEFSI

COLLECT EFFECTIVE M STATISTICS

NDIS(NNODEO)=NDIS(NNODEO)+1

IF(QPLOT.EQ.'Y') CALL MFILE

PRUNE THE TREE BY NOT SAVING INFORMATION FROM THE PRUNED NODES.

IF(LEVEL.EQ.HINOIS) GO TO 2

NLOMO=NLOMO+1

GO TO 3

NHINO=NHINO+1

GO TO 3

CONTINUE

CALL SNRVAL(-CN(RANK(I))),SNR)

IF(.NOT.((LEVEL.EQ.LNOIS).AND.(SNR.LT.LOSNR))) GO TO 4

LEVEL=HINOIS

M=1

GO TO 4

CONTINUE

IF(.NOT.((LEVEL.EQ.HINOIS).AND.(SNR.GT.HSNR))) GO TO 5

LEVEL=LNOIS

M=1

GO TO 5

CONTINUE

NNODEO=

DO INN=1,NNODEN

IF(.NOT.

1 ((CN(RANK(INN),L).EQ.QN(RANK(I),L))

2 .AND.(NNODEO.LT.M)

3 .AND.((-CN(RANK(INN))).LE.FACTOR*(-CN(RANK(I))))))

4 GO TO 6

NNODEO=NNODEO+1

CALL SAVE(INN)

CONTINUE

CONTINUE

RETURN

END
SUBROUTINE LEFSST

PURPOSE: TO COLLECT THE EFFECTIVE L STATISTICS.

V.R DATE NAME COMMENTS
#1 16-JUL-79 J.M. KRESSE EXISTING SOFTWARE
1.0 17-JUL-79 J.M. KRESSE BACKED UP

LEFSST COLLECTS EFFECTIVE L STATISTICS

PARAMETER MXL=1#, MXM=1, MXQM=11
PARAMETER DIFF=.FALSE., SAME=.TRUE.
INTEGER QN
LOGICAL COND
COMMON /LEFDIS/ LEFDIS(MXL)
COMMON /MNODEN/ MNODEN
COMMON /PARAM/ M,L
COMMON /QN/ QN(MXQM=MXM,MXL)

COND=SAME
LEVEL=L

DO UNTIL (DIFFERENCE FOUND)

CONTINUE

NODE=2

DO UNTIL ((OUT OF NODES) OR (DIFFERENCE FOUND))

CONTINUE

IF(QM(NODE,LEVEL).NE.QM(L,LEVEL))COND=DIF

IF((NODE.EQ.MNODEN).OR.(COND.EQ.DIFF))GO TO 3#

NODE=NODE+1

GO TO 2#

CONTINUE

IF((COND.EQ.DIFF))GO TO 4#

LEVEL=LEVEL-1

GO TO 1#

CONTINUE

LEFDIS(LEVEL)=LEFDIS(LEVEL)+1

RETURN

END
SUBROUTINE MFILE

PURPOSE: TO COLLECT THE EFFECTIVE M STATISTICS AND PUT THEM OUT
ON LOGICAL UNIT 4.

V.R DATE NAME COMMENTS
1 16-JUL-79 J.M KRESSE EXISTING SOFTWARE
1.0 17-JUL-79 J.M KRESSE BACKED UP

PARAMETER BUFL=16
COMMON /ICNT/ ICNT
COMMON /MBUFF/ MBUFF(BUFL)
COMMON /NNODEO/ NNODEO

ICNT=ICNT+1
MBUFF(ICNT)=NNODEO
IF(ICNT.GE.BUFL)GO TO 20
ICNT=0
WRITE(4,10)MBUFF
FORMAT(<BUFFL>15)
CONTINUE
RETURN
END
C SUBROUTINE SNRVAL( NOISE, SNR )

C PURPOSE: TO CALCULATE THE SNR CORRESPONDING TO NOISE.

C V.R DATE NAME COMMENTS
0.1 16-JUL-79 J.M. KRESSE EXISTING SOFTWARE
1.0 17-JUL-79 J.M. KRESSE BACKED UP

C--------DECLARATIONS

C PARAMETER MNL=1.0
C PARAMETER MAXVAL=1.E3,MINVAL=-1.E3
C REAL NOISE
C COMMON /INBUF/ VN(MNL)
C COMMON /PARAM/ M,N

C--------PROCEDURE

C IF( NOISE .GE. 0 ) GO TO 10
C SNR=MAXVAL
C GO TO 50
C 10 CONTINUE
C SIG=0.
C DO 20 IX=1,N
C SIG=SIG+VN(IX)**2
C 20 CONTINUE
C IF( SIG .GE. 0 ) GO TO 30
C SNR=MINVAL
C 30 GO TO 40
C SNR=1.0.*LOG10(SIG/NOISE)
C SNR=AMAX1(SNR,MINVAL)
C SNR=AMIN1(SNR,MAXVAL)
C GO TO 40
C CONTINUE
C GO TO 50
C CONTINUE
C RETURN
C END
SUBROUTINE OUTBUF(BUF,NTQ,REFIX,0)

PURPOSE: TO COMPUTE THE SIZE OF THE BUFFER AFTER EACH SAMPLE.

V.R. DATE NAME COMMENTS
1.2 22-JUL-79 J.M. KRESSE ORIGINATION
1.3A 19-SEP-79 J.M. KRESSE ADD BUFFER CONTROL
1.4 09-OCT-79 J.M. KRESSE NEW SOURCE CODE

--- DECLARATIONS ---

PARAMETER RUNLEN=14, RNCOLN=4.
PARAMETER RUNLEN=18, RNCOLN=7.
PARAMETER R2NLEN=14, R2COLN=4.
PARAMETER QMAX=11, XMTCNT=1.35
PARAMETER QMAX=11, XMTCNT=1.6
INTEGER PREFIX, Q
REAL BITCNT(QMAX), BUF
REAL BITCNT(QMAX, 8:1), BUF

--- DEFINITIONS ---

DATA BITCNT /0, . . . , 4, . . . , 6, . . . , 7, . . . , 7, . . . , 7 /
DATA BITCNT /5, . . . , 3, . . . , 6, . . . , 8, . . . , 10, . . . , 11, . . . , 10 /
1 .5, 2, 5, 3, 5, 6, 5, 6, 5, 9, 5, 7, 5, 10, 5, 9, 5, 10, 5, 10, 5, 10 /

--- PROCEDURE ---

IF(Q.EQ.1)GO TO 10
BUFFNT=BUFQNT+PREFIX+BITCNT(Q)
R2CNT=PREFIX/R2NLEN
BUFFNT=BUFQNT+FLOAT(R2NT)*R2COLN+FLOAT(PREFIX-
R2NT*R2NLEN)+BITCNT(Q)
PREFIX=0
GO TO 36

10 CONTINUE
PUFFIX=PREFIX+1
IF(PREFIX.NE.RUNLEN)GO TO 26
BUFFNT=BUFQNT+RNCOLN
PREFIX=0
GO TO 26

26 CONTINUE
GOTO 36

--- END ---
## SUBROUTINE IEND

**PURPOSE:** TO COLLECT AND OUTPUT STATISTICAL DATA.

<table>
<thead>
<tr>
<th>V.R</th>
<th>DATE</th>
<th>NAME</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>16-JUL-79</td>
<td>J.M. KRESSE</td>
<td>EXISTING SOFTWARE</td>
</tr>
<tr>
<td>1.0</td>
<td>17-JUL-79</td>
<td>J.M. KRESSE</td>
<td>BACKED UP</td>
</tr>
<tr>
<td>1.4</td>
<td>26-SEP-79</td>
<td>J.M. KRESSE</td>
<td>ADD BUFFER OCCUPANCY</td>
</tr>
</tbody>
</table>

### SUBROUTINE IEND GATHERS AND OUTPUTS STATISTICAL DATA

- `PARAMETER MXL=10, MXM=1, MXOM=11`
- `PARAMETER NRACT=10`
- `INTEGER BUFOC, QMAX`
- `INTEGER QLV, QMAX`
- `LOGICAL LEVEL`
- `REAL NOISE`
- `COMMON BUFOC/ BUFOCC(NRACT)`
- `COMMON LEFDIS/ LEFDIS(MXL)`
- `COMMON MEFDIS/ MEFDIS(MXM)`
- `COMMON HVAR/ LEVEL, M1, NLO, NHINO`
- `COMMON PARAM/ M, L, N, QMAX`
- `COMMON QLV/ QLV(MXOM)`
- `COMMON STAT/ NSAMP, VSGR, NOISE`

**C**

- OUTPUT THE NUMBER OF SAMPLES HANDLED BY THE TREE SEARCHING ALGORITHM.

**C**

- `WRITE(6, 1) NSAMP`

**C**

- `WRITE(6, 2) OF Samples (Tree Searching) = ', 15)`

**C**

- `WRITE(6, 2) NLO, NHINO`

**C**

- `WRITE(6, 2) OF Samples (M=1) = ', 15, ' OF Samples (M=M) = ', 15)`

**C**

- OUTPUT SNR FOR TREE SEARCHING

**C**

- `WRITE(6, 3)(, '10, 'ALOGIC(VSGR, NOISE)`

**C**

- `FORMAT(16, 4, 'DB')`
OUTPUT QUANTIZER LEVEL FREQUENCY DISTRIBUTION AND ENTROPY

ALOG2=ALOG(2.)
ENTRY=0.
WRITE(6,4)
FORMAT( 'QUANTIZER LEVEL FREQUENCY DISTRIBUTION: ', 'LEVEL NUMBER', 'FREQ', 'P', '7X', 'P=LOG(P)' )
DO 99 IQ=1,OMAX
FREQ=FLOAT(QLVL(IQ))/FLOAT(NSAMP)
IF(FREQ.EQ.0.)GO TO 99
ANLN=-(FREQ*ALOG(FREQ))/ALOG2
GO TO 60
ANLN=0.
GO TO 60
ENTRY=ENTRY+ANLN
WRITE(6,7)IQ,QLVL(IQ),FREQ,ANLN
FORMAT(14,4X,15,2G12.4)
CONTINUE
WRITE(6,9)ENTRY
FORMAT( 'CORRESPONDING ENTROPY: ',G12.4,' BITS/SAMPLE ' )
WRITE(6,10)
FORMAT( 'EFF', '3X', 'F8.2', '16' )
WRITE(6,11)IQ,FLEFDIS(IQ),FLEFDIS(IQ)*IQ
1 LEMFDIS(IQ),IQ=1,L
FORMAT(15,F8.2,16)
WRITE(6,12)
FORMAT( 'MEFDIS', '3X', 'F8.2', '16' )
WRITE(6,13)IX2,MEFDIS(IX2),MEFDIS(IX2)*IX2
1 MMEFDIS(IX2),IX2=1,M1
FORMAT(15,F8.2,16)
WRITE(6,=)'BUFFER OCCUPANCY: ',BUFCC
RETURN
END
SUBROUTINE SAVE(NRANK)

PURPOSE: TO SAVE THE INFORMATION FROM THE NODE RANKED NRANK.

<table>
<thead>
<tr>
<th>V.R</th>
<th>DATE</th>
<th>NAME</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>16-JUL-79</td>
<td>J.M. KRESSE</td>
<td>EXISTING SOFTWARE</td>
</tr>
<tr>
<td>1.0</td>
<td>17-JUL-79</td>
<td>J.M. KRESSE</td>
<td>BACKED UP</td>
</tr>
</tbody>
</table>

SAVE SAVES THE INFORMATION FROM THE NODE RANKED NRANK.

PARAMETER MXL=1#, MXLN=1#, MXM=1, MXM=8, MXQ=11
INTEGER QMAX, QN, QQ, RANK
COMMON /AN/ AN(MXQ*MXM, MXN)
COMMON /AO/ AO(MXM, MXM)
COMMON /EAASN/ EAASN(MXQ*MXM)
COMMON /EAASO/ EAASO(MXM)
COMMON /NODEO/ NNODEO
COMMON /PARAM/ M,L,M,OMAX, MXVHN
COMMON /QN/ QN(MXQ*MXM, MXLN)
COMMON /QQ/ QQ(MXM, MXL-1)
COMMON /RANK/ RANK(MXQ*MXM)
COMMON /SIGMN/ SIGMN(MXQ*MXM)
COMMON /SIGMO/ SIGMO(MXM)
COMMON /VHATN/ VHATN(MXQ*MXM, MXLN)
COMMON /VHATO/ VHATO(MXM, MXLN-1)

DO 1# ICOEF=1, M
     AO(NNODEO, ICOEF) = AN(RANK(NRANK), ICOEF)
1# CONTINUE
SIGMN(NNODEO) = SIGMN(RANK(NRANK))
EAASN(NNODEO) = EAASN(RANK(NRANK))
DO 2# IBCK=1, MXVHN
VHATN(NNODEO, IBCK) = VHATN(RANK(NRANK), IBCK)
2# CONTINUE
IF(L.LE.1) GO TO 4#
DO 3# IBCK=1, L-1
3# CONTINUE
GO(NNODEO, IBCK) = QN(RANK(NRANK), IBCK)
3# CONTINUE
RETURN
END
APPENDIX B
SEGMENTED SNR PLOTS

B.1. Introduction

A number of similar programs were developed to aid in the analysis of the performance of PARC. The purpose of the programs were to generate plots, called SNRSIG, which indicate graphically the short-time performance of the system versus the short-time signal level. This information proved useful, as it appears that the average performance of the systems over short periods of time is more indicative than the overall average performance.

There are three groups of programs, with two programs in each group. The first program in each group performs the short-time analysis, and generates the data to be plotted. The second program then takes the data and generates the actual plot.

The first group, DBCALC and SNRSIG, deals with the short-time signal-to-noise ratio. This group is useful in analyzing the performance of the quantizer, especially problems like slope overload noise and granular noise. The second group, BFCALC and BFPLT, deals with the average buffer length, and the third group, BFDIFF and BFDFPL, deals with the difference in the average buffer length. These programs are useful in analyzing the performance of the source coding and of the buffer control.
PROGRAM DBCALC

PURPOSE: TO CALCULATE THE LOCAL SIGNAL STRENGTH AND THE LOCAL SNR FOR A SPEECH FILE AND AN SHAT FILE.

V.R DATE NAME COMMENTS
1 29-JUN-79 J.M. KRESSE ORIGINATION
2 12-JUL-79 J.M. KRESSE TO ADD THE SNR HEADER

--DECLARATIONS--

PARAMETER DIMENS=33, MAXBUF=16, MAXLEN=1000, MAXVAL=1.E38
PARAMETER MINVAL=1.E38, NO=.FALSE., YES=.TRUE.
BYTE DBFILE(DIMENS), SFILE(DIMENS), SHFILE(DIMENS)
INTEGER BLKLEN, BPOINT, BUFLEN, LUNDB, LUNS, LUNSCR, LUNSH
INTEGER S(MAXLEN), SBUF(MAXBUF), SHT(MAXLEN), SHBUF(MAXBUF), SPOINT
LOGICAL EOF, MORES
REAL OVRSNR, SIGSTR, SNR, TOTSIG, TOTNOI

---INITIALIZATION---

BPOINT=MAXBUF
BUFLEN=MAXBUF
EOF=NO
MORES=YES
TOTNOI=NO
TOTSIG=NO.

---PROCEDURE---

ASK FOR FILENAMES

CALL ASK(SFILE, SHFILE, DBFILE, DIMENS)
OPEN AND READY THE FILES

CALL OPENER(SFILE, SHFILE, DBFILE, DIMENS, LUNDB, LUNS, LUNSCR)
ASK FOR THE DESIRED CALCULATION BLOCK LENGTH.

CALL ASK2(MAXLEN, BLKLEN)
DO UNTIL (MORES.EQ.NO)

IF
     CONTINUE
       GET S, SHAT

CALL GETSEM(S, SHAT, SBUF, SHBUF, LUNSH, LUNS, BLKLEN, SPOINT, MAXBUF, BUFLEN, BPOINT, EOF, MORES)
IF (SPOINT.NE.1)
IF(SPOINT.EQ.1) GO TO 29

CALCULATE THE LOCAL SIGNAL STRENGTH AND SNR.
CALL DBC(S,SHAT,SPINT,SIGSTR,SNR,TOTSIG,TOTNOI)

PUT OUT THE SIGNAL STRENGTH AND SNR TO THE SCRATCHFILE.

CALL PUTSEM(SIGSTR,SNR,LUNSCR)

GO TO 20

IF(MORES.EQ.YES) GO TO 10
CONTINUE

CALCULATE THE OVERALL SNR.

OVRSNR=AMIN1(MAXVAL,AMAX1(MINVAL,10.*ALOG10(TOTSIG/TOTNOI)))

WRITE THE OVERALL SNR HEADER ON THE DBFILE.

WRITE(LUNDB,3#)OVRSNR

COPY THE DATA FROM THE SCRATCHFILE TO THE DBFILE.

REWINL LUNSCR

READ(LUNSCR,*),END=6#SIGSTR,SNR

WRITE(LUNDB,5#)SIGSTR,SNR

FORMAT(5G12.4)

GO TO 40

CLOSE THE FILES.

CALL CLOSER(LUNS,LUNSH,LUNDB,LUNSCR)

STOP

END
SUBROUTINE OPENER(SFILE, SHFILE, DBFILE, DIMENS,
1 LUNS, LUNSH, LUNDB, LUNSCR)

C
PURPOSE: TO OPEN AND READY THE FILES REQUESTED.
C
V.R DATE NAME COMMENTS
0.1 26-JUN-79 J.M. KRESSE ORIGINATION
0.2 12-JUL-79 J.M. KRESSE TO ADD THE SNR HEADER
C
DECLAREATIONS
C
INTEGER DIMENS
BYTE DBFILE(DIMENS), SFILE(DIMENS), SHFILE(DIMENS)

INTEGER LUNDB, LUNS, LUNSCR, LUNSH
C
INITIALIZATIONS
C
LUNS = 1
LUNSH = 2
LUNDB = 3
LUNSCR = 4
C
PROCEDURE
C
OPEN (UNIT=LUNS, NAME=SFILE, TYPE='OLD', READONLY, SHARED)
OPEN (UNIT=LUNSH, NAME=SHFILE, TYPE='OLD', READONLY, SHARED)
OPEN (UNIT=LUNDB, NAME=DBFILE, TYPE='NEW'; CARRIAGECONTROL='LIST')
OPEN (UNIT=LUNSCR, TYPE='SCRATCH', CARRIAGECONTROL='LIST')
READ (LUNS, 10)
READ (LUNSH, 10)
FORMAT (8X)
RETURN
END
SUBROUTINE ASK2(MAXLEN,BLKLEN)

PURPOSE: TO ASK FOR THE CALCULATION BLOCK LENGTH.

V.R NAME DATE COMMENTS
#1 J.M. KRESSE 26-JUN-79 ORIGINATION

PARAMETER MINLEN=1,RATE=6.4
INTEGER BLKLEN,MAXLEN
REAL BLKTIM,MAXTIM,MINTIM

FUNCTIONS

TIME(INT)=FLOAT(INT)/RATE

PROCEDURE

MINTIM=TIME(MINLEN)
MAXTIM=TIME(MAXLEN)

DO UNTIL A VALID BLOCK LENGTH IS INPUT

CONTINUE

WRITE(5,28)MINLEN,MAXLEN,MINTIM,MAXTIM
FORMAT( 'THE CALCULATION BLOCK LENGTH MUST BE BETWEEN ',
1 '15, AND ',15,' SAMPLES,'/'' INCLUSIVE (BETWEEN ',G11.4,
2 ' AND ',G11.4,' MS)'/
3 ' HOW MANY SAMPLES PER BLOCK DO YOU WANT?')

READ(5,'*)BLKLEN

IF((BLKLEN.LT.MINLEN).OR.(BLKLEN.GT.MAXLEN))GO TO 19

CONTINUE

BLKTIM=TIME(BLKLEN)

WRITE(5,38)BLKTIM

FORMAT( 'THE BLOCK IS ',G11.4,' MS LONG.')

RETURN

END
SUBROUTINE GETSEM(SHAT, SBUF, SBUF, LUNS, LUNSH, BLKLEN, SPOINT, 
MAXBUF, BUFLEN, BPOINT, EOF, MORES)

PURPOSE: TO GET A BLOCK OF S, SHAT SAMPLES

V.R. NAME DATE COMMENTS
#1 J.M. KRESSE 26-JUN-79 ORIGINATION

-----DECLARATIONS-----

PARAMETER DIGITS=5, NO=.FALSE., YES=.TRUE.
INTEGER BLKLEN, BPOINT, BUFLEN, COUNT, IX, MAXBUF, S(BLKLEN)
INTEGER SBUF(MAXBUF), SHAT(BLKLEN), SHBUF(MAXBUF), SPOINT
LOGICAL EOF, MORES

-----INITIALIZATION-----

SPOINT=0

-----PROCEDURE-----

DO WHILE ((SPOINT.LT.BLKLEN).AND.(MORES.EQ.YES))

1 IF (.NOT.((SPOINT.LT.BLKLEN).AND.(MORES.EQ.YES))) GO TO 110

DO WHILE ((BPOINT.LT.BUFLEN).AND.(SPOINT.LT.BLKLEN))

2 IF (.NOT.((BPOINT.LT.BUFLEN).AND.(SPOINT.LT.BLKLEN))) GO TO 30

BPOINT=BPOINT+1
SPOINT=SPOINT+1
(S(PPOINT)+SBUF(BPOINT)
SHAT(SPOINT)=SHBUF(BPOINT)

GO TO 20

CASE: ((BPOINT.EQ.BUFLEN).AND.(EOF.EQ.NO))

1 IF (.NOT.((BPOINT.EQ.BUFLEN).AND.(EOF.EQ.NO))) GO TO 80

READ(LUNS,5&B,END=48)COUNT,
(SBUF(IX),IX=1,(COUNT/DIGITS))
READ(LUNS,5&B,END=68)COUNT,
(SBUF(IX),IX=1,(COUNT/DIGITS))
FORMAT(9,BUFLEN),(DIGITS)
IF(COUNT.EQ.9)GO TO 60

EOF=YES
CONTINUE

EOF=YES
BUFLEN=(COUNT/DIGITS)
GO TO 70
CONTINUE

BPOINT=#

GO TO 100

CASE: ((BPOINT.EQ.BUFLEN).AND.(EOF.EQ.YES))

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE

CONTINUE
SUBROUTINE DBC(S,SHAT,SPROI,T,SIGSTR,SNR,TOTSIG,TOTNOI)

PURPOSE: TO CALCULATE THE SIGNAL STRENGTH AND SNR FOR A BLOCK OF SAMPLES.

V.R DATE NAME COMMENTS
#1 27-JUN-79 J.M. KRESSE ORIGINATION
#2 12-JUL-79 J.M. KRESSE TO ADD THE SNR HEADER

-----DECLARATIONS-----

INTEGER SPOI
INTEGER IX,SHAT(POINT),SHAT(SPOI)
REAL NOISE,SIGSTR,SNR,TOTNOI,TOTSIG

-----FUNCTIONS-----

CLIP(VAL)=AMNI1(AMAX1(VAL,MINVAL),MAXVAL)
GETCLIP(ASIG)=CLIP(10.**ALOG10(ASIG))

-----INITIALIZATION-----

NOISE=0.
SIGSTR=0.

-----PROCEDURE-----

DO 1 IX=1,SPROI
   SIGSTR=SIGSTR*FLOAT(S(I))-**2
   NOISE=NOISE*FLOAT(S(I))-SHAT(I)**2
  CONTINUE

TOSIG=TOSIG+SIGSTR
TOTNOI=TOTNOI+NOISE

CASE:((SIGSTR.EQ.0.).AND.(NOISE.EQ.0.).)

IF(.NOT.((SIGSTR.EQ.0.).AND.(NOISE.EQ.0.).))GO TO 2

SIGSTR=MINVAL
SNR=MAXVAL
GO TO 5

CASE:((SIGSTR.NE.0.).AND.(NOISE.EQ.0.).)

IF(.NOT.((SIGSTR.NE.0.).AND.(NOISE.EQ.0.).))GO TO 3

SIGSTR=GETCLIP(SIGSTR/((FLOAT(SPOI)*REF))
SNR=MAXVAL
GO TO 5

CASE:((SIGSTR.EQ.0.).AND.(NOISE.NE.0.).)

IF(.NOT.((SIGSTR.EQ.0.).AND.(NOISE.NE.0.).))GO TO 4

SIGSTR=MINVAL
SNR=MINVAL
SUBROUTINE PUTESEM(SIGSTR,SNR,LUNSCR)

PURPOSE: TO WRITE THE SIGNAL STRENGTH AND SNR.

V.R DATE NAME COMMENTS
0:1 27-JUN-79 J.M. KRESSE ORIGINATION
0:2 12-JUL-79 J.M. KRESSE TO ADD THE SNR HEADER

-----DECLARATIONS

INTEGER LUNSCR
REAL SIGSTR,SNR

-----PROCEDURE

WRITE(LUNSCR,10)SIGSTR,SNR
10 FORMAT(2G12.4)

RETURN
END
SUBROUTINE CLOSER(LUNS,LUNSH,LUNDB,LUNSCR)

PURPOSE: TO CLOSE THE FILES

V.R DATE NAME COMMENTS
1 27-JUN-79 J.M. KRESSE ORIGINATION
2 12-JUL-79 J.M. KRESSE TO ADD THE SNR HEADER

DECLARATIONS

INTEGER LUNDB,LUNS,LUNSCR,LUNSH

PROCEDURE

CLOSE(UNIT=LUNS)
CLOSE(UNIT=LUNSH)
CLOSE(UNIT=LUNDB)
CLOSE(UNIT=LUNSCR)
RETURN
END
PROGRAM SNRSIG

PURPOSE: TO PLOT LOCAL SNR VS. LOCAL SIGNAL STRENGTH

V.R NAME DATE COMMENTS
0.1 J.M. KRESSE 05-JUL-79 ORIGINATION

C----DECLARATIONS

PARAMETER ENDPLT=999.LUNDAT=7,NEWORG=3,PXMAX=8.5,PXWIND=7.5
PARAMETER PYMAX=10.5,PYWIND=9.5
INTEGER COUNT
REAL PXVO, PYVO, RAWX, RAWXM, RAWYM
REAL RAWY, RAWYMN, RAWYMX, SNR, SNRSEG, SUM
REAL VX, VXWIND, VY, VYWIND

C-----DEFINITIONS

VXWIND=PYWIND
VYWIND=PXWIND

C------INITIALIZATION

CALL PLOTS(0,0,0)
COUNT=0
SUM=0.

C-----PROCEDURE

OPEN THE DATAFILE ON LOGICAL UNIT LUNDAT, AND READ IN THE SNR.

CALL OPEN1(LUNDAT)
READ(LUNDAT,*),SNR

ASK FOR THE PLOTTING RANGES.

CALL RANGE1(RAWXM, RAWXM, RAWYM, RAWYM)

MOVE THE ORIGIN TO THE VIRTUAL ORIGIN.

PXVO=PXMAX-(PXMAX-PXWIND)/2.
PYVO=(PYMAX-PYWIND)/2.
CALL PLOT(PXVO, PYVO, NEWORG)

DRAW THE VIRTUAL AXES.

CALL AXES1(RAWXM, RAWXM, RAWYM, RAWYM, VXWIND, VYWIND)

DO UNTIL EOF

CONTINUE

READ(LUNDAT,*,END=28) RAWX, RAWY
CALL RAWTOV(RAWX, RAWXM, RAWYM, VXWIND, VX)
CALL RAWTOV(RAWY, RAWYM, RAWYM, VYWIND, VY)
COUNT=COUNT+1
SUM=SUM+VY
CALL POINT1(VX, VY)
GO TO 16
CONTINUE

SUM = SUM/FLOAT(COUNT)
SNRSEG = SUM*(RAWMX-RAWMN)/VWIND*RAWMN

LABEL THE PLOT.
CALL LABEL1(RAWXMN, RAWMX, VXWIND, VWIND, SNR, SNRSEG)
END THE PLOTTING AND CLOSE THE DATAFILE.

CALL PLOT(#, #, ENDPLT)
CLOSE(UNIT=LUNDAT)
STOP
END
SUBROUTINE OPENI(LUANDAT)

PURPOSE: TO OPEN THE DATA FILE ON LOGICAL UNIT LUANDAT.

V.R. NAME DATE COMMENTS
J.M. KRESSE 05-JUL-79 ORIGINATION

PARAMETER DIMENS=33, NUL=9
BYTE DATAFI(DIMENS)
INTEGER LUANDAT

PROCEDURE

ASK FOR DATAFILENAME.

WRITE(5,1#)
FORMAT( 'WHAT IS THE NAME OF THE DATAFILE TO BE PLOTTED?')
READ(5,2#)DATAFI
FORMAT((DIMENS),A1)

INSERT END-OF-STRING MARK.

DATAFI(DIMENS)=NUL

OPEN THE DATAFILE ON LOGICAL UNIT LUANDAT.

OPEN(UNIT=LUANDAT, NAME=DATAFI, TYPE='OLD', SHARED, READONLY)
RETURN
END
SUBROUTINE RANGEI(RAWXMN, RAWXMX, RAWYMN, RAWYMX)

PURPOSE: TO ASK FOR THE PLOTTING RANGES.

V.R NAME DATE COMMENTS
S.1 J.M. KRESSE 09-JUL-79 ORIGINATION

C-----DECLARATIONS

PARAMETER DEFXMN=-18., DEFINXMX=68., DEFYMN=-18., DEFYMX=38.

INTEGER NCHARS

REAL RAWXMN, RAWXMX, RAWYMN, RAWYMX

C-----PROCEDURE

ASK FOR THE DESIRED RANGE.

WRITE(5,10)DEFXMN, DEFINXMX, DEFYMN, DEFYMX

10 FORMAT(' PLEASE ENTER THE RANGES IN THE X AND Y DIRECTIONS, ' 1
C ' RESPECTIVELY, / ' 2 ' (FOR THE DEFAULT VALUES OF ',F6.2,', ',F5.2,', ',F6.2,' 3 ', ',F5.2,', HIT RETURN.)')

READ(5,20)NCHARS, RAWXMN, RAWXMX, RAWYMN, RAWYMX

20 FORMAT(1X,6I4)

IF (NCHARS.NE.0) GO TO 30

RAWXMN=DEFXMN

RAWXMX=DEFXMX

RAWYMN=DEFYMN

RAWYMX=DEFYMX

GO TO 30

CONTINUE

RETURN

END
SUBROUTINE AXES1(RAWXMN, RAWXMM, RAWYMN, RAWYMX, VXWIND, VYWIND)

PURPOSE: TO DRAW THE AXES.

C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C

V.R NAME DATE COMMENTS
0.1 J.M. KRESSE 05-JUL-79 ORIGINATION

C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C

DECLARATIONS

PARAMETER CCW=1, CW=-1, LMASK1=-21846, LMASK2=-38584
PARAMETER MINDIV=5, UINDIV=10, YUNDIV=5
INTEGER DIR, NX, NY, VNX, VNY
REAL RAWXMN, RAWXMM, RAWYMN, RAWYMX, VANGLE, VAXLEN, VDV
REAL VFVAL, VX, VY, VXFACT, VXWIND, VY, VYD, VXFACT, VYWIND, XD, YD

PROCEDURE

DRAW THE X AXIS.

VF=0.
CALL RAVTOV(0., RAWXMN, RAWXMM, VXWIND, VY)
DIR=CCW
VAXLEN=VXWIND
VANGLE=0.
VF=VXWIND
VDV=RAWXMN
VDV=UINDIV
VXFACT=VXWIND/((RAWXMX-RAWXMN)/UINDIV)
VAXIS(VX, VY, DIR, VAXLEN, VANGLE, VFVAL, VDV, VXFACT)

DRAW THE Y AXIS.

VF=0.
CALL RAVTOV(0., RAWYMN, RAWYMX, VXWIND, VX)
DIR=CCW
VAXLEN=VYWIND
VANGLE=0.
VF=RAWYMN
VDV=UINDIV
VYFACT=VYWIND/((RAWYMX-RAWYMN)/YUNDIV)
VAXIS(VX, VY, DIR, VAXLEN, VANGLE, VFVAL, VDV, VYFACT)

DRAW THE MAJOR GRID.

VF=0.
VF=0.
VNX=FIX((RAWXMX-RAWXMN)/UINDIV)
VXD=VFFACT
VY=FIX((RAWYMX-RAWYMN)/YUNDIV)
VYD=VFFACT
CALL ROTATE(VX, VY, VANGLE, X, Y, ANGLE)
CALL ROTATE(VXD, VYD, VANGLE, XD, YD, ANGLE)
NX=VNY
NY=VNX
CALL GRID(X, Y, NX, XD, Y, YD, LMASK1)

DRAW THE MINOR GRID.
SUBROUTINE LABEL1(RAWMN, RAWMX, VXWIND, VYWIND, SNR, SNRSEG)

PURPOSE: TO LABEL THE PLOT.

V.R DATE NAME COMMENTS
0.1 18-JUL-79 J.M. KRESSE ORIGINATION

PARAMETER HAFLIN=40, HEIGHT=1.75, MARGIN=.5
PARAMETER NCHARD=36, NCHRD0=26, VANGLE=0.
INTEGER ASC1(3), ASC2(3)
INTEGER IX, LINE1(1:NCHARD+1/2), LINE2(HAFLIN), LINE3(1:NCHRD0+1/2)
INTEGER MXCHAR, NCHAR1, NCHAR2, NCHAR3, STR1(2), STR2(5)
REAL ANGLE, RAWMN, RAWMX, SNR, SNRSEG, VX, VXWIND, VY, VYWIND, X, Y

EQUIVALENCE (LINE1(1), STR1(1)), (LINE3(1), ASC1(1))
EQUIVALENCE (LINE3(6), STR2(1)), (LINE3(1), ASC2(1))

C------DEFINITION

C DATA LINE1 /'LO','CA','L','SN','R','(D','B)','-','V','S','
C DATA STR1 /'SN','R=('/
C DATA STR2 /'S','NR','SE','G='/

C------FUNCTION

C START(NCHAR)=YAXIS+(VXWIND-YAXIS)/. . . (FLOAT(NCHAR)*HEIGHT)/2.

C------PROCEDURE

C LOCATE THE Y-AXIS.
C CALL RAWTOV(B, RAWMN, RAWMX, VXWIND, VYAXIS)

C CALCULATE THE MAXIMUM NUMBER OF CHARACTERS PERMITTED.

C MXCHAR=MINB(2*HAFLIN, IFIX((VXWIND-YAXIS-2.*MARGIN)/HEIGHT))
C NCHAR1=MINB(MXCHAR, NCHARD)
C NCHAR3=MINB(MXCHAR, NCHRD0)

C ASK FOR THE LABELING INFORMATION.

C CONTINUE

C WRITE(5,2#1MXCHAR
C FORMAT(' HOW DO YOU WANT TO LABEL THE PLOT?/
C '(','13',' CHARACTERS, MAXIMUM')
C READ(5,3#NCHAR2,(LINE2(IX), IX=1,(NCHAR2+1)/2)
C FORMAT(6#,(NCHAR2+1)/2)A2)
C IF(NCHAR2.GT.MXCHAR)GO TO 1#

C START THE FIRST LINE AT THE TOP, CENTERED BETWEEN THE
C Y-AXIS AND THE RIGHT EDGE.

VX=START(NCHAR1)
VY=VYWIND-MARGIN-HEIGHT
CALL ROTATE(VX, VY, VANGLE, X, Y, ANGLE)
CALL SYMBOL(X,Y,HEIGHT,LINe1,ANGLE,NCHAR1)

PLOT THE SECOND LINE.

VX=START(NCHAR2)
VV=VV-2.*HEIGHT
CALL ROTATE(VX,VV,ANGLE,X,Y,ANGLE)
CALL SYMBOL(X,Y,HEIGHT,LINe2,ANGLE,NCHAR2)

GENERATE THE THIRD LINE.

ENCODER(6,4#ASC1)SNR
ENCODER(6,4#ASC2)SNRSEG
FORMAT(F6.2)

PLOT THE THIRD LINE.

VX=START(NCHAR3)
VV=VV-2.*HEIGHT
CALL ROTATE(VX,VV,ANGLE,X,Y,ANGLE)
CALL SYMBOL(X,Y,HEIGHT,LINe3,ANGLE,NCHAR3)
RETURN
END
SUBROUTINE VAXIS(VX, VY, DIR, VXLEN, VANGLE, VFVAL, VDV, FACT)

PURPOSE: TO DRAW A VIRTUAL AXIS.

V.R NAME DATE COMMENTS
C 1 J.W. KRESSE 9-JUL-79 ORIGINATION

DECLARATIONS

INTEGER DIR
REAL ANGLE, AXLEN, FACT, VANGLE, VXLEN, VDV, VFVAL, VX, VY, X, Y

PROCEDURE

ROTATE

CALL ROTATE(VX, VY, VANGLE, X, Y, ANGLE)

FIX ABSOLUTE

X=X/FACT
Y=Y/FACT
AXLEN=VXLEN/FACT
CALL FACTOR(FACT)
CALL AXIS(X, Y, ' ', DIR, AXLEN, ANGLE, VFVAL, VDV)
CALL FACTOR(1.)
RETURN
END
SUBROUTINE POINTI(VX,VY)

PURPOSE: TO PLOT A POINT CORRESPONDING TO (RAWX,RAWY).

V.R NAME DATE COMMENTS
S.1 J.W. KRESSE 10-JUL-79 ORIGINATION

PARAMETER HEIGHT=.576, ITEXT=11, NC=-1, VANGLE=8.

REAL ANGLE,VX,VY,X,Y

CALL ROTATE(VX,VY,VANGLE,X,Y,ANGLE)
CALL SYMBOL(X,Y,HEIGHT,ITEXT,ANGLE,NC)
RETURN
END
SUBROUTINE RAWTOV(RAW, RAWMN, RAWMX, VWIND, V)

PURPOSE: TO CONVERT THE RAW DATA TO VIRTUAL DATA.

V.R DATE NAME COMMENTS
R.1 16-JUL-79 J.M. KRESSE ORIGINATION

-----DECLARATIONS

REAL RAW, RAWMN, RAWMX, V, VWIND

-----PROCEDURE

V = AMIN1(1., AMAX1(0., (RAW - RAWMN)/(RAWMX - RAWMN))) * VWIND

RETURN

END
SUBROUTINE ROTATE(VX, VY, VANGLE, X, Y, ANGLE)

PURPOSE: TO ROTATE THE VIRTUAL DATA TO PHYSICAL DATA.

NR DATE NAME COMMENTS
#1 18-JUL-79 J.M. KRESSE ORIGINATION

REAL ANGLE, VANGLE, VX, VY, X, Y

PROCEDURE

X = VX
Y = VY
ANGLE = VANGLE + 90.
RETURN
END
PROGRAM BFCALC

PURPOSE: TO CALCULATE THE BLOCK SIGNAL STRENGTH AND THE BLOCK AVERAGE BUFFER LENGTH.

V.R. DATE NAME COMMENTS
0.1 22-JUL-79 J.M. KRESSE ORIGINATION

---DECLARATIONS---

PARAMETER LUNIN=1, LOUT=2, LUNTI=5, MINSIG=-60, MINSIG=-60.
PARAMETER NAMLEN=33, NO=FALSE, NUL=8, REF=1, YES=TRUE.
BYTE FNAME(NAMLEN)
INTEGER BLKLEN, ICNT
LOGICAL EOF
REAL BUFVAVG, BuFCNT, BUFSUM, SIG, SIGSTR, SIGSUM

---PROCEDURE---

ASK FOR THE INPUT FILENAME, AND OPEN THE INPUTFILE ON LOGICAL UNIT LUNIN.

WRITE(LUNTI,18)
18 FORMAT(' WHAT IS THE INPUT FILENAME?')
READ(LUNTI,28)NAME
WRITE(LUNTI,38)NAME
38 FORMAT(' WHAT IS THE OUTPUT FILENAME?')
READ(LUNTI,28)NAME
WRITE(LUNTI,48)NAME
48 FORMAT(' WHAT BLOCKLENGTH DO YOU WANT TO USE?')
READ(LUNTI,58)BLKLEN
EOF=NO
DO UNTIL (EOF.EQ.YES)

CONTINUE
SIGSUM=0.
BUFSUM=0.
ICNT=0.
DO UNTIL ((ICNT.EQ.BLKLEN).OR.(EOF.EQ.YES))
CONTINUE
READ(LUNIN,58)SIG, BUFCTN
SIGSUM=SIGSUM+ABS(SIG)
BUFSUM=BUFSUM+BUFCNT
ICNT=ICNT+1
GO TO 88
CONTINUE
EOF=YES
GO TO 88
CONTINUE
IF(.NOT.((ICNT.EQ.BLKLEN).OR.(EOF.EQ.YES)))GO TO 68
CONTINUE
IF(ICNT.EQ.8) GO TO 128
IF(SIGSUM.EQ.8.) GO TO 98
SIGSTR=AMIN(MAXSIG,AMAX)(MINSIG,(28.*
ALOG10(SIGSUM/(FLOAT(ICNT)*REF))))
GO TO 108
SIGSTR=MINSIG
GO TO 188
CONTINUE
BUFAVG=BUFSUM/FLOAT(ICNT)
WRITE(LUNOUT,118)SIGSTR,BUFAVG
FORMAT(F7.2,F18.2)
GO TO 128
CONTINUE
IF(EOF.EQ.NO) GO TO 58
CONTINUE
CLOSE(UNIT=LUNIN)
CLOSE(UNIT=LUNOUT)
STOP
END
PROGRAM BFLOT

PURPOSE: TO PLOT THE AVERAGE BUFFER LENGTH VS. LOCAL SIGNAL STRENGTH.

V.R DATE NAME COMMENTS
#1 23-JUL-79 J.M. KRESSE ORIGNATION

-----DECLARATIONS-----

PARAMETER ENDPLT=999, LUNDAT=7, NEWORG=-3, PXMAX=8.5, PXWIND=7.5
PARAMETER PYMAX=10.56, PYWIND=9.5
REAL PXVO, PYVO, RAWX, RAWXMN, RAWMX
REAL RAWV, RAWYMN, RAWYMX
REAL VX, VXWIND, VY, VYWIND

-----DEFINITIONS-----

VXWIND=PYWIND
VYWIND=PXWIND

-----INITIALIZATION-----

CALL PLOTS(0,0,0)

-----PROCEDURE-----

OPEN THE DATAFILE ON LOGICAL UNIT LUNDAT.

CALL OPENI(LUNDAT)

ASK FOR THE PLOTTING RANGES.

CALL RANGE2(RAWXMN, RAWMX, RAWYMN, RAWYMX)

MOVE THE ORIGIN TO THE VIRTUAL ORIGIN.

PXVO=PXMAX-(PXMAX-PXWIND)/2.
PYVO=(PYMAX-PYWIND)/2.
CALL PLOT(PXVO, PYVO, NEWORG)

DRAW THE VIRTUAL AXES.

CALL AXES2(RAWXMN, RAWMX, RAWYMN, RAWYMX, VXWIND, VYWIND)

DO UNTIL EOF

CONTINUE

READ(LUNDAT,* ,END=29) RAWX, RAW

CALL RAWTOV(RAWX, RAWXMN, RAWMX, VXWIND, VX)

CALL RAWTOV(RAWX, RAWYMN, RAWYMX, VYWIND, VY)

CALL POINT1(VX, VY)

GO TO 18

CONTINUE

LABEL THE PLOT.
SUBROUTINE RANGE2(RAWXMN, RAWXMX, RAWYMN, RAWYMX)

PURPOSE: TO ASK FOR THE PLOTTING RANGES.

V.R DATE NAME COMMENTS
0.1 23-JUL-79 J.M. KRESSE ORIGINATION

C-----DECLARATIONS

INTEGER NCHARS
REAL RAWXMN, RAWXMX, RAWYMN, RAWYMX

C-----PROCEDURE

ASK FOR THE DESIRED RANGE.

WRITE(5,1) DEFXMN, DEFXMX, DEFYMN, DEFYMX
1 FORMAT(' PLEASE ENTER THE RANGES IN THE X AND Y DIRECTIONS,
1 RESPECTIVELY. '/
2 ' (FOR THE DEFAULT VALUES OF '.F6.2', '.F5.2', '.F8.2,
3 ' , '.F6.2', HIT RETURN.)')
READ(5,2) NCHARS, RAWXMN, RAWXMX, RAWYMN, RAWYMX
2 FORMAT('I11 CHAR. NO.'GO TO 3
3 IF(NCHARS.NE.8) GO TO 3
4 IF(NCHARS.NE.8) GO TO 3
5 IF(NCHARS.NE.8) GO TO 3
6 CONTINUE
7 RETURN
8 END
SUBROUTINE AXES2(RAWXMN,RAWXMX,RAWYMN,RAWYMX,VXWIND,VYWIND)

PURPOSE: TO DRAW THE AXES.

V.R DATE NAME COMMENTS
8.1 23-JUL-79 J.M. KRESSE ORIGINATION

PARAMETER CCW=1,CW=-1,LMASK1=-21846,LMASK2=-30584
PARAMETER MINDIV=5,XUNDIV=18,YUNDIV=580.
INTEGER DIR,NX,NY,VNX,VNY
REAL RAWXMN,RAWXMX,RAWYMN,RAWYMX,VANGLE,VAXLEN,VDV
REAL VFVAL,VX,VXD,VFFACT,VXWIND,VY,VYD,VYFACT,VYWIND,XD,YD

DRAW THE X AXIS.

CALL RAWTOV(B,RAWXMN,RAWXMX,VXWIND,VY)
DIR=CW
VAXLEN=VXWIND
VANGLE=90.
VFVAL=RAWXMN
VDV=XUNDIV
VFFACT=VXWIND/((RAWXMX-RAWXMN)/XUNDIV)
CALL VAXIS(VX,VY,DIR,VAXLEN,VANGLE,VFVAL,VDV,VFFACT)

DRAW THE Y AXIS.

CALL RAWTOV(B,RAWXMN,RAWXMX,VXWIND,VX)
VY=B.
DIR=CCW
VAXLEN=VYWIND
VANGLE=90.
VFVAL=RAWXMN
VDV=YUNDIV
VFFACT=VYWIND/((RAWYMX-RAWYMN)/YUNDIV)
CALL VAXIS(VX,VY,DIR,VAXLEN,VANGLE,VFVAL,VDV,VFFACT)

DRAW THE MAJOR GRID.

VX=B.
VY=B.
VNX=FIX((RAWXMX-RAWXMN)/XUNDIV)
VXD=VFFACT
VNY=FIX((RAWYMX-RAWYMN)/YUNDIV)
VDV=VFFACT
CALL ROTATE(VX,VY,VANGLE,X,Y,ANGLE)
CALL ROTATE(VXD,VYD,VANGLE,XD,YD,ANGLE)
NX=VNX
NY=VNY
CALL GRID(X,Y,NX,NY,YD,LMASK1)

DRAW THE MINOR GRID.
SUBROUTINE LABEL2(RAWXMN, RAWXMX, VXWIND, VYWIND)

PURPOSE: TO LABEL THE PLOT.

V.R DATE NAME COMMENTS
#1 23-JUL-79 J.M. KRESSE ORIGINATION

PARAMETER HAFLIN=4#, HEIGHT=.175, MARGIN=.5
PARAMETER NCHARD=36, VANGLE=#.
INTEGER IX, LINE1((NCHARD+1)/2), LINE2(HAFLIN)
INTEGER MXCHAR, NCHAR1, NCHAR2

REAL ANGLE, RAWXMN, RAWXMX, VX, VXWIND, VY, VXWIND, X, Y

C―――――――DEFINITION

DATA LINE1 /'AV', 'G', 'BU', 'F', 'LE', 'NG', 'TH', 'V', 'S',
1 'L', 'OC', 'AL', 'S', 'IG', 'MA', 'L', '(D', 'B')/

C―――――――FUNCTION

START(NCHAR)=YAXIS*(VXWIND-YAXIS)/2.-(FLOAT(NCHAR)*HEIGHT)/2.

C―――――――PROCEDURE

LOCATE THE Y-AXIS.

CALL RAWTOV(#, RAWXMN, RAWXMX, VXWIND, YAXIS)

CALCULATE THE MAXIMUM NUMBER OF CHARACTERS PERMITTED.

MXCHAR=MIN#(2*HAFLIN, IFIX((VXWIND-YAXIS-2.*MARGIN)/HEIGHT))
NCHAR1=MIN#(MXCHAR, NCHARD)

ASK FOR THE LABELING INFORMATION.

18 CONTINUE
WRITE(5,20) MXCHAR
FORMAT(' HOW DO YOU WANT TO LABEL THE PLOT?/
1 '('.13,' CHARACTERS, MAXIMUM')
READ(5,30) NCHAR2, (LINE2(1X), IX=1, (NCHAR2+1)/2)
30 FORMAT(6,<NCHAR2+1)/2-A2)
IF(NCHAR2.GT.MXCHAR)GO TO 18 CONTINUE

START THE FIRST LINE AT THE TOP, CENTERED BETWEEN THE
Y-AXIS AND THE RIGHT EDGE.

VX=START(NCHAR1)
VY=VYWIND-MARGIN-HEIGHT
CALL ROTATE(VX, VY, VANGLE, X, Y, ANGLE)
CALL SYMBOL(X, Y, HEIGHT, LINE1, ANGLE, NCHAR1)

C—————-PLOT THE SECOND LINE.

VX=START(NCHAR2)
VY=VY-2.*HEIGHT
PROGRAM BDIFF

PURPOSE: TO CALCULATE THE BLOCK SIGNAL STRENGTH AND THE DIFFERENCE IN THE AVERAGE BUFFER LENGTH.

V.R DATE NAME COMMENTS
M.1 23-JUL-79 J.M. KRESS ORIGINATION

-----DECLARATIONS-----

PARAMETER LUNIN=12, LUNOUT=2, LUNM=5
PARAMETAM LAMLEN=33, NUL=1
BYTE FNAME(MNLEN)
REAL BUFAVG, LAST, SIGSTR

-----PROCEDURE-----

ASK FOR THE INPUT FILENAME, AND OPEN THE INPUTFILE ON LOGICAL UNIT LUNIN.

WRITE(LUNI,1B)
FORMAT(10 WHAT IS THE INPUT FILENAME?)
READ(LUNI,2B) FNAME
FORMAT(1NAMLEN)1A)
FNAME(MNLEN)=NUL
OPEN(UNIT=LUNIN, NAME=FNAME, TYPE='OLD', READEONLY, SHARED)

ASK FOR THE OUTPUT FILENAME, AND OPEN THE OUTPUTFILE ON LOGICAL UNIT LUNOUT.

WRITE(LUNI,3B)
FORMAT(10 WHAT IS THE OUTPUT FILENAME?)
READ(LUNI,2B) FNAME
FNAME(MNLEN)=NUL
OPEN(UNIT=LUNOUT, NAME=FNAME, TYPE='NEW', CARRIAGECONTROL='LIST')

DO UNTIL EOF

CONTINUE
READ(LUNIN,*,END=70) SIGSTR, BUFAVG
BUFF=BUFAVG-LAST
WRITE(LUNOUT,6B) SIGSTR, BUDIF
FORMAT(F7.2,1B.2)
LAST=BUFAVG
GO TO 50

CONTINUE

CLOSE(UNIT=LUNIN)
CLOSE(UNIT=LUNOUT)
STOP
END
PROGRAM BQDFPL

PURPOSE: TO PLOT THE DIFFERENCE IN THE AVERAGE BUFFER LENGTH VS. THE LOCAL SIGNAL STRENGTH.

V.R DATE NAME COMMENTS
# 1 23-JUL-79 J.M. KRESSE ORIGINATION

-----DECLARATIONS

PARAMETER ENDPNL=999, LUNDAT=7, NEWORG=-3, PXMAX=8.5, PXWIND=7.5
PARAMETER PMAX=18.56, PYWIND=9.5
REAL PXVO, PYVO, RAWX, RAWXN, RAWXMX
REAL RAWY, RAWYMN, RAWYMX
REAL VX, VXWIND, VY, VYWIND

-----DEFINITIONS

VXWIND=PYWIND
VYWIND=PXWIND

-----INITIALIZATION

CALL PLOTS(0, 0, 0)

-----PROCEDURE

OPEN THE DATAFILE ON LOGICAL UNIT LUNDAT.
CALL OPEN1(LUNDAT)
ASK FOR THE PLOTTING RANGES.
CALL RANGE3(RAWXN, RAWXMX, RAWYMN, RAWYMX)
MOVE THE ORIGIN TO THE VIRTUAL ORIGIN.
PXVO=PXMAX-(PXMAX-PXWIND)/2.
PYVO=(PYMAX-PYWIND)/2.
CALL PLOT(PXVO, PYVO, NEWORG)
DRAW THE VIRTUAL AXES.
CALL AXES3(RAWXN, RAWXMX, RAWYMN, RAWYMX, VXWIND, VYWIND)
DO UNTIL EOF

10 CONTINUE
11 READ(LUNDAT, *, END=20) RAWX, RAWY
12 CALL RAVTOV(RAWX, RAWXN, RAWXMX, VXWIND, VX)
13 CALL RAVTOV(RAWY, RAWYMN, RAWYMX, VYWIND, VY)
14 CALL POINT1(VX, VY)
15 GO TO 10
16 CONTINUE

LABEL THE PLOT.
CALL LABEL3(RAWXMN, RAWXMN, VXWIND, VYWIND)

CALL PLOT(#..#..ENDPLT)
CLOSE(UNIT=LUNDAT)
STOP
END
SUBROUTINE RANGE3(RAWXMN,RAWXMX,RAWYMN,RAWYMX)

PURPOSE: TO ASK FOR THE PLOTTING RANGES.

V.R DATE NAME COMMENTS
0.1 23-JUL-79 J.M. KRESSE ORIGINATION

--------DECLARATIONS

PARAMETER DEFXMN=-10.,DEFXMX=60.,DEFYMN=-13.,DEFYMX=120.
INTTEGER NCHARS
REAL RAWXMN,RAWXMX,RAWYMN,RAWYMX

--------PROCEDURE

ASK FOR THE DESIRED RANGE.

WRITE(5,10)DEFXMN,DEFXMX,DEFYMN,DEFYMX
10 FORMAT(1X,'PLEASE ENTER THE RANGES IN THE X AND Y DIRECTIONS,'
1                   1 'RESPECTIVELY.'/2
2                  'FOR THE DEFAULT VALUES OF ',F6.2,', ',F5.2,', ',F6.2,',
3                  ',F6.2,'. HIT RETURN.')
READ(5,20)NCHARS,RAWXMN,RAWXMX,RAWYMN,RAWYMX
20 FORMAT(1X,5E14.7)
IF(NCHARS.NE.0)GO TO 30
RAWXMN=DEFXMN
RAWXMX=DEFXMX
RAWYMN=DEFYMN
RAWYMX=DEFYMX
GO TO 30
CONTINUE
RETURN
END
SUBROUTINE AXES3(RAWXMN,RAWXMX,RAWYMN,RAWYMX,VXWIND,VYWIND)

PURPOSE: TO DRAW THE AXES.

V.R DATE NAME COMMENTS
8.1 23-JUL-79 J.M. KRESSE ORIGINATION

--------DECLARATIONS

PARAMETER CCW=1,CW=-1,LMASK1=-2,LMASK2=-30504
PARAMETER MINDIV=5,XUNDIV=18.,YUNDIV=58.
INTEGER DIR,NX,NY,VNX,VNY
REAL RAWXMN,RAWXMX,RAWYMN,RAWYMX,VANGLE,VAXLEN,VDV
REAL VFVAL,VX,VXD,VXFACT,VXWIND,VY,VDV,VYFACT,VYWIND,XD,YD

--------PROCEDURE

DRAW THE X AXIS.

CALL RAWTOV(O.,RAWXMN,RAWXMX,VXWIND,VY)
DIR=CCW
VAXLEN=VXWIND
VANGLE=O.
VFVAL=RAWXMN
VDV=XUNDIV
VXFACT=VXWIND/((RAWXMX-RAWXMN)/XUNDIV)
CALL VAXIS(VX,VY,DIR,VAXLEN,VANGLE,VFVAL,VDV,VXFACT)

DRAW THE Y AXIS.

CALL RAWTOV(O.,RAWXMN,RAWXMX,VXWIND,VX)
VY=O.
DIR=CCW
VAXLEN=VYWIND
VANGLE=90.
VFVAL=RAWXMX
VDV=YUNDIV
VYFACT=VYWIND/((RAWYMX-RAWXMN)/YUNDIV)
CALL VAXIS(VX,VY,DIR,VAXLEN,VANGLE,VFVAL,VDV,VYFACT)

DRAW THE MAJOR GRID.

VX=O.
VY=O.
VNX=IFIX((RAWXMX-RAWXMN)/XUNDIV)
VXD=VXFACT
VNY=IFIX((RAWYMX-RAWXMN)/YUNDIV)
VYD=VYFACT
CALL ROTATE(VX,VY,VANGLE,X,Y,ANGLE)
CALL ROTATE(VXD,VYD,VANGLE,XD,YD,ANGLE)
NX=VNY
NY=VNX
CALL GRID(X,Y,NX,XD,NY,YD,LMASK1)

DRAW THE MINOR GRID.
$4N_{4A} - 0 \times \mu - a^2 > 2$
SUBROUTINE LABEL3(RAWXMN, RAWXM, VXWIND, VYWIND)

PURPOSE: TO LABEL THE PLOT.

V.R DATE NAME COMMENTS
S.1 3-JUL-79 J.M. KRESSE ORIGINATION

PARAMETER HAFLIN=48, HEIGHT=.175, MARGIN=.5
PARAMETER NCHARD=36, VANGLE=0.
INTEGER IX, LINE1((NCHARD+1)/2), LINE2(HAFLIN)
INTEGER MXCHAR, NCHAR1, NCHAR2
REAL ANGLE, RAWXMN, RAWXM, VX, VXWIND, VY, VYWIND, X, Y

-----DEFINITION-----

DATA LINE1 ('D', 'F', 'AV', 'G', 'BU', 'FF', 'ER', 'V', 'S', 'L', 'OC', 'AL', 'S', 'IG', 'NA', 'L', '(D', 'B')/

-----FUNCTION-----

START(NCHAR)=YAXIS+(VXWIND-YAXIS)/2.-((FLOAT(NCHAR)*HEIGHT)/2.

-----PROCEDURE-----

LOCATE THE Y-AXIS.

CALL RAWTOV(E, RAWXMN, RAWXM, VXWIND, VYWIND)

CALCULATE THE MAXIMUM NUMBER OF CHARACTERS PERMITTED.

MXCHAR=MINS((2*HAFLIN*FIX((VXWIND-YAXIS-2.*MARGIN)/HEIGHT))
NCHAR1=MINS(MXCHAR, NCHARD)

ASK FOR THE LABELING INFORMATION.

CONTINUE

WRITE(5,20)MXCHAR

FORMAT(' HOW DO YOU WANT TO LABEL THE PLOT? /

1 ' (' I3 ' CHARACTERS, MAXIMUM')

1 READ(5,20)NCHAR2, LINE2(IX), IX=1,(NCHAR2+1)/2

IF(NCHAR2.GT.MXCHAR)GO TO 18

CONTINUE

START THE FIRST LINE AT THE TOP, CENTERED BETWEEN THE

Y-AXIS AND THE RIGHT EDGE.

VX=START(NCHAR1)

VY=VYWIND-MARGIN-HEIGHT

CALL ROTATE(VX, VX, VY, VANGLE, X, Y, ANGLE)

CALL SYMBOL(X, Y, HEIGHT, LINE1, ANGLE, NCHAR1)

PLOT THE SECOND LINE.

VX=START(NCHAR2)

VY=VY-2.*HEIGHT
APPENDIX C

PLOTTING PROGRAM

This program is used to plot a data file or the difference of two data file on a Versaplot 07 system. It employs the Versaplot-07 PPEP Software Package.

Two options are provided by this program:

1. It can plot a whole data file. The ranges of x-axis and y-axis are specified by the user through a terminal.

2. It can plot a number of sections of a data file. The starting location and number of sections can be specified by the user through a terminal. However, the size of a section is fixed to 1600 samples.

Data must be stored in 1615 format with a standard format header card. Details are shown in the program listing.
C VERSATEC DATAFILE PLOTTING PROGRAM

MAWLN YEH
DATE:6/28/79
PROGRAM NAME: DATAPLOT.FTN

THIS PROGRAM IS USED TO PLOT A SPEECH SAMPLE FILE ON
ON A VERSATEC PRINTER/PLOTTER. THE FORMAT FORMAT FOR
THE SPEECH SAMPLES IS 1615.

THERE ARE TWO FUNCTIONS IN THIS PROGRAM:
1. PLOT A DATA FILE
2. PLOT THE DIFFERENCE BETWEEN TWO DATA FILES

ALSO, TWO OPTIONS ARE AVAILABLE AS FOLLOWS:
1. PLOT THE WHOLE DATA FILE.
2. PLOT PART OF DATA FILE.

DIMENSION X(16#2),Y(16#2)
LOGICAL* NAME1(32),NAME2(32),SYM1(8#),SYM2(8#)
INTEGER* H1(4#),H2(4#),X1(16),X2(16),X3(16)

ASK FOR THE CHOICE

TYPE = ' WHICH FUNCTION DO YOU WANT?'

TYPE = ' 1 = PLOT A DATA FILE'

TYPE = ' 2 = PLOT THE DIFFERENCE BETWEEN TWO FILE (FILE1-FILE2)'

ACCEPT *.IOPT

TYPE = ' DO YOU WANT TO PLOT THE WHOLE DATA FILE?(1=YES,2=NO)'

ACCEPT *.JOPT2

IF(IOPT2 .EQ. 1) GO TO 70

TYPE = ' WHAT IS THE STARTING SAMPLE NUMBER?'

ACCEPT *.ISTART

TYPE = ' NUMBER OF WINDOWS YOU WANT TO PLOT?'

TYPE = ' ( WINDOW SIZE = 16#0 SAMPLES )'

ACCEPT *.NWINDO

IF(IOPT .EQ. 1) GO TO 20

IF(IOPT .NE. 2) GO TO 10

FUNCTION 2

TYPE = ' INPUT 1ST FILE NAME?'

ACCEPT 'NAME1'

NAME1(32)='

OPENUNIT=9,N=NAME1,READONLY,SHARED,TYPE='OLD')

READ9,1#1NSENT1,IREA1,NSAMP1,IPR1,IL0R1,NT0R1,H1

TYPE = ' 1ST FILE HEADER IS'

TYPE 1#01,NSENT1,IREA1,NSAMP1,IPR1,IL0R1,NT0R1,H1

TYPE = ' THE LABEL OF THIS FILE FOR THE PLOT?'

TYPE = ' ( 8# CHARACTERS, MAXIMUM )'

ACCEPT 'SYM1'
FORTRAN IV-PLUS V82-S1

DATAPLOT.FTN /TRI-BLOCKS/WR

##28 TYPE *' INPUT 2ND FILE NAME'
##29 ACCEPT 1000,NAME2
##30 NAME(32)=
##31 OPEN(UNIT=7,NAME=NANE2,READONLY,SHARED,TYPE='OLD')
##32 READ(7,1#01)INSENT,IRATE,NSAMP,1UPER,1LOWR,TERM2,H2
##33 TYPE *' 2ND FILE HEADER IS'
##34 TYPE 1#01,INSENT,IRATE,NSAMP,1UPER,1LOWR,TERM2,H2
##35 TYPE *' THE LABEL OF THIS FILE FOR THE PLOT'
##36 TYPE *' ( 80 CHARACTERS, MAXIMUM )'
##37 ACCEPT 1000,SYH2

##38 PRODUCE A SCRATCH FILE

##39 OPEN(UNIT=8,NAME='SCRATCH.DAT',TYPE='SCRATCH')
##40 READ(9,1#04,END=4#I1)
##41 READ(7,1#04,END=4#I2)
##42 DO 39 I=1,16
##43 X3(I)=X1(I)-X2(I)
##44 GO TO 50
##45 39 REWIND 8
##46 GO TO 50

##47 TYPE *' INPUT FILE NAME'

##48 ACCEPT 1000,NAME1
##49 NAME(32)=
##50 OPEN(UNIT=8,NAME=NANE1,READONLY,SHARED,TYPE='OLD')
##51 TYPE *' INPUT FILE HEADER IS'
##52 READ(8,1#01)INSENT,IRATE,NSAMP,1UPER,1LOWR,TERM2,H1
##53 TYPE 1#01,INSENT,IRATE,NSAMP,1UPER,1LOWR,TERM2,H1
##54 TYPE *' THE LABEL OF THIS FILE FOR THE PLOT'
##55 TYPE *' ( 80 CHARACTERS, MAXIMUM )'
##56 ACCEPT 1000,SYH1

##57 1001 FORMAT(615,18X,4#A1)
##58 1002 FORMAT(32A1)
##59 1003 FORMAT(16I5)
##60 1004 FORMAT(8#A1)

##61 PLOTTING ROUTINE

##62 TYPE *' THE WIDTH FOR X-AXIS IS "X-WIDTH" INCHES FOR 16# SAMPLES'
##63 TYPE *' THE WIDTH FOR Y-AXIS IS" INCHES.'
##64 TYPE *' INPUT WINDOW SIZE(XMAX,XMIN,YMAX,YMIN)'
##65 ACCEPT *' XMAX,XMIN,YMAX,YMIN

##66 TYPE *' X-WIDTH,X-SCALING FACTOR,Y-STARTING VALUE,Y-SCALING FACTOR ?'
##67 TYPE *' E.G. 16. 100. -2400. 600. ( FOR ONE FILE )'
##68 TYPE *' E.G. 16. 100. -500. 150. ( FOR THE DIFFERENCE FILE )'
##69 ACCEPT *' XWIDTH,XSCAL,YSTART,YSCAL

##70 IF(IOT2 .EQ. 1)GO TO 99

##71 ISAMP=16

##72 IF(ISAMP .GE. ISTART)GO TO 100

##73 ISAMP=ISAMP+16

##74 READ(8,1#04)X1

##75 GO TO 99

##76 SAMP=ISAMP

##77 SAMP=ISAMP
CALL PLOTS(S,...,S.)
CALL PLOT(S,...,3)
CALL WINDOW(XMAX,XMIN,YMAX,YMIN)
CALL SYMBOL(2,9,S,SYM,1,88)
IF(IOPT.EQ.1) GO TO 685
CALL SYMBOL(2,8,0,2,SYM,2,88)
CALL PLOT(1,...,1,3)
X(1601)=SAMP
X(1602)=SCALE
Y(1601)=YSTART
Y(1602)=YSCALE
CALL AXIS(S,...,13HSAMPLE NUMBER,-13,X,WIDTH,...

1
Y(1601),Y(1602))
CALL AXIS(S,...,9HAMPLITUDE,9,8,98,98
1
Y(1601),Y(1602))
CALL LINE(X,Y,1600,1,8,8)
IF(IOPT1.EQ.1) GO TO 581
WINDO=WINDO+1
IF(WINDO.EQ.8) GO TO 118
XTEMP=X(1600)
YTEMP=Y(1600)
ICOUNT=0
READ(1,1004,END=788) X1
DO 333 J=1,16
ICOUNT=ICOUNT+1
333
Y(J)=Y(J+1)
IF(ICOUNT.NE.16) GO TO 688
GO TO 688
DO 988 I=ICOUNT+1,1688
988
Y(I)=0
DO 444 I=1,1688
444
FA=FA+1
X(I)=FA
XX=X(I)-X(1601)
YY=Y(I)-Y(1601)
XTEMP1=XTEMP-X(1601)
YTEMP1=YTEMP-Y(1601)
CALL PLOT(XTEMP1,YTEMP1,3)
CALL PLOT(X,YY,2)
X(1601)=X(1601)+1688.
CALL PLOT(16,...,3)
CALL AXIS(S,...,13HSAMPLE NUMBER,-13,16,...

1
X(1601),X(1602))
IF(END.NE.1) GO TO 688
CALL LINE(X,Y,1688,1,8,8)
CALL PLOT(S,...,999)
This set of D/A programs is used for a digital-to-analog converting operation employing the DATEL's ST-PDP device which is an analog I/O module for DEC PDP-11 minicomputers. The ST-PDP device has three different modes:

1. Program Control Interface
2. Interrupt Serviced Interface
3. Direct Memory Address

The set of D/A programs uses the Program Control Interface mode.

The set consists of six modules:

1. DA.CMD — The indirect command file to execute the D/A converting operation.
2. DASP.FTN — The Fortran file to output a speech data file to the D/A converter.
3. DARAMP.FTN — The Fortran file to output a ramp function to the D/A converter in order to check the timing and operation.
4. DA64.MAC — The MACRO-11 assembly language subroutine for 6400 samples per second sampling frequency.
5. DA80.MAC — The MACRO-11 assembly language subroutine for 8000 samples per second sampling frequency.
6. COMMON.MAC — The MACRO-11 assembly language file used to build a common device block inside the PDP-11 operating system.

This set of programs allows a user to output a data file from disk to the D/A converter. Data must be stored in 1615 format with a standard format header card. Details are shown in the program listing.
A common device block has to be built the first time the D/A device is used. The way to build is shown below. Further details can be found in the I/O Driver Reference Manual of the PDP-11 RMS-I1M operating system.

1. Logon a privileged UIC: For future reference, use UIC = [3,1]
2. > MAC COMMON = COMMON
   (underline means the prompt of the computer)
3. > SET /UIC = [1,1]
4. > TKB
   TKB> COMMON/MM, LP:, ST: COMMON/PI/-HD = [3,1] COMMON
   TKB> /
   ENTER OPTIONS:
   TKB> PAR = COMMON: 0:16000
   TKB> STACK = 0
   TKB> 1
5. Logoff

The procedure to execute the D/A modules is as follows:

1. Logon a privileged UIC
2. Execute the indirect command file DA.CMD (i.e., TYPE @ DA)

During the execution, the D/A modules will ask for additional information to set up the D/A operation. It will also set the CPU at the highest hardware priority, i.e., it occupies the CPU. Thus, it will suspend other users' programs and stop the real-time clock. After the D/A operation, it will restart other users' programs and the real-time clock.

The program listings are as follows:
speech recording program
main program: dasp.ftn

---

date: 4/87/79
mawlin yeh

indirect command file: da.cmd --- execute d/a program

main programs: dasp.ftn --- read speech data file
darmp.ftn --- test d/a terminal
da64.mac --- 6.4khz d/a output
da88.mac --- 8khz d/a output

this program will read data from a data file and then
output to d/a converter.

this program will set cpu at the highest hardware priority
so, it will suspend other users' programs and stop the real-time clock.
afterward, the real-time clock will not be accurate.

the procedure to use this file are as follows:
1. logon privilege uic
2. type @da

the command file will do the following things:
1. it will ask which function are you going to use
   there are three functions in this file:
   a. ramp function -- test d/a & check delay time.
   b. 6.4khz d/a output rate
   c. 8khz d/a output rate
2. it will produce suitable task.
3. then, it will set one device common region called
   common.
4. then, it will install and fix the proper task in
   order to avoid the relocation & the using of cache
   memory.
5. execute the task.
6. after execution, it will remove the task and
   one device common region

integer*2 filnam(16)
common /ibuf/iis(16028)
common /inf/na,nb,nc

11 type = '* file name?'
accept 2000,filnam
200 format(16a2)
filnam(16)=
open(unit=2,type= 'old',readonly,shared,name=filnam)
input speech data
C--- ERROR CHECKING IN SIZE OF DATA
#15  80 IF(NSAMP .LE. 1602#) GO TO 333
#16   TYPE = 'YOUR DATA FILE IS LARGER THAN DATA BUFFER I'
#17   TYPE = '.# OF SAMPLE =', NSAMP
#18   NSAMP=1602#
#19   NF=1

C--- ERROR CHECKING IN MAGNITUDE OF DATA
#20  333 DO 54 I=1, NSAMP
#21  54 IF(I>=IIS(I)) GO TO 334
#22   IF(I<.LT.-2#4#) GO TO 334
#23   GO TO 54
#24  334 MG=MG+I
#25  56 IF(MG .EQ. 1) GO TO 336
#26   NF=1
#27   TYPE = 'DATA OUTSIDE RANGE OF D/A DATA REGISTER I'
#28  335 TYPE = 'IIS(.,,)=', IIS(I)
#29   54 CONTINUE
#30   IF(NF .EQ. 0) GO TO 22
#31   TYPE = '. DO YOU WANT CONTINUE [Y/N] ?'
#32   336 ACCEPT 336,Q
#33  338 FORMAT(A1)
#34   IF(Q .EQ. 'Y') STOP
#35   MG=#
#36   NF=#

C--- CONDITIONS
#37  22 TYPE = '.' CHANNEL # ?'
#38   ACCEPT = NCHNL
#39   NRT=#

C--- OUTPUT DATA TO D/A
C--- HAVE DELAY TIMING CONTROL
#40  12 CALL DA(NSAMP, NCHNL, NRT)

C--- CONTINUE?
#41  13 TYPE = 'DO YOU WANT TO CONTINUE?'
#42   WRITE(*,*) '1=NEW FILE. 2=OUTPUT AGAIN. 3=STOP.'
#43   ACCEPT = M
#44   IF(M .NE. 1) GO TO 33
#45   CLOSE(UNIT=2)
#46   GO TO 11
#47  33 IF(M .EQ. 2) GO TO 12
#48   IF(M .EQ. 3) STOP
#49   TYPE = '.HA,MB,NC
#50   TYPE = '. ?'
#51   GO TO 13
#52   END
C--- D/A CONVERTER TESTING PROGRAM
C--- MAIN PROGRAM: DARAMP.FTN
C---
C---
DATE: 3/25/79
NAVIN YEH
C---
MAIN PROGRAM: DASP.FTN --- READ SPEECH DATA FILE
DARAMP.FTN --- TEST D/A TERMINAL
SUB PROGRAM: DA44.MAC --- 6.4KHZ D/A OUTPUT
DA88.MAC --- 8KHZ D/A OUTPUT
C---
THIS PROGRAM WILL PRODUCE A SET OF DATA FROM A RAMP FUNCTION,
AND THEN TRANSFER IT TO D/A CONVERTER.
C---
THIS PROGRAM WILL SET CPU AT THE HIGHEST HARDWARE PRIORITY
SO, IT WILL SUSPEND OTHER USERS' PROGRAM AND STOP THE REAL-TIME CLOCK.
AFTERWARE, THE REAL-TIME CLOCK WILL NOT BE ACCURATE.
C---
THE PROCEDURE TO USE THIS FILE ARE AS FOLLOWS:
1. LOGN PRIVILEGE UIC
2. TYPE #DA
C---
THE COMMAND FILE WILL DO THE FOLLOWING THINGS:
1. IT WILL ASK WHICH FUNCTION ARE YOU GOING TO USE
   THERE ARE THREE FUNCTION IN THIS FILE:
   A. RAMP FUNCTION -- TEST D/A & CHECK DELAY TIME.
   B. 6.4KHZ D/A OUTPUT RATE
   C. 8KHZ D/A OUTPUT RATE
2. IT WILL PRODUCE SUITABLE TASK.
3. THEN, IT WILL SET ONE DEVICE COMMON REGION CALLED
   COMMON.
4. THEN, IT WILL INSTALL AND FIX THE PROPER TASK IN
   ORDER TO AVOID THE RELOCATION & THE USING OF CACHE
   MEMORY.
5. EXECUTE THE TASK.
6. AFTER EXECUTION, IT WILL REMOVE THE TASK AND
   ONE DEVICE COMMON REGION
C---
C---
INTEGER#4 HIGH
COMMON BLOCKS
COMMON /IBUF/11S(1682#)
COMMON /INF/NA,NS,NC
C--- CONDITIONS:
33 TYPE =',' STARTING SAMPLE # ?'
ACCEPT =',NSTART
TYPE = ',' END OF SAMPLE ?'
ACCEPT =',NEND
TYPE = ',' SAMPLE INCREMENT ?'
ACCEPT =',NS1
TYPE = ',' REPEAT # ?'
C--- PRODUCE RAMP DATA

IK=1
DO 44 I=NSTART,NEND,NSI
IK=IK+1
X=I-1./6432.**(I-1)
X1=2#47.*X
X1=X1-#.6
IF(X1.LT. #.5) GO TO 44
X1=X1+1.
44 IIS(IK)=X1
GO TO 55
C--- PRODUCE SQUARE DATA

3# TYPE = 'AMPLITUDE ?'
ACCEPT = ,NHIGH
IK=1
DO 99 I=1,N,
IK=IK+1
44 IIS(IK)=NHIGH
99 IIS(IK)=#.
55 IF(IK .GT. 16#25) GO TO 455
C--- OUTPUT DATA TO D/A
C--- HAVE DELAY TIMING CONTROL
CALL DA(IK,NCHNL,NRT)
C--- CONTINUE?

13 TYPE = 'DO YOU WANT TO CONTINUE?'
WRITE(*,*)'1-NEW PARAMETERS. 2-OUTPUT AGAIN. 3=STOP.'
ACCEPT = ,M
IF(M .EQ. 1) GO TO 33
IF(M .EQ. 2) GO TO 12
IF(M .EQ. 3) STOP
TYPE = 'NA,MB,NC'
45 TYPE = ' ?'
GO TO 13
455 TYPE = ',SAMPLE # IS LARGER THAN BUFFER !'
STOP
END
D/A OUTPUT PROGRAM
SUB PROGRAM: DA64 MAC

DATE: 3/25/79
MAULIN YEH

MAIN PROGRAM: DASP.FTN
SUB PROGRAM: DARAMP.FTN
SUB PROGRAM: DA64.MAC
DA8#.MAC

THIS SUBROUTINE TRANSfers DATA TO D/A DATA REGISTER
IT USE ONE DEVICE COMMON BLOCK ---- COMMON

THIS PROGRAM IS USED FOR THE ST-PDP PROGRAM CONTROLLED INTERFACE
D/A DEVICE.

.TITLE SUB
.PSECT
.GLOBL DA

; TRANSFER ARGUMENTS
DA: MOV (R5)+,R0 ; GET NUMBER OF ARGUMENTS
    MOV #SAMNOR1 ; PSELECT POINTER
    MOV (R0)+,R2 ; GET ADDR IN CALLING PROG.
    MOV (R2),(R1)+ ; MOVE ARGs.

; INITIALIZE PARAMETERS
START: MOV @CHANEL,R4 ; RESET DATA REGISTER
       MOV R4,#LDADDR ; # OF SAMPLES
       MOV @SAMNO,R1 ; STARTING ADDRESS OF DATA
       MOV @BUF1,R3 ; # OF REPEAT
       MOV @REPEAT,R2 ; STORE CURRENT PSW
       MOVB @PSW,@TEMP ; SET THE HIGHEST HARDWARE PRIORITY

; OUTPUT LOOP
LOOP: MOV R4,#LDADDR ; CHANNEL ADDRESS
      MOV (R3)+,#LDADDR ; DATA OUTPUT

; INITIALIZE
BPL 16

; EXIT

D/A OUTPUT PROGRAM
SUB PROGRAM: DA88.MAC

DATE: 3/25/79
MAVIN VEN

MAIN PROGRAM: DASP.FTN
DARAMP.FTN

SUB PROGRAM: DA64.MAC
DA88.MAC

THIS SUBROUTINE TRANSFERS DATA TO D/A DATA REGISTER
IT USE ONE DEVICE COMMON BLOCK --- COMMON

THIS PROGRAM IS USED FOR THE ST-PDP PROGRAM CONTROLLED INTERFACE
D/A DEVICE.

.TITLE SUB
.PSEC
.GLOBL DA

20 DA: TRANSFER ARGUMENTS
21 MOV (R5)+,R8 ; GET NUMBER OF ARGUMENTS
22 MOV @SAMHO, R1 ; PSEC POINTER
23 MOV (R5)+,R2 ; GET ADDR IN CALLING PROG.
24 MOV (R2), (R1)+ ; MOVE ARRS.
25 DEC R5
26 BPL 16

27 INITIALIZE PARAMETERS
28 START: RESET DATA REGISTER
29 MOV @CHANEL,R4
30 MOV R4,#0LDAADR
31 MOV @SAMP, R1 ; # OF SAMPLES
32 MOV #BUF1.R3 ; STARTING ADDRESS OF DATA
33 MOV #REPEAT,R2 ; # OF REPEAT
34 MOV @PSW,#TEMP ; STORE CURRENT PSW
35 MOV #34,#0PSW ; SET THE HIGHEST HARDWARE PRIORITY
36 OUTPUT LOOP
37 MOV R4,#0LDAADR ; CHANNEL ADDRESS
38 LOOP: MOV (R3)+,#0LDAADR ; DATA OUTPUT
SUB   MACRO H111# 31-MAR-89 11:20 PAGE 1-2

116  #0316   #1437   #4422'   MOV   R4, @LDAADR   ; SET CHANNEL #
117  #0314   #12737  #4422'   MOV   #8, @LDAADR   ; RESET D/A
119  #0322   #12783  #4422'   MOV   #BUF1, R3   ; STARTING ADDRESS OF DATA
121  #0326   #05382   DEC   R2   ;
122  #0330   #108256  BPL   LOOP   ; REPEAT
128  #0332   #13737  #15776'   MOV   @TEMP, @PSW   ; RESTORE PSW
129  #0348   #44227   RTS   PC   ; TRANSFER FINISHED

; DATA BUFFER
124  .PSECT   IBUF, D, OVR, GBL
125  .GLOBAL   BUF1
126  .PSECT   TEMPARY STORAGE SPACE
127  .PSECT   DATA, D, OVR, GBL
128  .GLOBAL   TEMP
129  .PSECT   TEMP STORAGE FOR PSW
130  .PSECT   INFORMATION TRANSFER
132  .GLOBAL   SAMNO, CHANH, REPEAT
133  .PSECT   COMMON BLOCK
134  .PSECT   COMMON, D, OVR, GBL
135  .PSECT   LSR
136  .PSECT   LDAADR
138  .PSECT   LOADR
140  .PSECT   BLK1
142  .PSECT   BLK2
143  .PSECT   BLK3
144  .PSECT   BLK4
145  .PSECT   PSW
146  .GLOBAL   END
DEFINITION OF DEVICES COMMON BLOCK

THIS PROGRAM IS USED TO SET A DEVICE COMMON BLOCK.

PSECT COMMON, RW, D, G8L, REL, OVR

LSR: .BLKW 1
LDAADR: .BLKW 1
RADDR: .BLKW 1
LADADR: .BLKW 1
LDADR: .BLKW 1
LMADR: .BLKW 1
LWDCT: .BLKW 1
PSW: .BLKW 1
RSR=LSR
RADADR=LADADR
RMADR=LMADR
.END
INDIRECT COMMAND FILE FOR D/A CONVERTING

THIS COMMAND IS USED TO RUN THE D/A PROGRAM.
YOU HAVE TO USE THE PRIVILEGE UIC.
SO, BE CAREFUL!
WHEN YOU EXECUTE THIS FILE, DON'T DO OTHER ACTIONS
UNTIL IT FINISHES.

SET /MAIN=COMMON:762#:15#:DEV
INS #K:11,13COMMON
COMMON DEVICE REGION HAS BEEN SET UP.

.ENABLE SUBSTITUTION
.SETF A1
.SETF A2
.SETF A3
.SETF A4
.SETF A5
.SETF A6
.SETF A7
.SETF A8

1. 6.4KHZ SPEECH
2. 6.4KHZ SPEECH
3. 6.4KHZ RAMP
4. 6.4KHZ RAMP

..ASK A 6.4KHZ SPEECH
..IFF A .GOTO 1B
..IFT A1 .GOTO 2B
.SETT A1

..WAIT FOR
FOR DASP=DASP
..IFT A2 .GOTO 3B
.SETT A2

..WAIT MAC
MAC DAS4=DAS4
..SETS B1 "DASP54"
..IFT A3 .GOTO 2B
.SETT A3
..SETS B2 "DAS4"
..SETS B3 "DASP"
.GOTO 1B
..ASK C 6.4KHZ SPEECH
..IFF C .GOTO 5B
..IFT A1 .GOTO 8B
.SETT A1

..WAIT FOR
FOR DASP=DASP
..IFT A4 .GOTO 7B
.SETT A4

..WAIT MAC
MAC DASH=DASH
..SETS B1 "DASP8B"
..IFT A5 .GOTO 2B
.SETT A5
..SETS B2 "DASH"
..SETS B3 "DASP"
.GOTO 1B
..ASK D 6.4KHZ RAMP
..IFF .GOTO 1B
.WAIT FOR
FOR DARM6=DRAMP
.SB:  .IFT A2 .GOTO 91
.SET A2
.WAIT MAC
MAC DA64=DA64
.SB:  .SETS B1 "DARM64"
.IFT A7 .GOTO 288
.SET A7
.SETS B2 "DA64"
.SETS B3 "DARAMP"
.GOTO 188
.SB:  .ASK E 8.XHZ RAMP
.IFF E .GOTO 388
.IFT A6 .GOTO 81
.SET A6
.WAIT FOR
FOR DARM6=DRAMP
.SB:  .IFT A4 .GOTO 82
.SET A4
.WAIT MAC
MAC DA88=DA88
.SB:  .SETS B1 "DARM88"
.IFT A8 .GOTO 288
.SET A8
.SETS B2 "DA88"
.SETS B3 "DARAMP"
.188:  .OPEN DA881.CHD
.ENABLE DATA
'B1'//PR=8='B2'.'B3'
//
.COMMON=COMMON:RW
//
.DISABLE DATA
.CLOSE
.WAIT TBK
TBK #0DA881
.288:  1
THE FOLLOWING STEPS HAVE TO BE DONE CAREFULLY.
1 IF YOU ARE READY, PLEASE TYPE 'RES ...AT.'
1 OR YOU CAN TYPE 'ABO ...AT.' TO ABORT THIS COMMAND FILE.
.PAUSE DA
INS 'B1'
FIX 'B1'
1 COMMAND WILL BE PAUSED RIGHT HERE.
1 AFTER EXECUTION OF D/A FILE, PLEASE TYPE 'RES ...AT.'
1 TO REMOVE THE TASK.
RUN 'B1'
.PAUSE DA
REM 'B1'
.388:  .ASK F DO YOU WANT TO RUN IT AGAIN
1.IFT F .GOTO 2
.SET /COMMON=COMMON
PIP DA881.CHD:"/DE
.IFF A1 .GOTO 381
.WAIT PIP
PIP DA881 OBJ:"/DE
.381:  .IFT A2 .GOTO 382
.WAIT PIP
PIP DA881 OBJ:"/DE
.382:  .IFT A3 .GOTO 383
.WAIT PIP
PIP DA881 OBJ:"/DE

END
DATE FILMED
8-80
DTIC