NASA-CR-203945

ORIGINAL

FINAL REPORT

NOVEMBER 1996

02/277

Conversion-Integration of MSFC Nonlinear Signal Diagnostic Analysis Algorithms for realtime Execution on MSFC's MPP Prototype System

NASA CONTRACT NO. NAS8-40341

Prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
MARSHALL SPACE FLIGHT CENTER, AL 35812

by

AI SIGNAL RESEARCH, INC.
3322 South Memorial Parkway, Suite 67
Huntsville, AL 35801
(205)880-1968

AI SIGNAL RESEARCH	NASA
Program Manager: J. Jong	Contract Monitor: T. Zoladz
	Contracting Officer: Patricia Fundum

TABLE OF CONTENTS

FOREWORD

This report was prepared by AI Signal Research, Inc. (ASRI) for the George C. Marshall Space Flight Center, National Aeronautics and Space Administration. The work was performed under contract NAS8-40341, entitled "Conversion-Integration of MSFC Nonlinear Signal Diagnostic Analysis Algorithms for realtime Execution on MSFC's MPP Prototype System " over the time period from June 7, 1995 through December 2, 1996.

The work was carried out by Dr. Jen-Yi Jong serving as Program Manager. Dr. Jong was responsible for the software development, conversion and evaluation of Nonlinear Signal Diagnostic Analysis Algorithms. Ms. Polly Lu serving as programmer was responsible for implementing the software on C40 DSP using Data Translation's DSP-EZ programming language.

Mr. T Zoladz, MSFC/ED32, and Mr. T. Fiorucci, MSFC/23, provided valuable guidance through coordination of computer resources and definition of task requirements and priority. In addition, Mr. T. Bapty at Vanderbilt has provided valuable assistance in establishing the handshake for the conversion from ASRI's DSP algorithms to Vanderbilt's MGA Model Based Programming environment

1.0 PROGRAM SUMMARY

NASA's advanced propulsion system SSME/ATD has been undergoing extensive flight certification and developmental testing, which involves large numbers of health monitoring measurements. To enhance engine safety and reliability, detailed analysis and evaluation of the measurement signals are mandatory to assess its dynamic characteristics and operational condition. Efficient and reliable signal detection techniques will reduce the risk of catastrophic system failures and expedite the evaluation of both flight and ground test data, and thereby reduce launch turn-around time. During the development of SSME, ASRI participated in the research and development of several advanced non-linear signal diagnostic methods for health monitoring and failure prediction in turbomachinery components. However, due to the intensive computational requirement associated with such advanced analysis tasks, current SSME dynamic data analysis and diagnostic evaluation is performed off-line following flight or ground test with a typical diagnostic turnaround time of one to two days. The objective of MSFC's MPP Prototype System is to eliminate such "diagnostic lag time" by achieving signal processing and analysis in real-time. Such an on-line diagnostic system can provide sufficient lead time to initiate corrective action and also to enable efficient scheduling of inspection, maintenance and repair activities.

The major objective of this project was to convert and implement a number of advanced nonlinear diagnostic DSP algorithms in a format consistent with that required for integration into the Vanderbilt Multigraph Architecture (MGA) Model Based Programming environment. This effort will allow the real-time execution of these algorithms using the MSFC MPP Prototype System. ASRI has completed the software conversion and integration of a sequence of nonlinear signal analysis techniques specified in the SOW for real-time execution on MSFC's MPP Prototype. This report documents and summarizes the results of the contract tasks; provides the complete computer source code; including all FORTRAN/C Utilities; and all other utilities/supporting software libraries that are required for operation.

2.0 TECHNICAL DISCUSSION

2.1 Introduction

Machinery fault detection and diagnosis have always been significant technical challenges in the aeronautics and transportation industries. Yet they remain a necessity since a reliable health monitoring system can prevent catastrophic failures and costly unscheduled down time due to false alarms. As computer information processing technology continues to advance, the major challenge associated with machinery monitoring and diagnosis is shifting from how to obtain machinery vibration data to methods of information extraction and interpretation. Therefore, the incorporation of advanced signal processing element into a health monitoring device has become invaluable. However, due to the intensive computational requirement associated with such advanced analysis tasks, current SSME dynamic data analysis and diagnostic evaluation is performed off-line following flight or ground test with a typical diagnostic turnaround time of one to two days. The objective of MSFC's MPP Prototype System is to eliminate such "diagnostic lag time" by achieving signal processing and analysis in real-time. Such an on-line diagnostic system can provide sufficient lead time to initiate corrective action and also to enable efficient scheduling of inspection, maintenance and repair activities.

2.2. Signal Analysis Techniques

Over the past decade, significant progress has been made in the aerospace, transportation and industrial communities to enhance performance of machinery failure detection and diagnosis through advances in instrumentation, modeling, and signal analysis techniques. ASRI has participated in the development of a hierarchy of advanced nonlinear diagnostic signal analysis techniques for turbomachinery system. These techniques can extract and identify intelligent information hidden in a measurement signal which is often unidentifiable using conventional signal analysis methods. By providing additional insight into the system response, these analysis tools can better identify well-hidden defect symptoms as well as false-alarm signatures. As a result, false-alarm/mis-interpretation rates are reduced and greatly improve system reliability. These techniques have been applied to day-to-day Space Shuttle Main Engine (SSME) hot-firing test/flight data, and appear to be highly promising for turbomachinery health monitoring and failure diagnostics.

In the following subsections, the technical discussion of several nonlinear diagnostic signal analysis to be converted/integrated into MSFC MPP prototype system will be discussed. These techniques include:

1. TOPO: Topographical Joint Time-Frequency Mapping Algorithm
2. Hyper-Coherence Analysis - Nonlinear correlation characteristics between integer spectral components

3. Bi-Spectral Analysis -Nonlinear correlation characteristics between three spectral components with frequency sum and difference
4. Tri-Spectral Analysis - higher order coherence estimations among four spectral components and modulation sideband
5. Envelope Detection Method - Demodulation of envelope signal using Hilbert Transform
6. GHC - Generalized Hyper-Coherence using instantaneous phase information

Software implementation of these techniques has been performed on a TI-C40 DSP, which are able to run in real-time on ASRI's C40 DSP board and display the results on a PC computer system. Since the Multigraph Architecture (MGA) is not available at ASRI, a handshake point and format for the conversion from ASRI's DSP algorithms to Vanderbilt MGA Model Based Programming environment must be utilized. The algorithm and codes of the programs documented in this report utilize such a conversion format in order to reach the handshake point acceptable by Vanderbilt.

2.2.1 TOPO: Topographical Joint Time-Frequency Mapping Algorithm

2.2.1.1 General background, algorithm and simulation example TOPO

The Topo spectral time/frequency display algorithm has been implemented on a TI-C40 DSP running in real-time and display Topo on a PC computer system. This format has been proved to be acceptable by Vanderbilt as a feasible handshake point during the conversion of the bi-coherence program. This format utilizes both Data Translation's DSP-EZ language to program signal processing application directly on a TI C40 DSP board, and the Visual Basic language to program the tasks on PC computer. . Detailed information of this Topo program is discussed below:

The 3-D PSD isoplot or waterfall plot (as shown in figure 1.1) has been used extensively for vibration signal analysis for engine diagnostics. It can display the spectral information in a joint frequency, time and amplitude format. This format is especially useful for nonstationary dynamic signal to allow clear visual inspection and identification of the trace of any particular spectral component as a function of time and its relationship with respect to some other frequency components. However, there exist several drawback about this format. First of all, a threshold level must be chosen, so that only the spectral components above the threshold level will be plotted. While the rest peaks below that level will be ignored (as shown in Figure 1.2-a). However, for machinery diagnostic application, these low amplitude spectral components could still represent some important defect signature and should not be left out. secondly, if the threshold level is set too low in order to include the low-amplitude PSD peaks, then the fluctuating noise floor with larger amplitude will smear the isoplot in its corresponding frequency region (as shown in Figure 1.2-b). Thirdly, for a nonstationary data, when there are too many spectral peaks present and cross-over occur among different spectral components, then it would become very difficult to identify the trace of those spectral components in the isoplot.

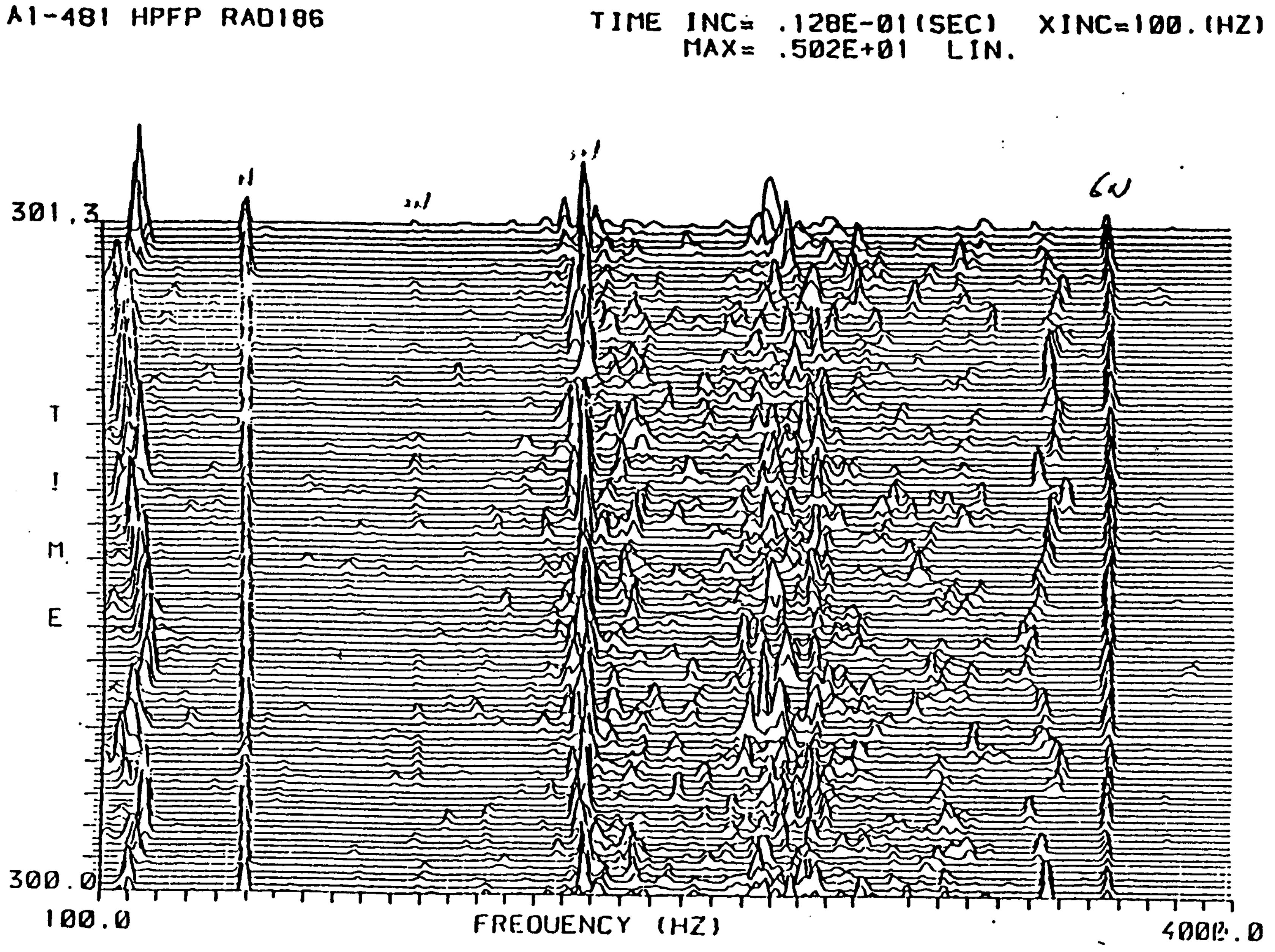
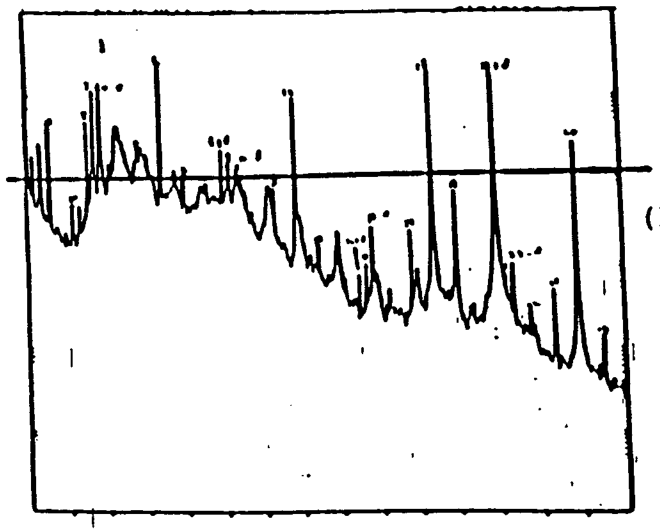

A1-481 HPFP RAD186
TIME INC= .128E-01(SEC) XINC=100.(HZ)
MAX= .502E+01 LIN.
1N
2N
3N
6N
301.3
T
I
M
E
300.0
100.0
FREQUENCY (HZ)
4000.0

Figure 1.1

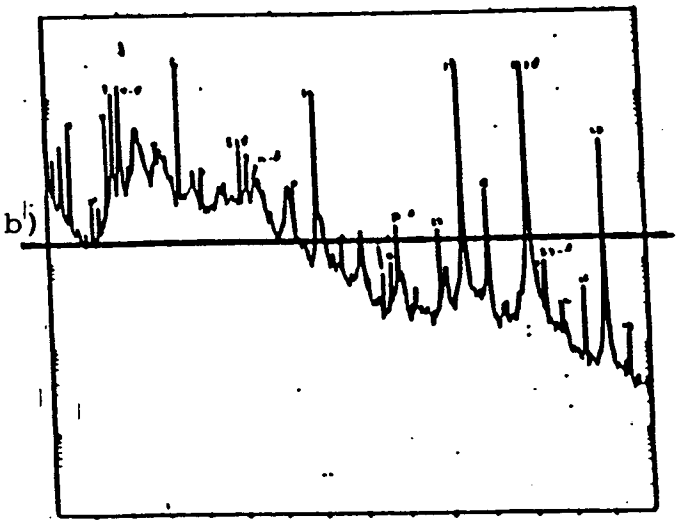
- SHRESHOLD LEVEL TOO HIGH--- Important Low Amplitude Defect Signature Will Be Ignored.

- SHRESHOLD LEVEL TOO LOW --- The Fluctuating Noise Floor Will Smear The Entire Isoplot

(a)

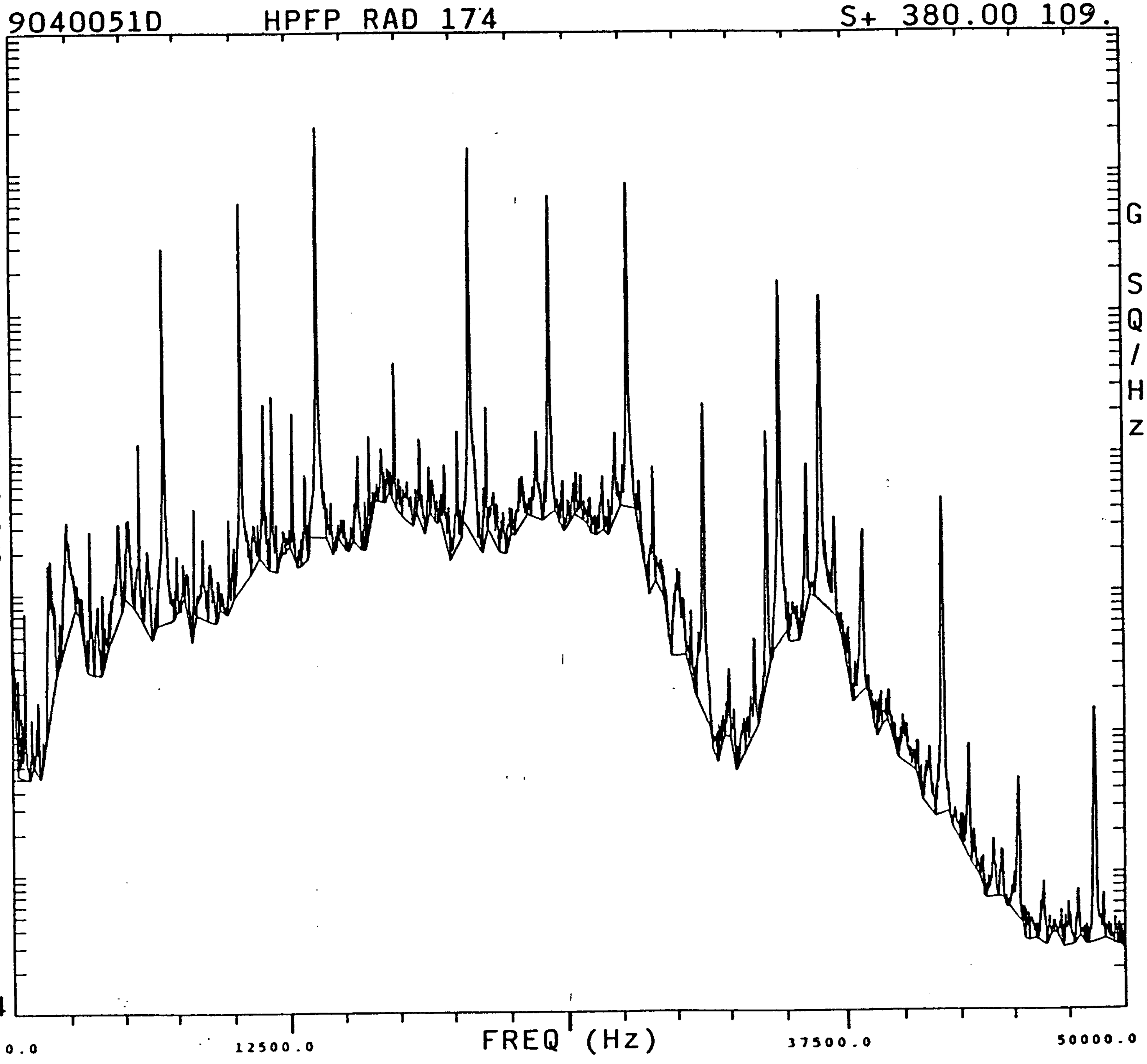


(b)



- NON-STATIONARY DATA --- Many PSD Peaks Get Compressed Too Closely Together & Cross-over Occur Among Different Spectral Components.

Figure 1.2



13891.6	214.615
20849.6	152.573
27807.6	84.772
24316.4	69.221
10424.8	62.568
6958.0	29.864
34741.2	16.908
36499.0	13.260
11840.8	2.678
31274.4	2.308
34155.3	1.463
5786.1	1.237
41699.2	0.475
585.9	0.075
48657.2	0.015

```
#AVGS=  50
   BW=  24.41
 COMP= 207.176
01/06/91
ED23
```

8
Figure 1.3

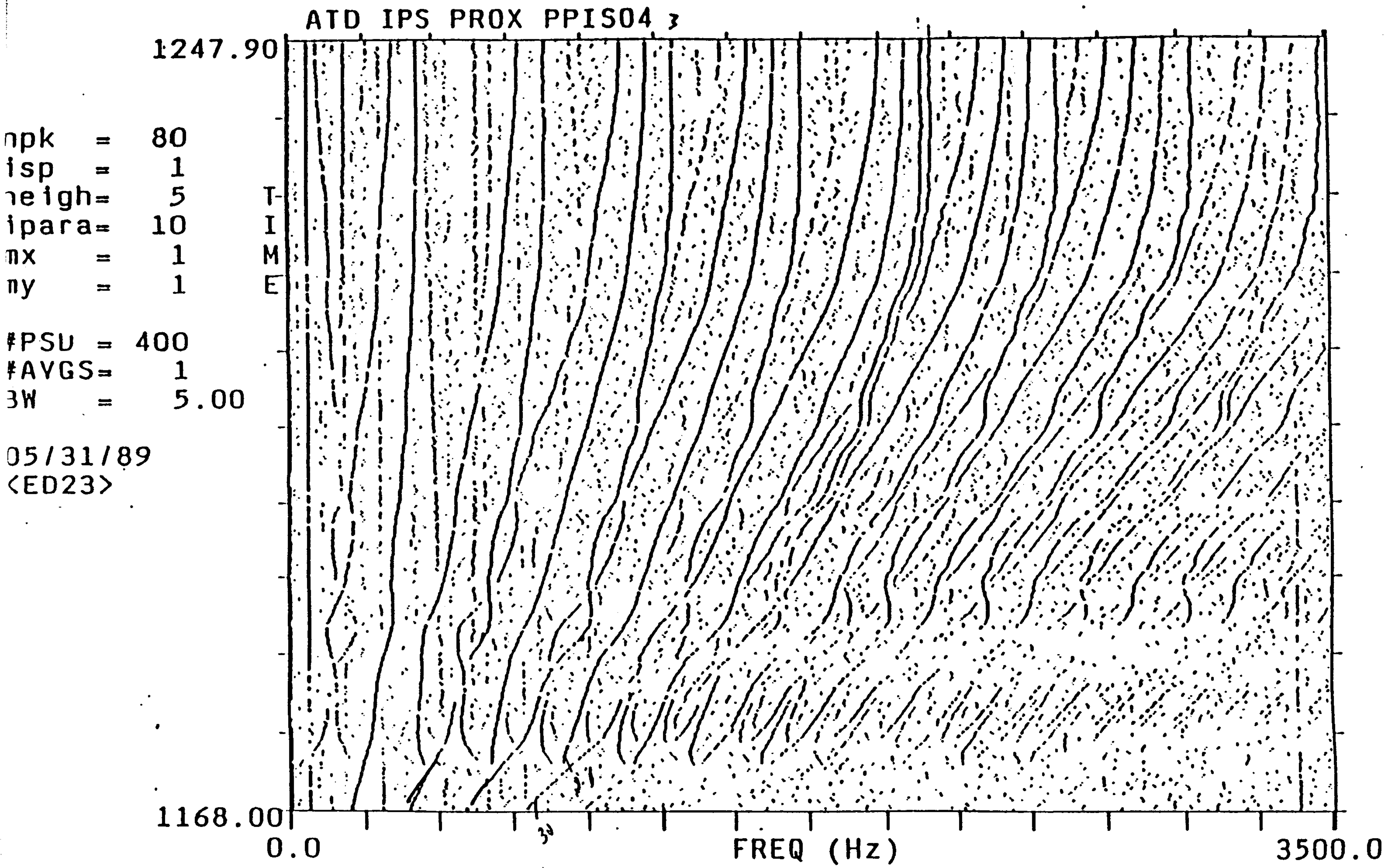


FIGURE 1.4 TOPO PLOT OF BEARING DISPLACEMENT DURING SPEED CHANGE

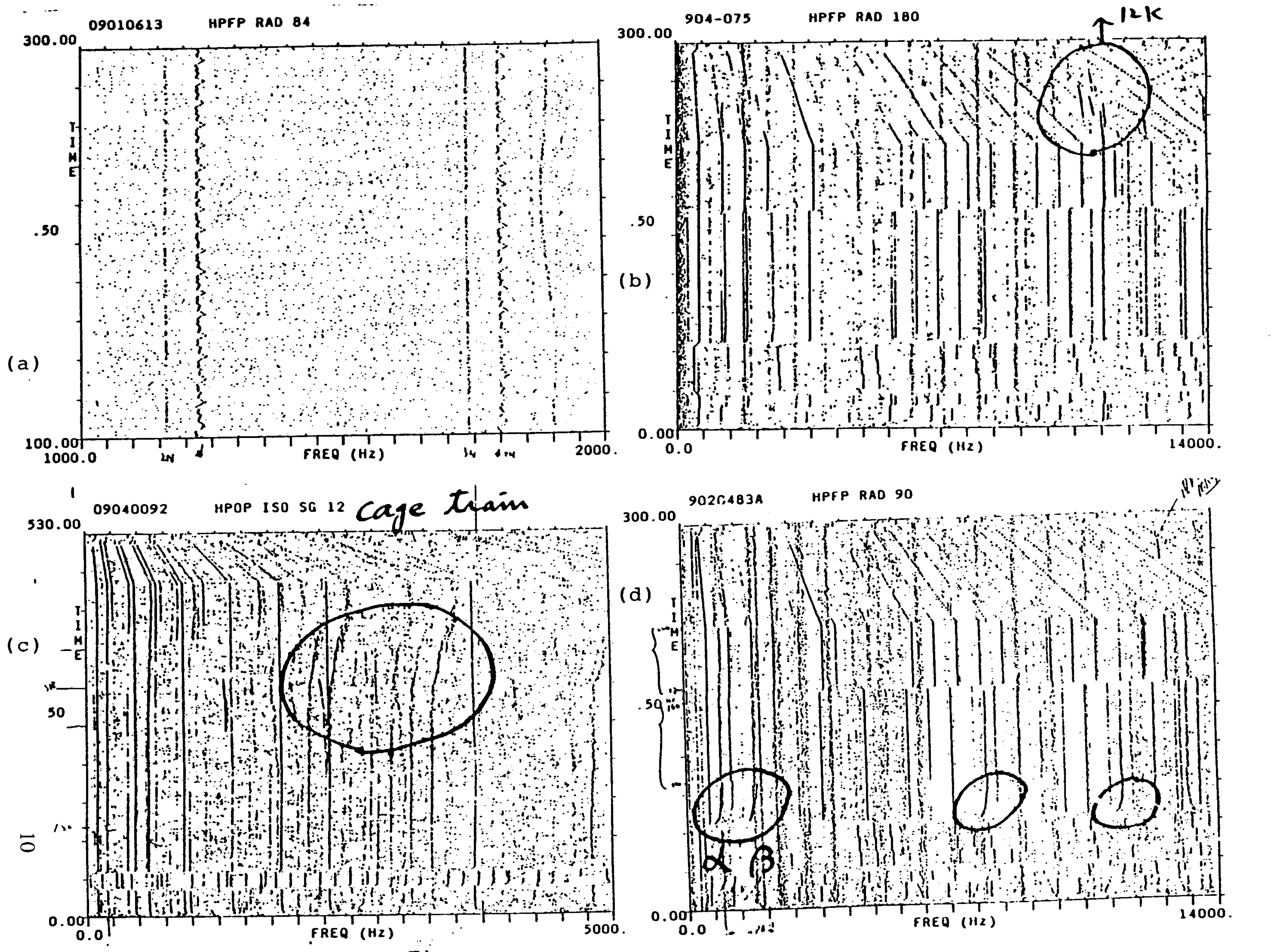

(a)
09010613 HPFP RAD 84
300.00
TIME
.50
100.00
1000.0
FREQ (Hz)
2000.
(b)
904-075 HPFP RAD 180
12K
300.00
TIME
.50
0.00
0.0
FREQ (Hz)
14000.
(c)
09040092 HPOP ISO SG 12
cage train
530.00
TIME
50
0.00
0.0
FREQ (Hz)
5000.
(d)
902C483A HPFP RAD 90
300.00
TIME
.50
0.00
0.0
FREQ (Hz)
14000.
α
β

Figure 1.5

To overcome these limitations, the PSD to be display must be Normalized so that the noise floor variation would not smear the isoplot. This can be achieved with the TOPO algorithm by first identifying the noise floor of the PSD which is shown by the solid line near the bottom of the PSD in figure 1.3. The original PSD is then Normalized by removing the noise floor from the raw PSD so that all the spectral peaks will have the same "ground" level. From this normalized PSD, all the spectral peaks can be better identified without the interference of noise floor variation. This noise floor estimation is performed using a "Rainfall" algorithm to be discussed in detail in this report.

Figure 1.4 shows a typical Topo plot of a SSME vibration measurement signal during engine start-up. In this Topo plot, 400 PSD are plotted as a function of frequency and time. Each line or dot in this plot represent a PSD peak location. The width of each line is proportional to the signal-to-noise ration (SNR is defined as the amplitude of spectral peak above noise floor in dB) of each PSD peak. During this non-stationary operation, detailed spectral information and cross-over between different component can be clearly visualized and identified which would be difficult from an isoplot. The Topo Algorithm can transform a large amount of dynamic signal into a simple image-like pattern, which would allow an easy identification of any special pattern associated with certain particular phenomenon. The examples in figure 1.5 show 4 different SSME dynamics pattern. Figure 1.5-a shows the Zipper frequency component with a unique periodic frequency variation phenomenon. Figure 1.5-b shows 12KHZ anomalous phenomena with a group of spectral peaks not tracking the Sync and Sync harmonics during engine shut-down. Figure 1.5-c shows the development trend of the modulation/sideband pattern associated with bearing defect. And figure 1.5-d shows the subtle difference of frequency variation during PWL change between a rotor related components and a fluid related component.

2.2.1.2 RainFall Algorithm for Noise floor Estimation.

The rainfall algorithm forms the basis for noise floor estimation in generating major spectral peaks along with their signal-to-noise ration (SNR) and bandwidth (BW). The following steps demonstrate the Rainfall algorithm for noise floor estimation.

1. The deep valleys are first removed and smoothed (interpolated) with the two neighboring points.

2. All PSD peaks are first identified and counted.

3. The left and right roots of each peak are also identified based on the rainfall trace shown in figure 6-a.
4. Remove Small Peaks by Applying Rainfall roll-over as shown in figure 6-b for any peak amplitude less than the user-specified threshold.
5. Consolidate peaks by merging smaller peaks been rolled-over with larger peaks not been rolled-over. Assign new left and right roots after peak consolidation.
6. calculate noise floor by interpolating all settle-down rainfall drops

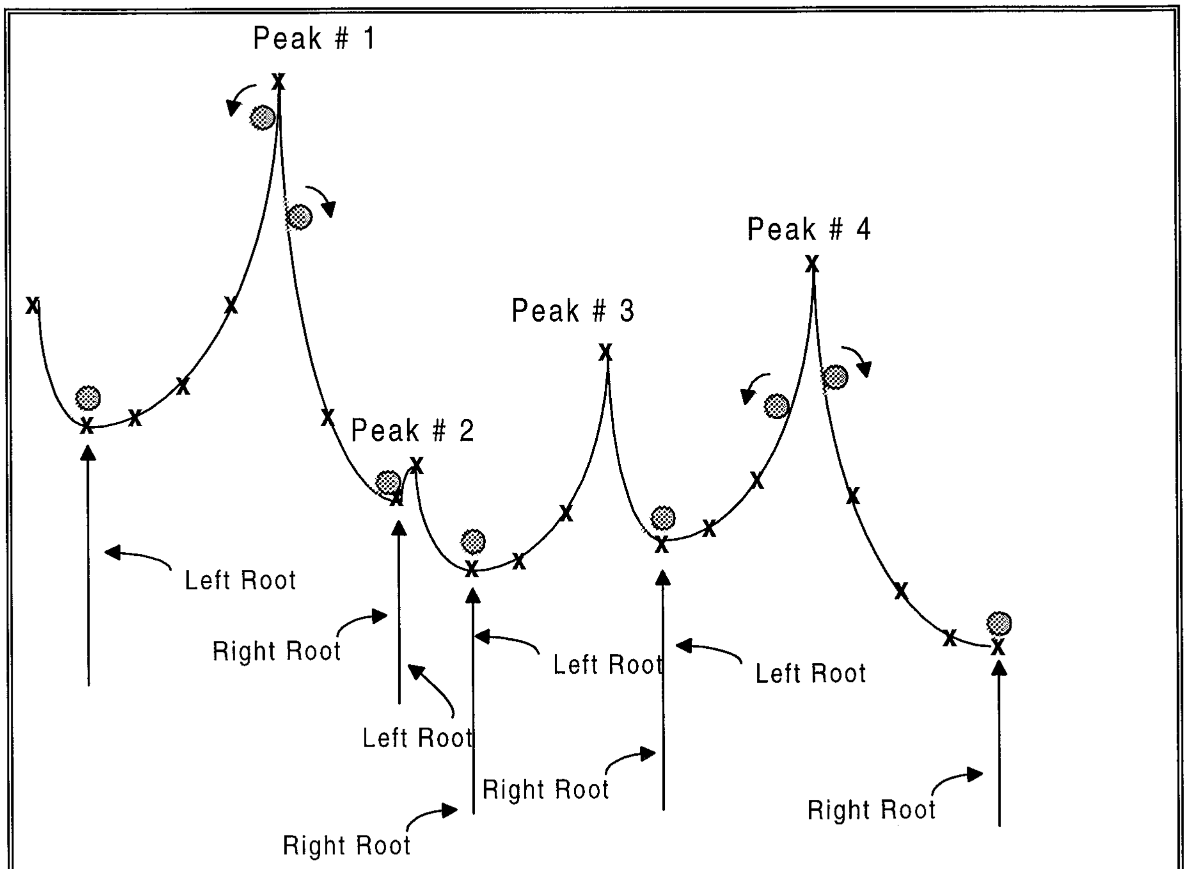


Figure 1.6-a

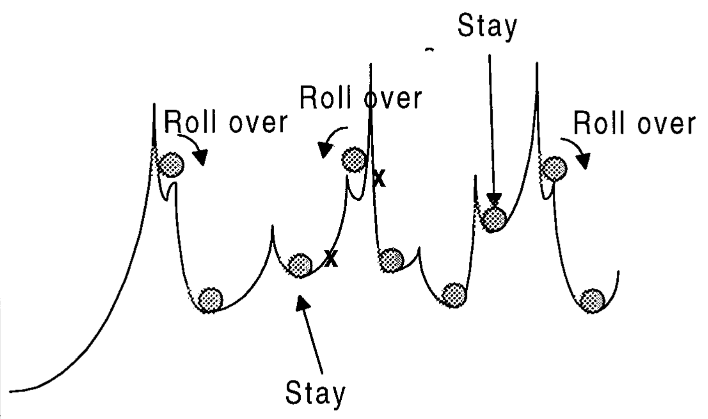


Figure 1.6-b

2.2.1.3 DSP Code Overview of TOPO using Data Translation's DSP-EZ programming language running on C40 DSP.

The Topo algorithm has been implemented on ASRI's TI-C40 DSP, and has been preliminary tested using some "simple" input analog wave from a signal generator. The complete DSP Codes are listed below:

%%

PART1: Function ApplyRainFall is implemented in DSP-EZ side.
It takes PSD, rainfall parameter and number of PSD elements as arguments and returns the Noise Floor data and two arrays of data storing all the peaks and indices of the peaks in the array.

```
'                         ********************************
'                         **       Function ApplyRainFall   **
'                         **                                **
'                         ** Global: PkLoc( ) and PkAmp( )  **
'                         ** Return: Noise Floor            **
'                         ********************************
'

Function ApplyRainFall(A As FloatArray,  RainFall As Integer, n As Integer) As FloatArray
Dim PSDG As FloatArray
Dim iLeft As FloatArray
Dim iRight As FloatArray
Dim iFlag As FloatArray
Dim Floor As FloatArray
Dim Ad As FloatArray
Dim  nPeak, i,j, L, R,  iNext, iL, iR, kPeak As Integer
Dim AmpR, AmpL, aL, aR, delta, root  As Single
Dim iLeft0 As FloatArray

        PSDG = GenLine(0,0, n+1)                    'allocate memories for array elements
        iLeft = GenLine(0,0,n+1)
        iRight = GenLine(0,0,n+1)
        iFlag = GenLine(0,0,n+1)
        PkLoc= GenLine(0,0,n+1)
        PkAmp = GenLine(0,0,n+1)
        Floor = GenLine(0,0,n+1)
        iLeft0 = GenLine(0,0,n+1)

        For i = 0 To n-1
                PSDG(i+1) = A(i)
        Next i

        For i = 2 To n-1
                If PSDG(i) <= PSDG(i-1) And PSDG(i) <= PSDG(i+1) Then
                        PSDG(i) = 0.5 * (PSDG(i-1) + PSDG(i+1))
                End If
        Next i

        PSDG(1) = PSDG(2) + Abs(PSDG(2)) * 0.001
PSDG(n) = PSDG(n-1) + Abs(PSDG(n-1)) * 0.001

npeak = 0
i = 2
Do While ( i <= n-1 )
        If PSDG(i) > PSDG(i-1) And PSDG(i) > PSDG(i+1) Then
                npeak = npeak+1
                L = i-1
                Do While ( L > 1  And  PSDG( L-1) <= PSDG(L))
                        L = L-1
                Loop
                R = i
                Do While ( R < n  And PSDG( R+1) <= PSDG(R) )
                        R = R+1
                Loop
                iLeft (nPeak) = L
                iRight(nPeak) = R
                PkLoc(nPeak) = i
                iFlag(nPeak) = 1
                i = R
        End If
        i = i + 1
Loop

For i = 2 To npeak -1
        If  iFlag(i) =1  Then
                AmpL = PSDG(PkLoc(i))  - PSDG(iRight(i) )
                AmpR = PSDG(PkLoc(i+1))  - PSDG(iLeft(i+1) )
                If ( AmpR > AmpL) And ( AmpL <= RainFall)  Then
                        iFlag(i) = -9                   'Left merge
                End If
                If ( AmpL > AmpR)  And (AmpR <= RainFall)  Then
                        iFlag( i+1) =  9                'Right merge
                End If
        End If
Next i
iFlag(1) = 1
iFlag(npeak) =1
iFlag(npeak +1) =1
iLeft(nPeak + 1) = iRight( nPeak)
kPeak = 0
For i = 1 To nPeak
        If iFlag(i) = 1 Then
                kPeak = kPeak+1
                iL =i
                If i  > 1 Then
                        iL = iL - 1
                        Do While ( iL > 1 And  iFlag(iL) =  -9)
                                iL = iL -1
                        Loop
                        iL = iL + 1
                End If
                iR = i
                If i  < nPeak Then
                        iR = iR + 1
                        Do While ( iR < nPeak And  iFlag(iR ) = 9)
                                    iR = iR + 1
                        Loop
                        iR = iR - 1
                End If

                PkLoc(kPeak) = PkLoc(i)
                iLeft0(kPeak) = iLeft(iL)
                iRight(kPeak) = iRight(iR)
                iL = iLeft0(kPeak)
                iR = iRight(kPeak)
                aL= PSDG(iL)
                aR = PSDG(iR)
                delta = (aR-aL)/(iR-iL)
                root = aL+ (PkLoc(kPeak)  - iL) * delta
                PkAmp(kPeak) = PSDG(PkLoc(kPeak) ) - root
        End If
    Next i

    For i = 1 To kPeak
        iLeft(i) = iLeft0(i)
    Next i

    For i = 1 To  kPeak
        iL = iLeft(i)
        iR = iRight(i)
        aL= PSDG(iL)
        aR = PSDG(iR)
        delta = (aR - aL)/(iR - iL)
        For  j = iL To iR
            Floor(j) = aL + (j -iL) * delta
        Next j
    Next i
    For i = 1 To iLeft(1)
        Floor(i) = Floor(iLeft(1))
    Next i

    For i = iRight(kPeak) To n
        Floor(i) = Floor(iRight(kPeak))
    Next i
    PkLoc=Extract(PkLoc, kPeak, 1)          'Global for passing to VB
    PkAmp=Extract(PkAmp, kPeak, 1)          'Global for passing to VB
    ApplyRainFall =Extract(Floor, n, 1)             'Function Returns Noise Floor

End Function
```

%%

PART2: In Visual Basic Host side, two arrays of data, PkLoc() and PkAmp() are passed from DSP-EZ, and then Functions FindPeak and SortMax are called to find the cutoff value so to obtain the TOPO data for plotting.

```
'       *****************************************************
'       **    Procedure SortMax                            **
'       **                                                 **
'        ** Returned Twk() is sorted PkVal() in descending order          **
'        ********************************************************
'
'
Sub SortMax (PkVal() As Single, Twk() As Single, m As Integer)
Dim i, k, mm, z    As Integer
Dim T1() As Integer
Dim t2() As Single
Dim iWk() As Integer
ReDim T1(m)
ReDim t2(m)
ReDim iWk(m)

  For i = 0 To m - 1
    iWk(i) = 0
    Twk(i) = -1000
  Next i
  mm = m - 1
  For i = 0 To m - 1
    z = mm
    For k = 0 To m - 1
      If PkVal(i) >= Twk(k) Then
        z = k
        Exit For
      End If
    Next k
    For k = z To m - 2
      T1(k + 1) = iWk(k)
      t2(k + 1) = Twk(k)
    Next k
    For k = z + 1 To m - 1
      iWk(k) = T1(k)
      Twk(k) = t2(k)
    Next k
    iWk(z) = i
    Twk(z) = PkVal(i)
  Next i

End Sub

'        ******************************************
'        **     Function FindPeak                **
'        **                                      **
'        ** Call  : SortMax and obtain Twk()     **
'        ** Return: Peak Value(single)           **
'        ******************************************
'
'
Function FindPeak (A() As Single, ChoicePK As Integer, Neigh As Integer, n As Long) As Single
Dim NumPk As Integer
Dim Nex, i, j, MinPk  As Integer

Dim PeakLoc() As Integer
Dim Peakamp() As Single
Dim Twk() As Single
```

```
ReDim PeakLoc(n)
ReDim Peakamp(n)
ReDim Twk(n)


  NumPk = 0
  For i = 2 To n - 2
    If (A(i) > A(i - 1)) And (A(i) > A(i + 1)) Then
      Nex = i + 2
      NumPk = 1
      PeakLoc(0) = i
      Peakamp(0) = A(i)
      Exit For
    End If
  Next i
  For i = Nex To n - 2
    If A(i) > A(i - 1) And A(i) > A(i + 1) Then
      If (i < PeakLoc(NumPk - 1) + Neigh) Then
        PeakLoc(NumPk - 1) = i
        Peakamp(NumPk - 1) = A(i)
      Else
        NumPk = NumPk + 1
        PeakLoc(NumPk - 1) = i
        Peakamp(NumPk - 1) = A(i)
      End If
    End If
  Next i

  SortMax Peakamp(), Twk(), NumPk
  If ChoicePK >= NumPk Then
    MinPk = NumPk
  Else
    MinPk = ChoicePK
  End If
  FindPeak = Twk(MinPk - 1)

End Function


'**********************************************
'**       Host obtains array data from DSP-EZ .       **
'**********************************************

If CommChannel = TopoChan Then
  If bufferreadyfromdsp(frmDemo.DspEz1, TopoChan) Then
      GetBufferFromDsp frmDemo.DspEz1, TopoChan, Databuffer(), validpts
       Plotbuffer plot5, Databuffer(), validpts
       lblFilYmax.Caption = plot5.yMax
       lblFilYmin.Caption = plot5.yMin
  End If
End If
If CommChannel = LocChan Then
  GetBufferFromDsp DspEz1, CommChannel, Databuffer(), validpts
  For i = 0 To validpts - 1
    PkLoc(i) = Databuffer(i)
  Next i
End If
If CommChannel = AmpChan Then
  GetBufferFromDsp DspEz1, CommChannel, Databuffer(), validpts
  For i = 0 To validpts - 1
    PkAmp(i) = Databuffer(i)
  Next i


  If validpts > 20 Then
    CutOff = FindPeak(Databuffer(), 20, 5, validpts)
  End If
  If CutOff < 5 Then
    LowVal = 5
  Else
    LowVal = CutOff
  End If
  HighVal = 1000 - LowVal
  frmTopo!txtCutOff.Text = Str(CutOff)

  For i = 0 To validpts - 1
    If PkAmp(i) >= CutOff Then
      Sx = PkAmp(i) / 60
      If Sx > 1 Then
        Sx = 1
      End If
      Ex = Sx * Broad + 4 * PkLoc(i)
      TopoData(i) = LowVal
      frmTopo!Picture1.Line (4 * PkLoc(i), TopoData(i))-(Ex, TopoData(i))
    End If
  Next i
    r1% = BitBlt(ByVal frmTopo.Picture1.hDC, ByVal HDisp, ByVal 0, ByVal frmTopo.Picture1.Width
- HDisp, ByVal frmTopo.Picture1.Height - VDisp, ByVal frmTopo.hDC, ByVal frmTopo.Picture1.Left,
ByVal frmTopo.Picture1.Top + VDisp, ByVal SRCCopy)
    r1% = BitBlt(ByVal frmTopo.Picture1.hDC, ByVal frmTopo.Picture1.Left, ByVal
frmTopo.Picture1.Height - VDisp, ByVal frmTopo.Picture1.Width, ByVal VDisp, ByVal frmTopo.hDC,
ByVal 0, ByVal 0, ByVal Whiteness)
End If
```

2.2.2 Hyper-Coherence Analysis - Nonlinear correlation characteristics between integer spectral components

2.2.2.1 General background, algorithm and simulation example of hyper-coherence function

In machinery health monitoring, the shaft synchronous (RPM) component and its harmonics in a monitoring signal contains much subtle information about the system operating condition. Many mechanical failure modes, such as shaft misalignment, loose coupling, or rubbing, will cause waveform distortion resulting in the generation of strong harmonics. For example, the radial rubbing of a rotating shaft will cause waveform clipping. This waveform clipping will then generate strong odd synchronous harmonics which can be easily identified from its PSD. However, due to a complex operational environment, many other independent sources may coincidentally generate spectral components coincident with harmonic frequencies appearing to be synchronous harmonics. With this potential for ambiguity, a diagnostic method which can identify whether an apparent harmonic is a true harmonic is indispensable. The hyper-coherence method developed by ASRI provides such a nonlinear technique for harmonic identification. Hyper-coherence represents a hierarchy of higher joint moments between a reference spectral component and any of its harmonics. It can identify when a spectral component at a harmonic frequency is truly a harmonic or not. This is achieved by measuring a special phase difference to identify the degree of correlation between the fundamental and a harmonic.

To describe the nonlinear interaction between harmonically related spectral components in a given stationary, zero mean signal, we define the Hyper-spectrum of order n by the relation

$$H(n; f1) = E[X^n(f1)\, X^*(nf1)], \quad n = 1,2,3,.. \tag{2.1}$$

where f1 is an arbitrary reference frequency, and nf1 is an integer multiple of f1. Thus, the ascending terms in H(n; f1) represents a single value from the linear spectrum, bi-spectrum, tri-spectrum, etc., at the specific value f1= f2= ...= fn.

In analogy with the ordinary coherence function, the hyper-coherence can be defined as a normalized hyper-spectrum:

$$G2(n; f1) = \frac{|E[X^n(f1)\, X^*(nf1)]|^2}{E[|X^n(f1)|^2 \;\; E[|X(nf1)|^2])} \qquad n=1,2,3.. \tag{2.2}$$

The hyper-coherence function defines the nonlinear correlation between a reference frequency component in a vibration signal and its harmonics. A major benefit is determination of whether an apparent harmonic in a complex vibration signal is correlated with the fundamental or caused by extraneous noise. The technique has been frequently applied to space shuttle main engine turbopump measurements in order to discriminate between a true Sync harmonic and other independent components. A typical example is the discrimination between a true 3N and a so-called "pseudo-3N" components.

2.2.2.2 Test Example for hyper-coherence Analysis

An example encountered from SSME hot firing test can best demonstrate the practical application of hyper-coherence for machinery diagnostics. Figures 2.1-a shows the PSD taken from a HPFTP accelerometer during SSME test 901-436. The peak marked "N" is corresponding to the fundamental RPM component, and the remaining major peaks are its harmonics 2N, 3N and 4N. Engine failure occurred near the end of the test. Debris was found in a down stream coolant line during tear-down hardware inspection. Off-line diagnostic signal analysis was performed after the fact to investigate the failure. Anomaly was detected due to a high level amplitude of 3N and this is considered to be defect signature associated with several different failure modes.

To confirm it as a defect signature, Hyper-Coherence analysis was performed. Figure 2.1-c shows the hyper-coherence function resulting from this test. It shows the correlation between the fundamental RPM (N) and all of its harmonics including itself. High level coherence peaks indicate that 2N, 3N, and 4N are all highly correlated with the reference component N. Therefore, they are all true harmonics of the Sync frequency component, and the original high level 3N PSD of the test is a true defect signature.

A similar situation occurred later during another SSME engine test 901-471. Figure 2.1-b shows the PSD taken from a HPFTP accelerometer during this test with similar abnormally high 3N PSD amplitude. However, the hyper-coherence analysis was able to identify this apparent defect signature as a false-alarm. Figure 2.1-d shows the hyper-coherence function resulting from this test. This time, the hyper-coherence at 3N is much smaller. This indicates that the spectral peak at the 3N frequency is not a true third harmonic. The signature turned out to be the so-called pseudo 3N component which is due to some independent source in the HPFTP not related to the machinery rotational process.

It should be pointed out that even though the hyper-coherence is only defined at the harmonic frequencies of the reference frequency. However, if the frequency of a higher harmonic is greater than the nyquist frequency, it will be folded back below the nyquist frequency due to aliasing effect. In this case, the hyper-coherence can still pick correlation of this aliased harmonic despite the frequency of this aliased harmonic is no longer a integer multiple of the reference frequency. For this reason, the hyper-coherence is still computed for all frequencies other than the harmonic frequencies. However, the

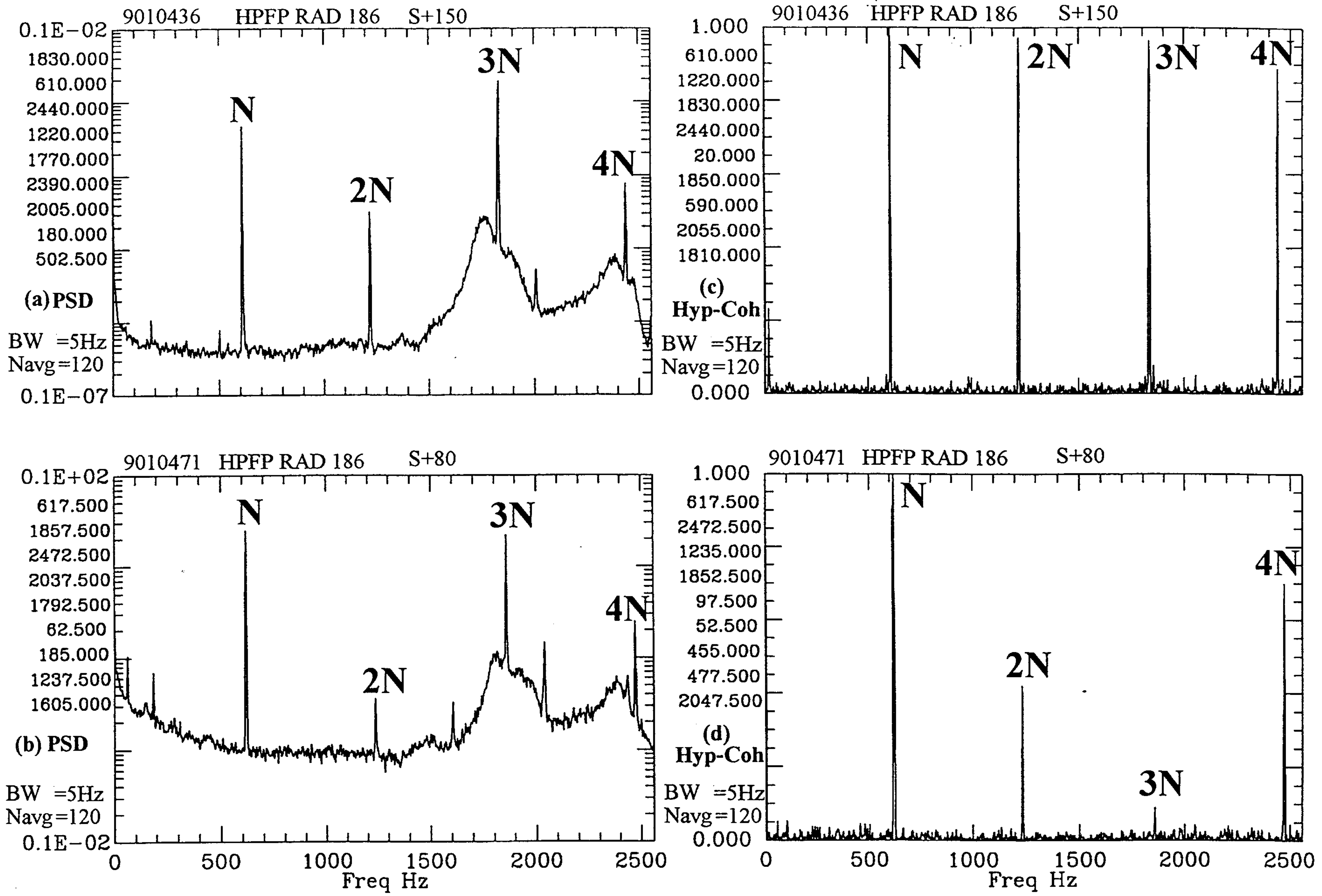


Figure 2.1 Hyper-Coherence Analysis for SSME Tests 9010436 & 9010471
(a) & (b) PSD For tests 9010436 & 9010471
(c) & (d) Hyper-coherence for tests 9010436 & 9010471

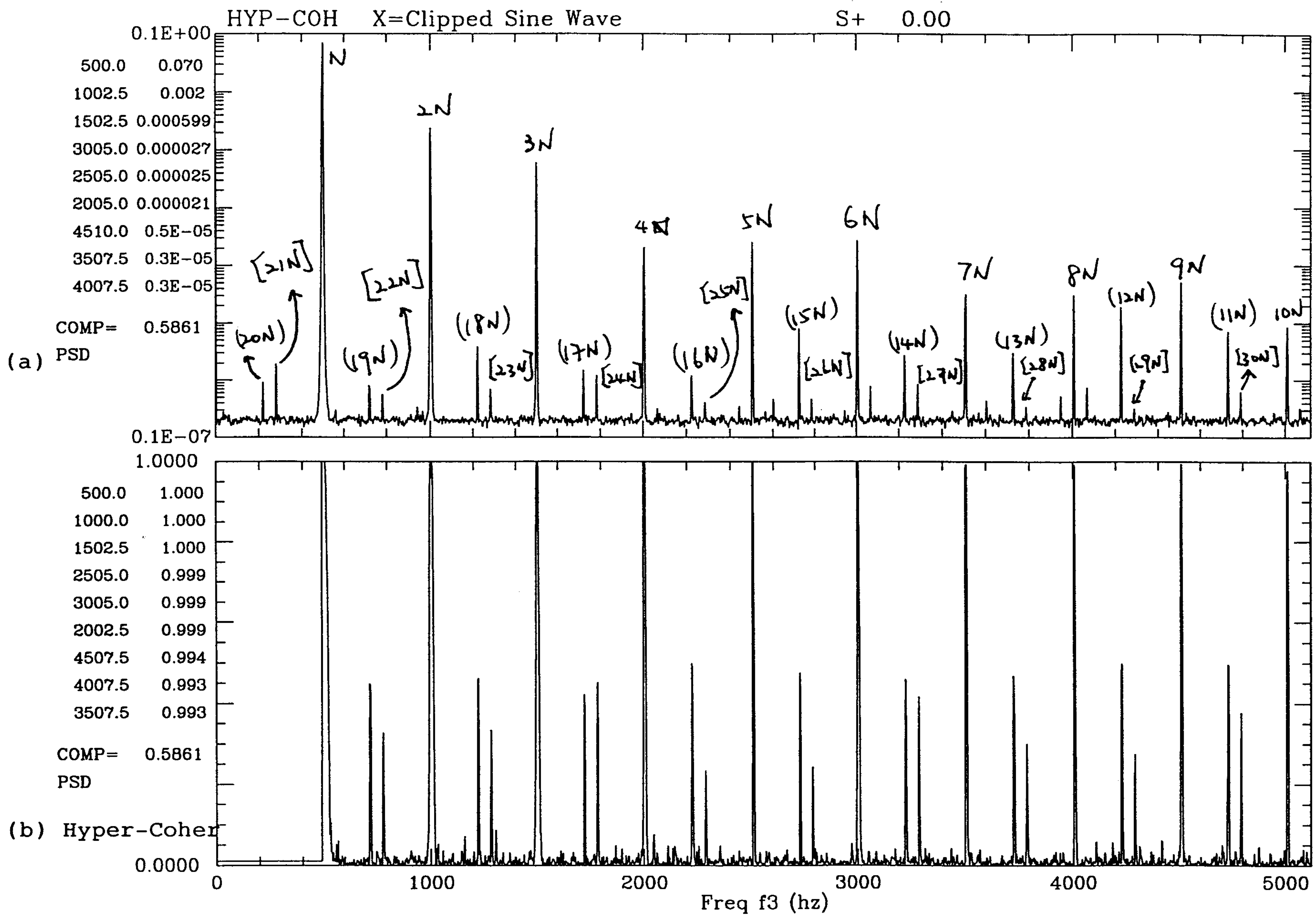


Figure 2.2 HYPER-COHERENCE Hc(f), (Ref Freq= 500.00) of clipped Sine Wave

hyper-coherence below the reference frequency is not computed. This is because the phase of the reference term, X^n, in equation (1) becomes a small number when n becomes smaller than 1, which would appear to be a constant phase. Figure 2.2 shows a simulation example for aliased harmonics. The simulation signal is a sine wave at 500 Hz. The amplitude of the sine wave is truncated at one side in order to generated its harmonics. Figure 2.2(a) shows the PSD of the simulation signal. The reference component (N) along with its harmonics, 2N, 3N,....upto to 10N, are non-aliased harmonics. The peaks marked (11N), (12N), upto (20N) are aliased harmonics. The peaks marked [21N], [22N], upto [30N] are double aliased harmonics. Figure 2.2(b) shows its hyper-coherence function with 500 Hz as the reference frequency. The hyper-coherence pick strong correlation for all the non-aliased harmonics (1N to 10N), and moderate correlation for aliased (and doubled aliased) harmonics (11N to 30N).

2.2.2.3 DSP Code Overview of hyper-coherence function using Data Translation's DSP-EZ programming language running on C40 DSP.

The hyper-coherence algorithm has been implemented on ASRI's TI-C40 DSP, using Data Translation's DSP-EZ programming language. This section lists the complete DSP Codes of it. The DSP-EZ utilizes a series of well-defined functions to construct a signal processing application directly on DSP. Therefore, the processing algorithm/flow-chart of any particular application program can be clearly defined. This format has proved to be acceptable as conversion format..

```
'                 ***********************************
'                 ***  Function Hyper Coherence   ***
'                 ***********************************
Function HyperCoherence( i1 As Integer,m1 As Integer)
Dim BHWnd As FloatArray
Dim WndData0 As FloatArray
Dim Tempbuff As FloatArray
Dim Fs As Single
Dim  kref  As Integer
Dim  kmax As Integer
Dim kmin As Integer
Dim bw As Single
Dim R1 As Single
Dim R2 As Single
Dim R3 As Single
Dim R4 As Single
Dim J1 As Complex
Dim J2 As Complex
Dim J3 As Complex
Dim d, j, i, m  As Integer
Dim dBScale As Single
Dim Power As Single
'+--------------------------------+
'| generate Window Data |
'+--------------------------------+
dbScale= -20 * log10(10*FFT_SIZE) + 6.0
BHWnd = GenWindow(BLACKMAN_HARRIS_61,BUFFSIZE)
'+------------------------------------------------+
'| acquire parameters from  buffer
'+------------------------------------------------+
Fs=Frequency
bw = Fs / BUFFSIZE
kref = ref / bw + 0.5
kmin=kref-2
kmax = nmax
'+-----------------------------------------------------------+
'| Put FFT data to host for plotting
'+-----------------------------------------------------------+
                    WndData0=Window(BHWnd,TimeData)
                 XX=fft(WndData0)
'+-------------------------------------------------------------
'         Calculate Hyper Coherence
'+-------------------------------------------------------------
                     For d=0 To Nmax -1
                                    R1=Abs(XX(d))
                      If i1=0  Then
                                          HPSD0(d)= HPSD0(d)+ R1*R1
                      ElseIf  i1=1  Then
                                          HPSD1(d)= HPSD1(d)+ R1*R1
                      ElseIf i1=2  Then
                                          HPSD2(d)= HPSD2(d)+ R1*R1
                      ElseIf i1=3  Then
                                          HPSD3(d)= HPSD3(d)+ R1*R1
                      ElseIf i1=4  Then
                                          HPSD4(d)= HPSD4(d)+ R1*R1
                      ElseIf i1=5  Then
                                          HPSD5(d)= HPSD5(d)+ R1*R1
                      ElseIf i1=6  Then
                                          HPSD6(d)= HPSD6(d)+ R1*R1
                      ElseIf i1=7  Then
                                          HPSD7(d)= HPSD7(d)+ R1*R1
                      End If
                     Next d
                    For d=kmin To  kmax-1
                              Power = d / kref
                      J1 =  XX(kref)^power
                          R1= -Imag(XX(d))
                                   R2= Real(XX(d))
                                   J2=MakeComplex(R2, R1)        'conj(XX(d))
                      If i1=0  Then
                                            HAA0(d)=HAA0(d)+J1*J2
                       ElseIf i1=1  Then
                                            HAA1(d)=HAA1(d)+J1*J2
                       ElseIf i1=2  Then
                                            HAA2(d)=HAA2(d)+J1*J2
                       ElseIf i1=3  Then
                                            HAA3(d)=HAA3(d)+J1*J2
                       ElseIf i1=4  Then
                                            HAA4(d)=HAA4(d)+J1*J2
                       ElseIf i1=5  Then
                                            HAA5(d)=HAA5(d)+J1*J2
                       ElseIf i1=6  Then
```

```
                                HAA6(d)=HAA6(d)+J1*J2
                Else
                                HAA7(d)=HAA7(d)+J1*J2
                End If

                If i1=0  Then
                                HBB0(d)=HBB0(d)+Abs(J1)*Abs(J1)
                ElseIf i1=1  Then
                                HBB1(d)=HBB1(d)+Abs(J1)*Abs(J1)
                ElseIf i1=2  Then
                                HBB2(d)=HBB2(d)+Abs(J1)*Abs(J1)
                ElseIf i1=3  Then
                                HBB3(d)=HBB3(d)+Abs(J1)*Abs(J1)
                ElseIf i1=4  Then
                                HBB4(d)=HBB4(d)+Abs(J1)*Abs(J1)
                ElseIf i1=5  Then
                                HBB5(d)=HBB5(d)+Abs(J1)*Abs(J1)
                ElseIf i1=6  Then
                                HBB6(d)=HBB6(d)+Abs(J1)*Abs(J1)
                Else
                                HBB7(d)=HBB7(d)+Abs(J1)*Abs(J1)
                End If
                            R4=Abs(XX(d))
                If i1=0  Then
                                HCC0(d)=HCC0(d)+R4*R4
                ElseIf i1=1  Then
                                HCC1(d)=HCC1(d)+R4*R4
                ElseIf i1=2  Then
                                HCC2(d)=HCC2(d)+R4*R4
                ElseIf i1=3  Then
                                HCC3(d)=HCC3(d)+R4*R4
                ElseIf i1=4  Then
                                HCC4(d)=HCC4(d)+R4*R4
                ElseIf i1=5  Then
                                HCC5(d)=HCC5(d)+R4*R4
                ElseIf i1=6  Then
                                HCC6(d)=HCC6(d)+R4*R4
                Else
                                HCC7(d)=HCC7(d)+R4*R4
                End If
            Next d
            tempbuff=AD.Scan(buffsize)
    If  (m1=NumOfAvg)  Then
          For d=0 To kmin-1
                If i1=0  Then
                      HBC0(d) = 0.001
                ElseIf i1=1 Then
                      HBC1(d)=0.001
                ElseIf i1=2 Then
                      HBC2(d)=0.001
                ElseIf i1=3 Then
                      HBC3(d)=0.001
                ElseIf i1=4 Then
                      HBC4(d)=0.001
```

```
                    ElseIf i1=5 Then
                          HBC5(d)=0.001
                    ElseIf i1=6 Then
                          HBC6(d)=0.001
                    ElseIf i1=7 Then
                          HBC7(d)=0.001
                    End If
              Next d
                   For d=kmin To kmax-1
                   If i1=0  Then
                 HBC0(d)=Abs(HAA0(d)) * Abs(HAA0(d)) /(HBB0(d)*HCC0(d))
                   ElseIf i1=1 Then
                 HBC1(d)=Abs(HAA1(d)) * Abs(HAA1(d)) /(HBB1(d)*HCC1(d))
                   ElseIf i1=2  Then
                 HBC2(d)=Abs(HAA2(d)) * Abs(HAA2(d)) /(HBB2(d)*HCC2(d))
                   ElseIf i1=3  Then
                 HBC3(d)=Abs(HAA3(d)) * Abs(HAA3(d)) /(HBB3(d)*HCC3(d))
                   ElseIf i1=4  Then
                 HBC4(d)=Abs(HAA4(d)) * Abs(HAA4(d)) /(HBB4(d)*HCC4(d))
                   ElseIf i1=5  Then
                 HBC5(d)=Abs(HAA5(d)) * Abs(HAA5(d)) /(HBB5(d)*HCC5(d))
                   ElseIf i1=6  Then
                 HBC6(d)=Abs(HAA6(d)) * Abs(HAA6(d)) /(HBB6(d)*HCC6(d))
                   Else
                 HBC7(d)=Abs(HAA7(d)) * Abs(HAA7(d)) /(HBB7(d)*HCC7(d))
                   End  If
                   Next d
              tempbuff=AD.Scan(buffsize)
            For d= 0 To Nmax-1
                   If i1=0  Then
                   HPSD0(d)=10*log10(HPSD0(d)/NumOfAvg)+ dBScale
                   ElseIf i1=1  Then
                   HPSD1(d)=10*log10(HPSD1(d)/NumOfAvg)+ dBScale
                   ElseIf i1=2  Then
                   HPSD2(d)=10*log10(HPSD2(d)/NumOfAvg)+ dBScale
                   ElseIf i1=3  Then
                   HPSD3(d)=10*log10(HPSD3(d)/NumOfAvg)+ dBScale
                   ElseIf i1=4  Then
                   HPSD4(d)=10*log10(HPSD4(d)/NumOfAvg)+ dBScale
                   ElseIf i1=5  Then
                   HPSD5(d)=10*log10(HPSD5(d)/NumOfAvg)+ dBScale
                   ElseIf i1=6  Then
                   HPSD6(d)=10*log10(HPSD6(d)/NumOfAvg)+ dBScale
                   Else
                   HPSD7(d)=10*log10(HPSD7(d)/NumOfAvg)+ dBScale
                   End If
             Next d
         End If
End  Function
```

2.2.3 Bi-Spectral Analysis -Nonlinear correlation characteristics between three spectral components with frequency sum and difference

The dynamic response of a rotating machine can exhibit frequency sum or difference phenomena. This type of nonlinear phenomena is primarily due to amplitude modulation between various rotational components and fluid/structure interactions. Amplitude modulation can be generated by at least six phenomena. Enrich and Eshleman have published works using analytical techniques to explain how these modulations are physically generated. The characteristics of such nonlinear interactions are usually reflected in a response waveform which can be identified by using the bi-spectral technique.

However, due to the lack of phase relationship information among different frequency components, conventional linear spectral analysis is not able to describe or identify such nonlinear phenomena. In linear spectral analysis, the time series of any fluctuating physical quantity is regarded as the superposition of statistically uncorrelated waves and can be described, in part, by its power spectrum which shows the power distribution in the frequency domain. However, if nonlinear interactions occur, some coherent phase relationship exists among different frequency components. As a result, the information content of harmonics and sidebands for machinery fault diagnostics is usually not fully exploited.

2.2.3.1 General background, algorithm and simulation example of bi-coherence function

As the power spectral density (PSD) function is a second moment statistic of a random signal, the auto-bi-spectrum (ABS) represents the third joint moment among three different waves at frequencies w_1, w_2 and the sum frequency w_1+w_2, and can be estimated by:

$$B_{XXX}(\omega_1,\omega_2) = E[X(\omega_1)X(\omega_2)X^*(\omega_1+\omega_2)] \tag{3.1}$$

Where X(ω) is the Fourier transform of x(t). The auto bi-coherence (ABC), a normalized bi-spectrum, is defined as:

$$b^2_{XXX}(\omega_1,\omega_2) = \frac{|B_{XXX}(\omega_1,\omega_2)|^2}{E[|X(\omega_1)X(\omega_2)|^2]E[|X(\omega_1+\omega_2)|^2]} \tag{3.2}$$

The discrete time formulation for bi-coherence function is:

$$\hat{b}^2_{XXX}(\omega_1,\omega_2) = \frac{\left|\sum_{i=1}^{NA}[X_i(\omega_1)X_i(\omega_2)X_i^*(\omega_1+\omega_2)]\right|^2}{\{\sum_{i=1}^{NA}[|X_i(\omega_1)X_i(\omega_2)|^2]\}\{\sum_{i=1}^{NA}[|X_i^*(\omega_1+\omega_2)|^2]\}} \tag{3.3}$$

Where $X_i(\omega)$ is the FFT of the i-th block ensemble average data

NA is the total number of ensemble average blocks

By using Schwarz' inequality, it can be shown that the bi-coherence is always bounded by zero and unity. Bi-coherence $b_{xxx}(\omega_1,\omega_2)$ is a function of two frequencies, ω_1 and ω_2. Therefore, a three dimensional plot would be required in order to display a complete bi-coherence function. To minimize the computation requirement associated with the complete 3D bi-coherence function, a special bi-coherence function $b_{xxx}(\omega_r,\omega)$ is frequently utilized for machinery diagnostic application. One of the bi-frequency arguments, ω_1, can be fixed at some particular reference frequency of interest, (e.g. $\omega_1 = \omega_r$ = bearing characteristic frequency or Sync frequency, etc.) while the other frequency argument, $\omega_2= \omega$, sweeps through the entire analysis frequency range. As will be discussed next, there exists a certain limit for the second frequency argument, $\omega_2= \omega$, which is function of the reference frequency ω_r and the sampling frequency ω_s.

The maximum frequency available for a discrete time data is the nyquist frequency, $\omega_n= \omega_s/2$, which is defined as half of its sampling frequency. Therefore, there exists a region over which the ABS is defined in the bi-frequency domain (ω_1, ω_2): First of all, both $\omega 1$ & $\omega 2$ must be limited by $\pm\omega_n$, which is corresponding to a square region in the bi-frequency domain. In addition, the sum frequency $\omega 1+\omega 2$ of the third (implicit) spectral component in the ABS function must also be limited by $\pm\omega_n$, which is corresponding to the region between the 2 -45 degree lines (line D-D & E-E in figure 3.1) in the bi-frequency domain. As a result, the ABS is only defined over the hexagon region as shown in figure 1. Furthermore, due to the symmetric property of the ABS, this hexagon region cab be further reduced. Referring to the definition of ABS in equation 3.1, bi-spectrum is the cumulant average of the 3 spectral components:

$B(\omega_1,\omega_2) = E [X(\omega_1) X(\omega_2) X(-\omega_1-\omega_2)]$. { Note: $X^*(\omega_1+\omega_2)= X(-\omega_1-\omega_2)$ }

Since the order of the spectral sequence is irrelevant, a total of 6 different permutations of these 3 spectral components exist which would lead to 6 identity among ABS:

<u>permutations:</u>

$[\omega_1, \omega_2; -\omega_1-\omega_2]$ $[\omega_2, -\omega_1-\omega_2; \omega_1]$ $[-\omega_1-\omega_2, \omega_1; \omega_2]$

$[\omega_2, \omega_1; -\omega_1-\omega_2]$ $[\omega_1, -\omega_1-\omega_2; \omega_2]$ $[-\omega_1-\omega_2, \omega_2; \omega_1]$

<u>Identities:</u>

$Bxxx(\omega_1, \omega_2) = Bxxx(\omega_2, -\omega_1-\omega_2) = Bxxx(-\omega_1-\omega_2, \omega_1)$

$= Bxxx(\omega_2, \omega_1) = Bxxx(\omega_1, -\omega_1-\omega_2) = Bxxx(-\omega_1-\omega_2, \omega_2)$

Furthermore, the complex conjugate of the ABS provides additional 6 different permutations and 6 more identities as:

$B^*(\omega_1,\omega_2) = E [X^*(\omega_1) X^*(\omega_2) X^*(-\omega_1-\omega_2)] = E [X(-\omega_1) X(-\omega_2) X(\omega_1+\omega_2)]$.

permutations:

$[-\omega_1, -\omega_2; \omega_1+\omega_2]$ $[-\omega_2, \omega_1+\omega_2; -\omega_1,]$ $[\omega_1+\omega_2, -\omega_1; -\omega_2]$

$[-\omega_2, -\omega_1; \omega_1+\omega_2]$ $[-\omega_1, \omega_1+\omega_2; -\omega_2,]$ $[\omega_1+\omega_2, -\omega_2; -\omega_1]$

Identities:

$$\begin{aligned} Bxxx(\omega_1, \omega_2) &= B^*xxx(-\omega_1, -\omega_2) = B^*xxx(-\omega_2, \omega_1+\omega_2) = B^*xxx(\omega_1+\omega_2, -\omega_1) \\ &= B^*xxx(-\omega_2, -\omega_1) = B^*xxx(-\omega_1, \omega_1+\omega_2) = B^*xxx(\omega_1+\omega_2, -\omega_2) \end{aligned}$$

This would lead to the following 3 symmetric properties of ABS:

1. Symmetric with respect to 45^o line A-A
2. Conjugate Symmetric with respect to -45^o line B-B
3. Symmetric with respect to -30^o line C-C: $\omega_1 = -2\omega_2$

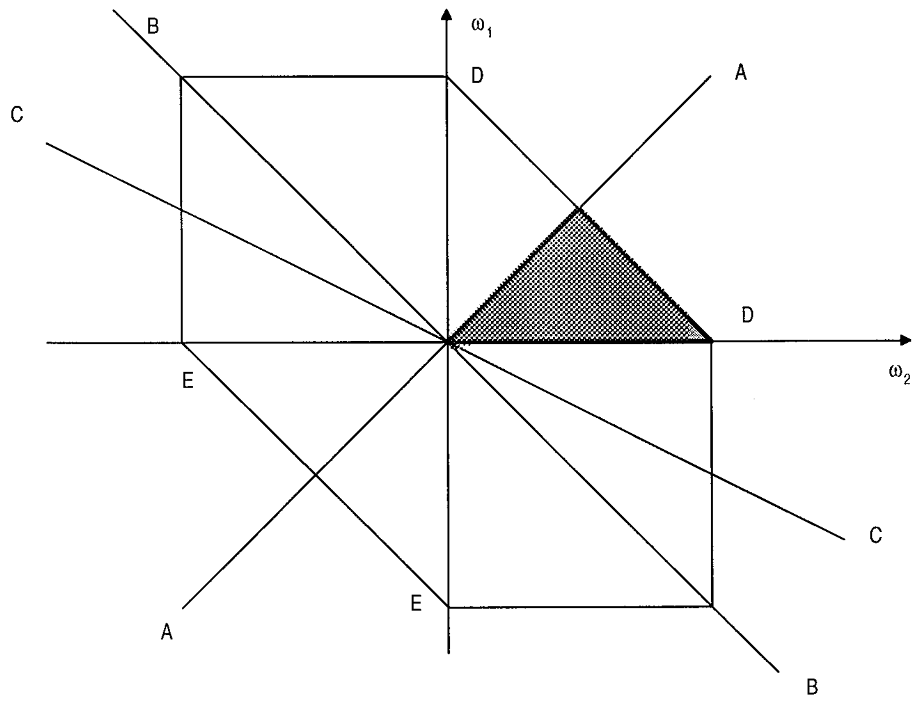


Figure 3.1 Symmetric Property of Auto Bi-Coherence Function

If the ABS in the triangular region in the 1st quadrant (the shaded triangle in figure 1) is known, then the ABS in the entire hexagon region can be obtained based on these 3 symmetric properties, . As a result, this triangular region is the only region over which the computation of ABS is required. However, for the special bi-coherence function $b(\omega_r,\omega)$ discussed above, the frequency argument, ω, should sweep through the entire first quadrant from 0 upto the -45 degree boundary line (line D-D). The parameter "fmax" in the computer codes to be discussed below is refereed to this boundary line.

2.2.3.2 Test Example of bi-coherence function

The bi-coherence algorithm implemented on ASRI's TI-C40 DSP has been preliminary tested utilizing a carefully-design test signal containing both nonlinearly-correlated wave and linearly-independent wave at the same frequency so that, with a known degree of nonlinear correlation in the test signal, the resulting bi-coherence value can be evaluated for its accuracy. In addition, Mr. Bapty of Vanderbilt requested ASRI to provide a set of simulation data with correct bi-coherence results so that he can test the bi-coherence program on the MPP system for verification purpose. Therefore, a simple simulation example has been designed which generates three time histories x1(t), x2(t) and x3(t) as follows:

$$
\begin{aligned}
X_1(t) &= \cos(\omega_a t + \Phi_1) + \cos(\omega_b t + \Phi_2) + \cos[(\omega_a + \omega_b)t + (\Phi_1 + \Phi_2)] + N_1(t) \\
X_2(t) &= \cos(\omega_a t + \Phi_1) + \cos(\omega_b t + \Phi_2) + \cos[(\omega_a + \omega_b)t + \Phi_3] + N_2(t) \\
X_3(t) &= \cos(\omega_a t + \Phi_1) + \cos(\omega_b t + \Phi_2) + 0.5 * \cos[(\omega_a + \omega_b)t + (\Phi_1 + \Phi_2)] \\
&\quad + 0.5 * \cos[(\omega_a + \omega_b)t + \Phi_3] + N_3(t)
\end{aligned}
$$

where $\phi 1$, $\phi 2$ and $\phi 3$ are independent random phase; N1(t), N2(t) and N3(t) are independent Gaussian White Noise, and the sum frequency ω_c (2700 Hz) = ω_a (1000 Hz)+ ω_b (1700 Hz), is the frequency of the third sine wave. Each one of these time series is composed of three sine waves at frequency ω_a , ω_b and ω_c. The first two waves at ω_a and ω_b are identical for each of the time histories. But the third wave at ω_c is quite different. For x1(t), the phase at ω_c is equal to the sum of the phases at ω_a and ω_b, which represents perfect phase coupling. For x2(t), the random phase at ω_c indicates total independence among these three sine waves. For x3(t), 50% of the component at ω_c is phase coupled with ω_a and ω_b, and the remaining 50% with random phase independent of ω_a and ω_b.

Since all these three signal have the same energy distribution in frequency domain, therefore, they all have identical PSD (Power spectral density) as shown in figures 3.2-a, 3.2-b and 3.2-c for x1(t), x2(t) and x3(t) respectively. Figures 3.3-a, 3.3-b, and 3.3-c show their corresponding auto-bicoherence function with the first frequency argument $\omega 1$ fixed at 1000 Hz. The X-axis in figure 3.3 represents the second frequency argument $\omega 2$ of the bi-coherence function. In figure 3.3-a, there is a strong peak at bi-frequency (ωa=1000 Hz ,ωb=1700 Hz) with amplitude close to one. This indicates the components at ω_a , ω_b and ω_c in x1(t) are highly correlated due to strong phase coupling. For x2(t), since the phases at these three frequencies are independent random phases, no significant bicoherence peaks are apparent in figure 3.3-b. For x3(t), the amplitude at bi-frequency (ωa=1000 Hz ,ωb=1700 Hz) in figure 3.3-c, is only 0.729. The corresponding bi-coherence square (0.729^2 = 0.53) indicates that only 50% of the power at ω_c is correlated with ω_a and ω_b, and the remainder is totally independent. Therefore, the bi-coherence analysis completely distinguishes these three time histories even though they have identical PSDs.

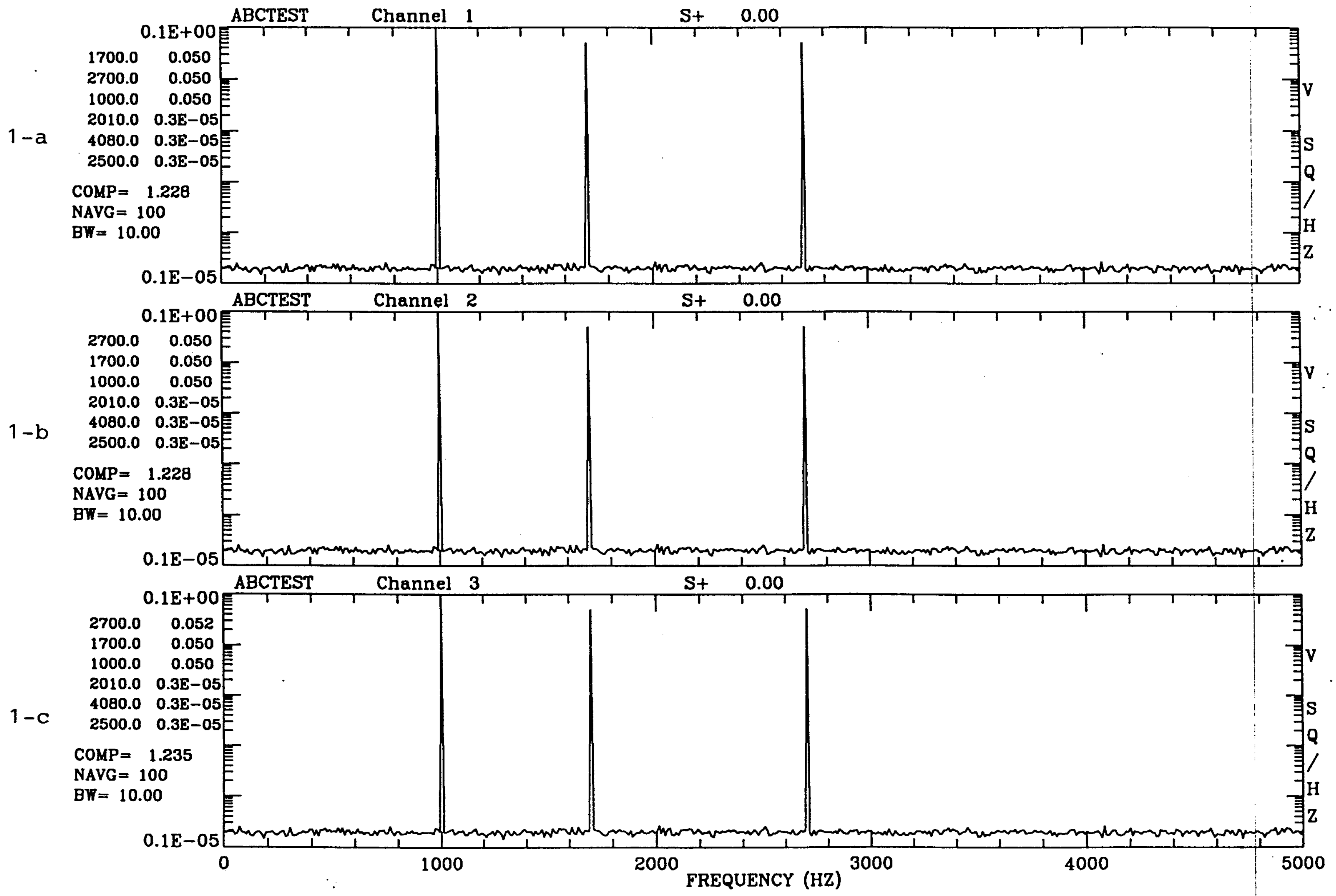


Figure 3.2 PSD NFFT=1024 NAVG=100

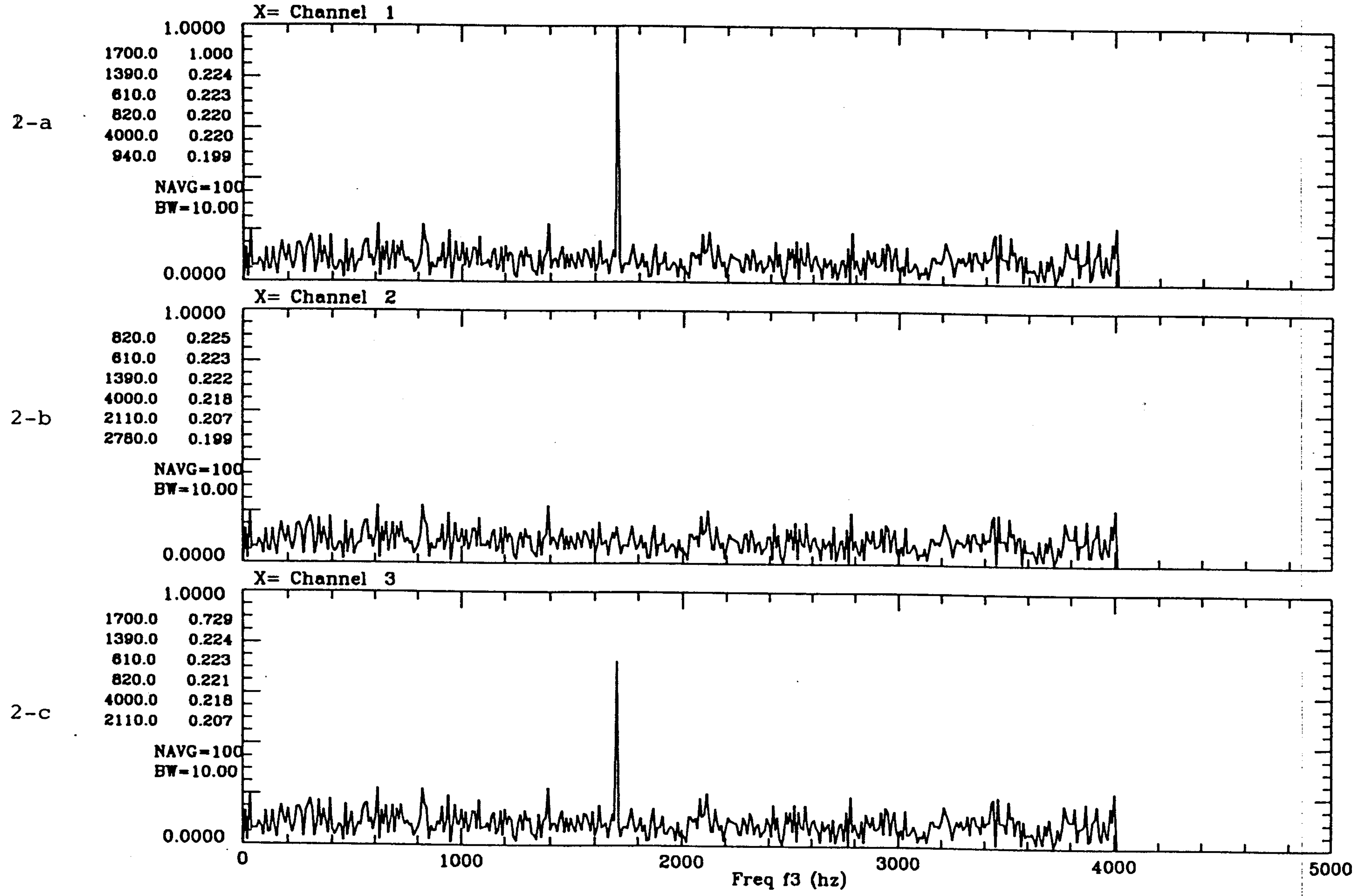


Figure 3.3 Bi-Coherence bxxx(W1= 1000 Hz, W2)

2.2.3.3 DSP Code Overview of bi-coherence function using Data Translation's DSP-EZ programming language running on C40 DSP

The bi-coherence algorithm has been implemented on ASRI's TI-C40 DSP, using Data Translation's DSP-EZ programming language. This section lists the complete DSP Codes of it.

Complete codes of bi-coherence function b(ω_r,ω) on C40 DSP using Data Translation's DSP-EZ programming language

```
'+------------------------+
'| Bicoherence.DSP |
'+------------------------+
'          ************************************************
'          ****  DSP Basic Program Global Declaration  ****
'          ************************************************
        Option Explicit
'+-----------------+
'| Constants |
'+-----------------+
        Const CTRLchan = 0
        Const PERCENTchan = 0
        Const Col5chan=1
        Const TimeChan1 = 1
        Const FFTChan1 =2
        Const BPTimeChan1=3
        Const BPFFTChan1=4
        Const ABSChan1=5
        Const FILTchan = 6
        Const FILTERPOINTS = 0
        Const CUTOFFLower = 1
        Const CUTOFFUpper = 2
        Const FILTER_SIZE =201
        Const BUFFSIZE =2048
        Const FFT_SIZE =2048
'+-----------+
'| Arrays |
'+------------+
        Dim ADbuff As FloatArray
        Dim BPFilter As FloatArray
        Dim TimeData As FloatArray
        Dim FFTdata As ComplexArray
        Dim ImpulseResponse          As FloatArray
        Dim XX As ComplexArray               ' Bicoherence calculation
        Dim PSD As FloatArray
        Dim AA As ComplexArray
        Dim BB As FloatArray
        Dim CC As FloatArray
        Dim BC As FloatArray
'+----------------+
'| Variables |
'+----------------+
        Dim Frequency As Single
        Dim FMax As Single
        Dim dbScale As Single
        Dim MaxFreq As Single
        Dim MinFreq As Single
        Dim Kmax As Single
        Dim NumOfAvg As Single
'           ***********************
'           **** Function Main ****
'           ***********************
Function Main()
'+-------------------+
'| Declaration |
'+-------------------+
Dim cmdbuff  As FloatArray
Dim Utilization As FloatArray
Dim DAdata As FloatArray
Dim BHWnd As FloatArray
Dim WndData As FloatArray
Dim MagData As FloatArray
Dim ZZ As ComplexArray
Dim YY As ComplexArray
Dim XX1 As ComplexArray

Dim ADREADY As Integer
Dim SampleTime As Single
Dim RetrieveTime As Single
Dim Parameterchecked As Integer
Dim i, j, k, kount,m,n, ecode, Length, start,t ,d,p,DSize As Integer
Dim bw, ref, nmax, Fs As Single
Dim  R1, R2, R3 , R4, kref As  Single
Dim J1 As Complex
Dim J2 As Complex
Dim J3 As Complex
'+--------------------------------------------+
'| get the initial A/D frequency |
'+--------------------------------------------+
'kount = 0
Do While Not Host.GetDataReady(CTRLchan)
Loop
cmdbuff = Host.GetBuffer(CTRLchan)
Frequency = cmdbuff(0)
'+-------------------------------------+
'| Define A/D parameters |
'+-------------------------------------+
AD.Frequency = Frequency
AD.ClockSource = DTDSP_ICLK
AD.TriggerSource = DTDSP_ITRG
AD.NumChannels = 1
AD.AcqMode = DTDSP_WRAP
AD.InternalChannelBufferSize = BUFFSIZE*2
'+-------------------------------------+
'| Define D/A parameters |
'+-------------------------------------+
'DA.Frequency = Frequency
'DA.ClockSource = DTDSP_ICLK
```

```
'DA.TriggerSource = DTDSP_ITRG
'DA.NumChannels = 1
'DA.AcqMode = DTDSP_WRAP
'DA.InternalChannelBufferSize = BUFFSIZE*2
'+------------------------------------------------+
'| initialize global array to all zeros |
'+------------------------------------------------+
ImpulseResponse = GenLine(0,0,BUFFSIZE)
ImpulseResponse(BUFFSIZE/2) = FFT_SIZE
Do While Not Host.GetDataReady(CTRLchan)
Loop
'+--------------------+
'| initialize filter |
'+--------------------+
ProcessControl()
'+-----------------------------------+
'| generate Window Data |
'+-----------------------------------+
BHWnd = GenWindow(BLACKMAN_HARRIS_61,BUFFSIZE)
dBScale = -20 * log10(10*FFT_SIZE) + 6.0
SampleTime = 1.0/AD.Frequency * FFT_SIZE
RetrieveTime = 0
Utilization = GenLine(0,0,1)
'+------------------------------+
'| start acquiring data |
'+------------------------------+
AD.ChannelType=DTDSP_DI
AD.Filter=AD.Frequency / 2
AD.Start()
'           ****************
'           * Main looping *
'           ****************
'+--------------------------------------------------+
'| acquire parameters from  buffer
'+--------------------------------------------------+
Fs=Frequency/2
bw=Fs/BUFFSIZE
nmax=BUFFSIZE/2
ref=500
Start =0
Length = BUFFSIZE/2
NumOfAVG=10
ParameterChecked = 1
XX=GenComplexCos(0,0,10000,FFT_SIZE/2+1)

Do While  ParameterChecked = 1
        If Host.GetDataReady(Col5chan) Then
                cmdbuff=Host.GetBuffer(Col5chan)
                NumOfAvg=cmdbuff(0)
                ref=cmdbuff(1)
        End If
        PSD=GenLine(0,0,FFT_SIZE/2+1)
        kref=ref/bw+0.5
        kmax=nmax-kref
        AA=GenComplexCos(0,0,10000,Kmax+1)
        AA = AA * 0
        BB=GenLine(0,0,Kmax+1)
        CC=GenLine(0,0,Kmax+1)
        BC=GenLine(0,0,FFT_SIZE/2+1)
        For m = 1 To NumOfAVG
                TimeStart()
                ADbuff = AD.Scan (BUFFSIZE)
                RetrieveTime = TimeRead()
                TimeStop()
'+------------------------------------------------+
'| calculate average of % utilization
'+------------------------------------------------+
                Utilization(0) = .6* Utilization(0)+.4*(((SampleTime-
RetrieveTime)/SampleTime)*100)
'+-------------------------------------+
'| Extract data from buffer
'+-------------------------------------+
                        i=0
                        TimeData = Extract(ADbuff, BUFFSIZE,i*BUFFSIZE)
'+----------------------------------------------------------------------------+
'| put time history data to host for plotting
'+----------------------------------------------------------------------------+
                        k=i*5+1
                        If (Host.PutDataReady(k)) Then
                                Host.PutBuffer(k)=TimeData
                        End If
'+------------------------------------------------------------+
'| Put FFT data to host for plotting
'+------------------------------------------------------------+
                        k = i*5+2
                        If (Host.PutDataReady(k)) Then
                                WndData=Window(BHWnd, TimeData)
                                FFTData=fft(WndData)
                                MagData=dB(FFTData)+dBScale
'                               Host.PutBuffer(k)=MagData
                                If Host.PutDataReady(PERCENTchan) Then
                                                Host.PutBuffer(PERCENTchan) =
Utilization
                                End If
                        End If
'+--------------------------------------------------------------
'       Calculate bicoherence
'+--------------------------------------------------------------
                        XX=FFTData
                        J1=XX(kref)
                        PSD = Abs(XX)
                        XX1 = Extract( XX, Kmax +1, 0)
                        ZZ = CopyVector(XX, Kref)
                        AA = AA + J1 * XX1 * Conjugate(ZZ)
                        BB = BB + Abs( J1 * XX1) * Abs(J1 * XX1)
                        CC = CC + Abs(ZZ) * Abs(ZZ)
                Next m
                For d=0 To kmax
                        BC(d)=Abs(AA(d)) * Abs(AA(d)) /(BB(d)*CC(d))
                Next d
                For d= Kmax+1 To  Nmax
                        BC(d)=0
                Next d
                If  Host.PutDataReady(BPTimechan1) Then
                        Host.PutBuffer(BPTimechan1)=BC
                        If Host.PutDataReady(PERCENTchan) Then
                                        Host.PutBuffer(PERCENTchan) = Utilization
                        End If
                End If
                For d= 0 To nmax
                                PSD(d)=10*log10(PSD(d)/NumOfAvg)+ dBScale
                Next d
                If( Host.PutDataReady(ABSchan1)) Then
                         Host.PutBuffer(ABSchan1) = PSD
                        If Host.PutDataReady(PERCENTchan) Then

                                Host.PutBuffer(PERCENTchan) = Utilization
                        End If
                End If
'+----------------------------------------+
'| Upgrade filter if necessary
'+----------------------------------------+
        If (Host.GetDataReady(CTRLchan)  And (AD.DataReady(0) < BUFFSIZE))
Then
                ProcessControl()
        End If
Loop   'End of Do While loop
End Function
'          ********************************
'          **** Function ProcessControl ****
'          ********************************
Function ProcessControl()
Dim ctrlbuff As FloatArray
Dim FilterDataTemp As FloatArray
Dim FFTDataTemp As ComplexArray
Dim ready As Integer
Dim lowercut, uppercut As Single
Dim length As Integer
ready = TRUE
Do While  ready
  ctrlbuff = Host.GetBuffer(CTRLchan)
  ready = Host.GetDataReady(CTRLchan)
Loop
'+-----------------------------------+
'| generate a band pass filter object |
'+-----------------------------------+
lowercut = ctrlbuff(CUTOFFLower)
uppercut =ctrlbuff(CUTOFFUpper)
length   = ctrlbuff(FILTERPOINTS)
BPFilter = InitBandPassFilter(lowercut,uppercut,Frequency,length)
FilterDataTemp = Filter(BPFilter,ImpulseResponse)
```

```
FFTDataTemp = fft(FilterDataTemp)
FilterDataTemp = dB(FFTDataTemp)
FilterDataTemp = FilterDataTemp - 20*log10(FFT_SIZE) + 6.0
'+-------------------------------------+
'| Send data to host for filter plotting |
'+-------------------------------------+
Host.PutBuffer(FILTchan) = FilterDataTemp
End Function
'                    ******************************************
'                    **              Function ApplyFFT( )            **
'                    ******************************************
Function ApplyFFT(SomeData As FloatArray) As FloatArray
Dim Temp1 As FloatArray
Dim Temp2 As ComplexArray
Dim BHWnd As FloatArray
        BHWnd=GenWindow(BLACKMAN_HARRIS_61,BUFFSIZE)
        Temp1=Window(BHWnd, SomeData)
        Temp2=fft(Temp1)
        ApplyFFT=dB(Temp2)+dBScale
End Function
'                    ******************************
'                    **              Function CopyVector            **
'                    ******************************
Function CopyVector(XX As ComplexArray,  SomeVal As Integer) As ComplexArray
Dim Temp As ComplexArray
Dim i As Integer
        Temp = GenComplexCos(0,0,10000,Kmax+1)
        For i = 0 To kmax
                Temp( i ) = XX( SomeVal + i )
        Next i
        CopyVector = Temp
End Function
'               ******************************
'               **              Function Conjugate              **
'               ******************************
Function Conjugate( SomeData As ComplexArray) As ComplexArray
Dim Temp As ComplexArray
Dim i As Integer
Dim r1, r2 As Single
        Temp = GenComplexCos(0,0,10000,Kmax+1)
        Temp = SomeData
        For i = 0 To  Kmax
                r1 = -Imag(Temp(i) )
                r2 = Real(Temp(i) )
                Temp(i) = MakeComplex(r2, r1)
        Next i
        Conjugate = Temp
End Function
```

2.2.4 Tri-Spectral Analysis - higher order coherence estimations among four spectral components and modulation sideband

2.2.4.1 General background, algorithm and simulation example of tri-coherence function

The auto-tri-coherence function has been implemented on a TI-C40 DSP, and has been converted to MSFC's MPP system. The tri-spectral analysis can identify a different type of correlation when four different spectral components are cubically correlated. A special case of it application is to identify whether an apparent PSD sideband is due to modulation or simply three independent waves. In machinery diagnostics, many failure modes will cause amplitude modulation between an envelope signal and its carrier signal, and, as a result, will generate a pair of sidebands around a center frequency component. Therefore, such sideband phenomenon is commonly treated as an off-nominal signature. The computer code for the tri-coherence program is currently being implemented and tested on ASRI's C40/PC computer system. In addition, a simulation example to be used by Vanderbilt for the verification of the tri-coherence program in the MPP system has been designed, and will be provided to Vanderbilt. The complete codes in a conversion format from ASRI's DSP system to Vanderbilt MGA Model Based Programming environment will be transferred to Vanderbilt in the next reporting period. The technical discussion in this report provides some background information to aide Vanderbilt's programmers in implementing and testing the tri-coherence program.

The tri-spectrum represents a fourth joint moment among four different waves. For a stationary random time series x(t), its fourth order cumulant spectrum, the tri-spectrum, is defined as :

$$\begin{aligned} T_{XXXX}(\omega_1,\omega_2,\omega_3) = {} & E[X(\omega_1)X(\omega_2)X(\omega_3)X^*(\omega_1+\omega_2+\omega_3)] \\ & -E[X(\omega_1)X(\omega_2)]E[X(\omega_3)X^*(\omega_1+\omega_2+\omega_3)] \\ & -E[X(\omega_1)X(\omega_3)]E[X(\omega_2)X^*(\omega_1+\omega_2+\omega_3)] \\ & -E[X(\omega_2)X(\omega_3)]E[X(\omega_1)X^*(\omega_1+\omega_2+\omega_3)] \end{aligned} \tag{4.1}$$

Where $X(\omega)$ is the Fourier transform of x(t). And the tri-coherence is a normalized tri-spectrum, normalized by the power of the four spectral components at w1, w2, w3 and w1+w2+w3.

In general, the tri-spectrum $T_{XXXX}(w_1,w_2,w_3)$ is a function of three frequency arguments. However, a special tricoherence function $T_X(w)$ defined in equation 2 can be used to identify the correlation within a PSD sideband pattern by fixing the first and second frequency arguments, w1 & w2, at the frequencies of two consecutive components within a cluster of sidebands spaced by frequency Δ. This special tricoherence function can be used to identify whether an apparent PSD sideband structure is due to modulation signal or simply three totally independent components.

$$T_X(\omega|\omega_{s1},\omega_{s2}) = T_{XXXX}(-\omega_{s1},\omega_{s2},\omega) = E[X^*(\omega_{s1})X(\omega_{s2})X(\omega)X^*(-\omega_{s1}+\omega_{s2}+\omega)]$$
$$-E[X^*(\omega_{s1})X(\omega_{s2})]E[X(\omega)X^*(-\omega_{s1}+\omega_{s2}+\omega)] \qquad (4.2)$$

Where ω_{S1} = Frequency of a Sideband component

ω_{S2} = Frequency of the Next Sideband Component (i.e. $\omega_{S2} = \omega_{S1} + \Delta$)

The auto tri-coherence , $t_X(w)$, a normalized tri-spectrum, is defined as:

$$t_X^2(\omega|\omega_{s1},\omega_{s2}) == \frac{|T_X(\omega|\omega_{s1},\omega_{s2})|^2}{E[|X(\omega_{s1})X(\omega_{s2})X(\omega\ )|^2]E[|X^*(\omega_{s1}+\omega_{s2}+\omega)|^2]} \qquad (4.3)$$

The discrete time formulation for the special tri-coherence function is:

$$\hat{t}_X^2(\omega\ |\omega_{s1},\omega_{s2}) = \frac{|\sum_{i=1}^{NA}[X_i^*(\omega_{s1})X_i(\omega_{s2})X_i(\omega)X_i^*(-\omega_{s1}+\omega_{s2}+\omega)] - [X_i^*(\omega_{s1})X_i(\omega)][X_i(\omega_{s2})X_i^*(-\omega_{s1}+\omega_{s2}+\omega)]\ |^2}{\{\sum_{i=1}^{NA}[|X_i^*(\omega_{s1})X_i(\omega_{s2})X_i(\omega\ )|^2]\}\{\sum_{i=1}^{NA}|X^*{}_i(-\omega_{s1}+\omega_{s2}+\omega\ ))|^2]\}}$$

(4.4)

Where $X_i(w)$ is the FFT of the i-th block ensemble average data

NA is the total number of ensemble average blocks

w_{S1} = Frequency of a Sideband component

w_{S2} = Frequency of the Next Sideband Component (i.e. $w_{S2} = w_{S1} + D$)

Due to the existence of nonlinear phase correlation among the participating sideband components, this special tri-coherence function should show a strong coherence peaks at these sideband frequencies except at ω_{S1} as well as the last component within the sideband sequence. Conversely, due to a lack of phase correlation, no coherence peak will be identified for a sequence of independent (uncorrelated) spectral peaks. As a result, this special tri-coherence function can discriminate between a modulation sideband signal and independent sideband signal.

It should be pointed out that, the purpose of the second numerator term in equation (4.4):

$$|\sum\ [X_i^*(\omega_{s1})X_i(\omega)][X_i(\omega_{s2})X_i^*(-\omega_{s1}+\omega_{s2}+\omega)]\ |^2 \qquad (4.5)$$

is to force $\hat{t}_X^2(\omega\ |\omega_{s1},\omega_{s2}) = 0$ when w =w_{S1}, since its tri-spectrum at this frequency

$$T_X(\omega_{s1}|\omega_{s1},\omega_{s2}) = T_{XXXX}(-\omega_{s1},\omega_{s2},\omega_{s1}) = E[X^*(\omega_{s1})X(\omega_{s2})X(\omega_{s1})X^*(\omega_{s2})]$$
$$= E[|X^*(\omega_{s1})|^2|X(\omega_{s2})|^2]$$

The frequency argument, ω, of the special tri-coherence function, $t_x(w)$, is defined over a frequency range from 0 to a maximum frequency ω_{max}. This limit frequency for ω is a function of the reference frequency ω_{s1}, ω_{s2} and the sampling frequency, since $t_x(w)$ is only defined over the triangular region as shown in figure 4.1. Therefore the frequency argument, ω, should sweep through the entire first quadrant from 0 upto the -45 degree boundary line which corresponding to the maximum frequency ω_{max}.

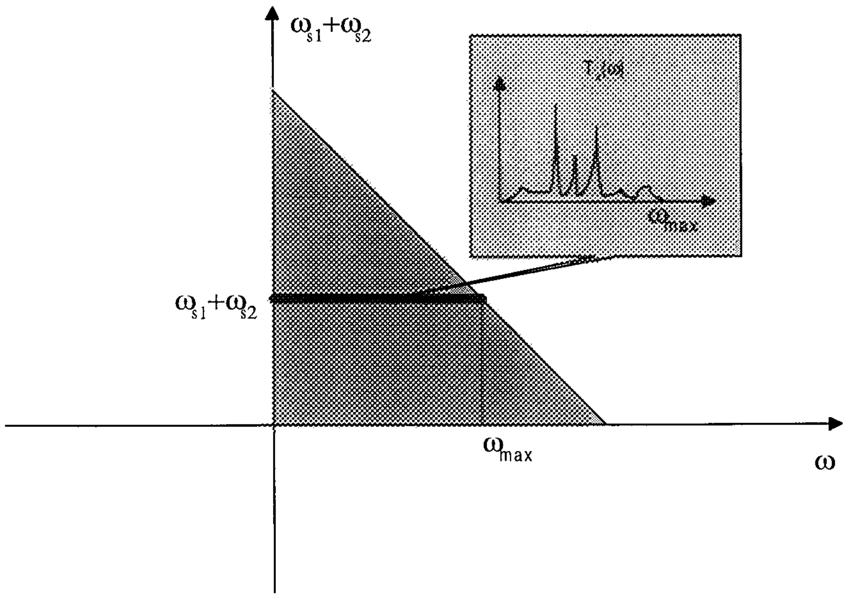


Figure 4.1 Frequency range for the special Auto Tri-Coherence Function

2.2.4.2 Simulation Example for Tri-Coherence Analysis Program

A simulation example has been designed and generated to demonstrate the sideband identification capability of the tri-coherence function. The data of this simulation example will also be provide to Vanderbilt for testing and verification of the tri-coherence program in the MPP system. The modulation sideband signal x(t) is composed of an envelop signal at frequency ω_e and drifting phase $\psi_e(t)$, and a carrier signal at frequency ω_c & phase $\psi_c(t)$. A small amount of Gaussian White Noise is added to the signal. The sampling frequency is 10,240 Hz. The modulation (multiplication) between the envelop and the carrier signal will generate a PSD side-band structure with left, center and right side-band components at frequencies ω_c-ω_e, ω_c, and ω_c+ω_e respectively. Notice that, not only the frequency becomes sum and difference, but also their phase become sum and difference. Such phase correlation can be identified by using the special tri-coherence. The independent sideband signal y(t) is composed of three independent sine waves at frequency ω_c-ω_e, ω_c, ω_c+ω_e, and phase $\psi_L(t)$, $\psi_C(t)$ $\psi_R(t)$, respectively. The PSD of this independent sideband signal will have an apparent side-band structure with left, center and right side-band components at frequencies ω_c-ω_e, ω_c, and ω_c+ω_e respectively. However, the phases among all these three waves within y(t) are totally independent of each other.

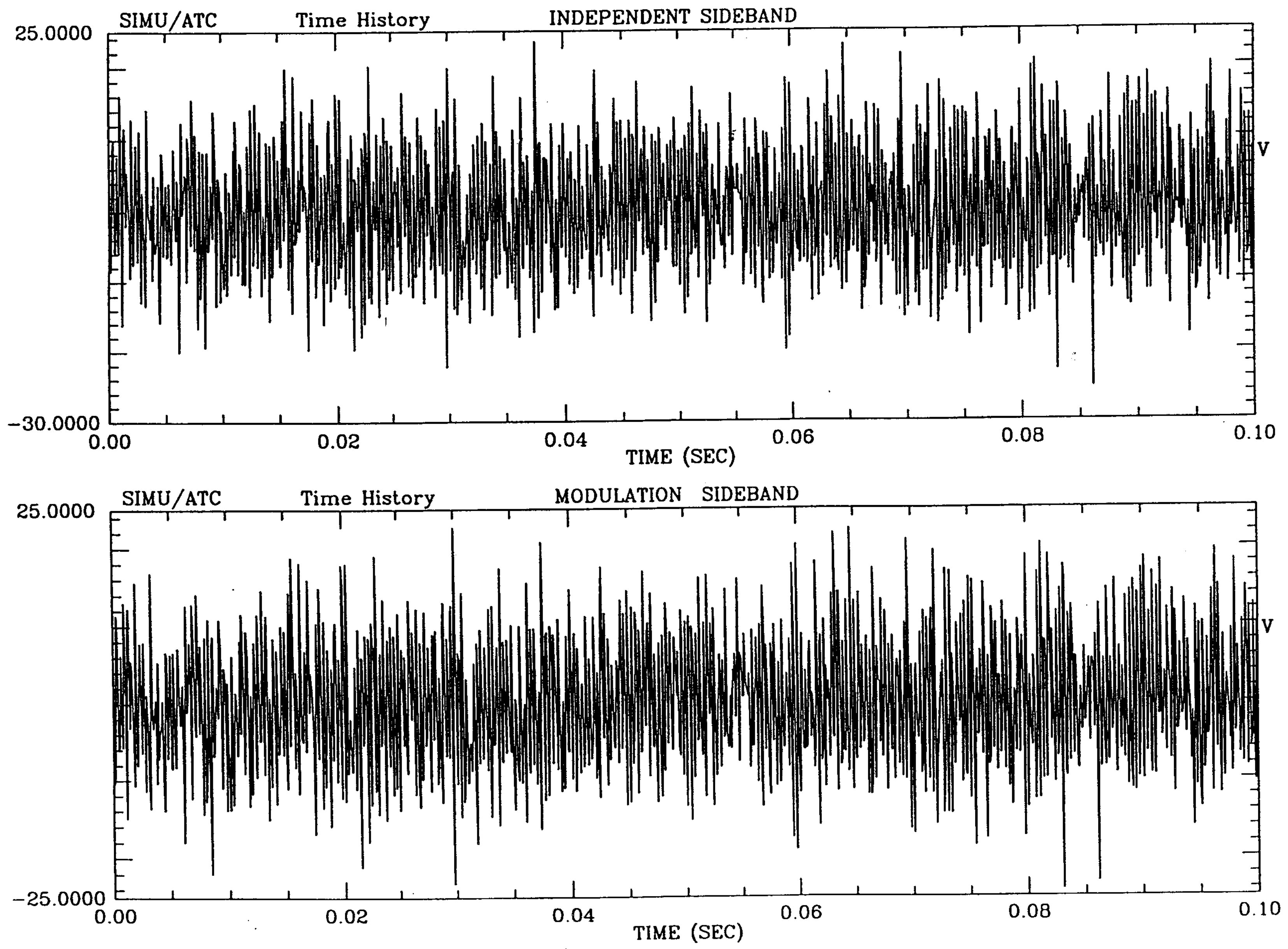
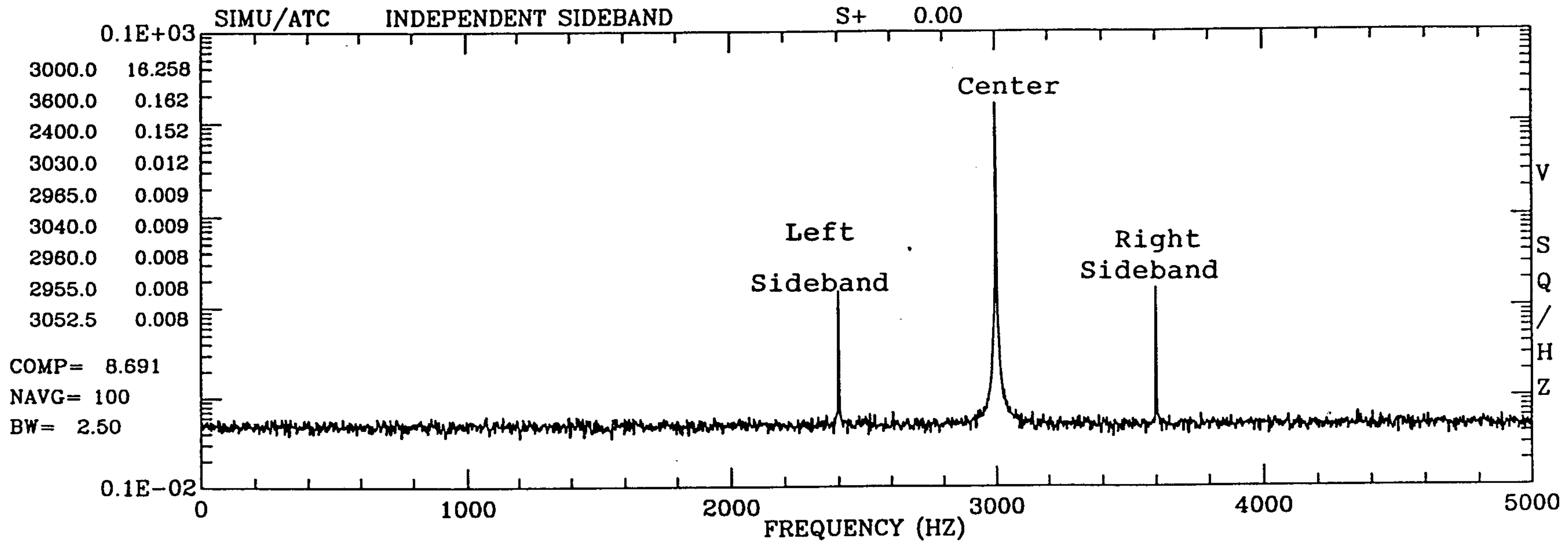
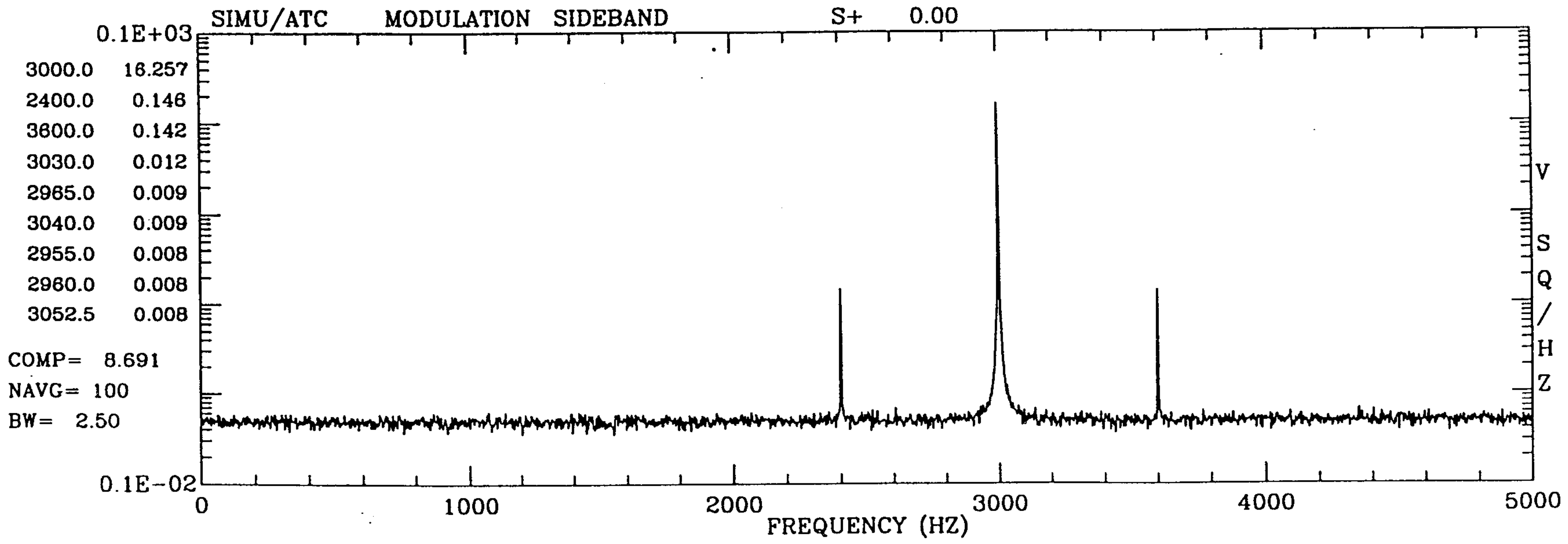


Figure 4.2 Time History



3000.0	16.258
3600.0	0.162
2400.0	0.152
3030.0	0.012
2965.0	0.009
3040.0	0.009
2960.0	0.008
2955.0	0.008
3052.5	0.008

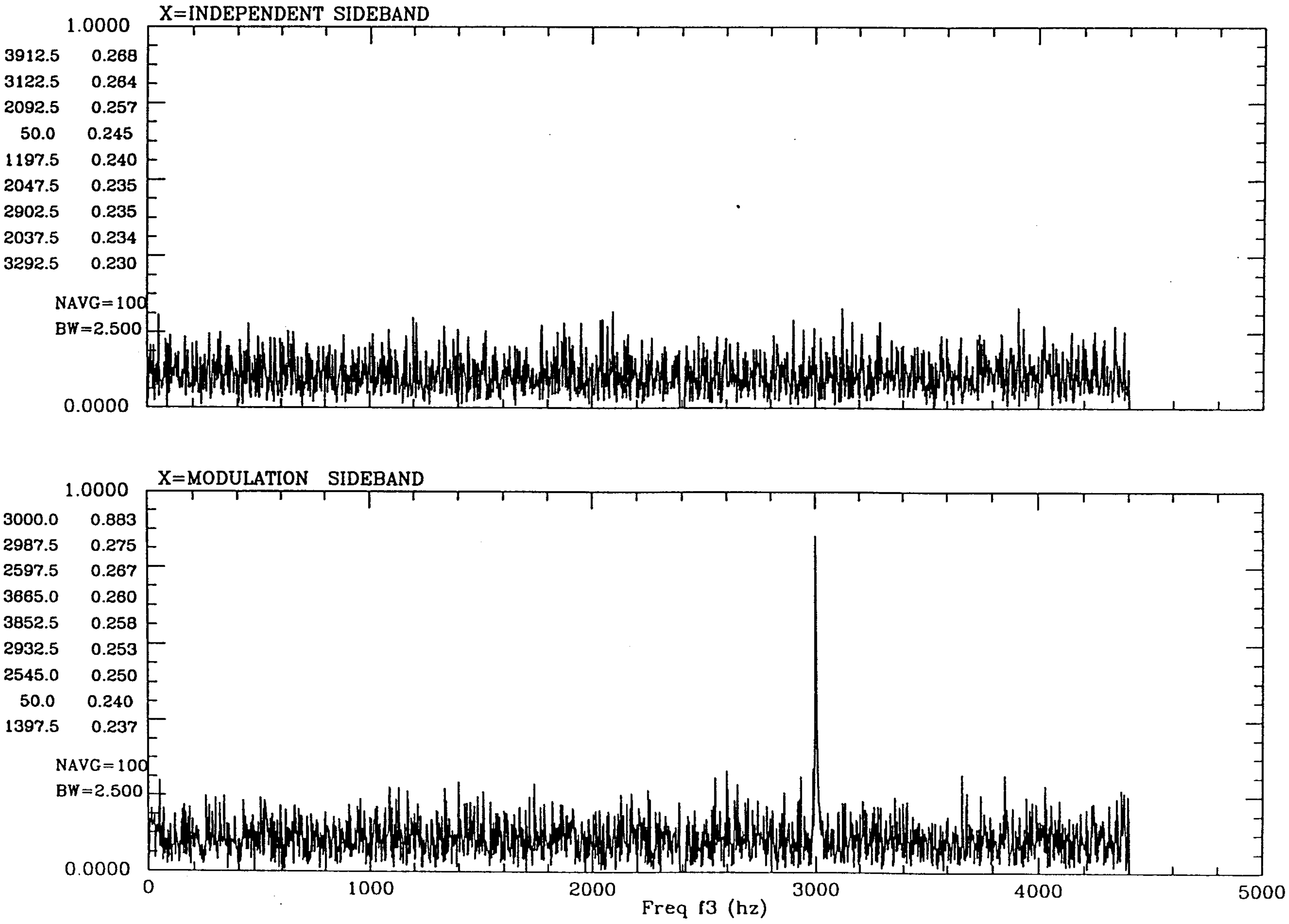
COMP= 8.691
NAVG= 100
BW= 2.50



3000.0	16.257
2400.0	0.146
3600.0	0.142
3030.0	0.012
2965.0	0.009
3040.0	0.009
2955.0	0.008
2960.0	0.008
3052.5	0.008

COMP= 8.691
NAVG= 100
BW= 2.50

Figure 4.3 PSD



¢ SIMU/ATC Txxxx(f1=−2400.0,f2= 3000.0,f3),S+ 0.00

Figure 4.4 Tri-Coherence

$$x(t) = \{10 + 2 * \cos[\omega_e t + \psi_e(t)]\}\cos[\omega_c t + \psi_c(t)] + GWN(t)$$

$$y(t) = 10 * \{\cos[(\omega_c - \omega_e)t + \psi_L(t)] + \cos[\omega_c t + \psi_C(t)] + \cos[(\omega_c + \omega_e)t + \psi_R(t)]\} + GWN(t)$$

Where $\omega_e = 600$ Hz
$\omega_c = 3000$ Hz

Figure 4.2 and 4.3 shows the time histories and PSDs of x(t) and y(t). An apparent sideband structure at 2400 Hz, 3000 Hz, and 3600 Hz is shown in both PSDs which are almost identical to each other. From the time history and PSD, it is difficult to discriminate between the independent and modulation sideband signal. This is because their difference is hidden in the phase coupling relationship among different frequencies. Figure 4.4 shows the special tri-coherence function for y(t) and x(t) with the first and the second frequency arguments fixed at the negative left side-band (-2400 Hz) and the center frequency (3000 Hz). The frequency of the tri-coherence is defined over the 0 to 4520 Hz (i.e. 5120-600 Hz). For the modulation sideband signal x(t), the tri-coherence indeed identify a strong coherence at the center frequency, 3000 Hz, which indicates that it is a true modulation generated side-band signal. While for the independent sideband signal y(t), no significant correlation is identified since the apparent side-band in y(t) is due to three totally independent waves. Therefore, the tri-coherence analysis successfully discriminate between these 2 sideband signal which all have identical PSD.

2.2.4.3 DSP Code Overview of tri-coherence function using Data Translation's DSP-EZ programming language running on C40 DSP

```
'+-------------------------+
'| TricCoherence.DSP |
'+-------------------------+
'        ***********************************************
'        ****  DSP Basic Program Global Declaration  ****
'        ***********************************************
        Option Explicit
'+-----------------+
'| Constants |
'+-----------------+
        Const CTRLchan = 0
        Const Col5chan=1
        Const PERCENTchan = 0
                Const TimeChan1 = 1
        Const FFTChan1 =2
        Const BPTimeChan1=3
        Const BPFFTChan1=4
        Const ABSChan1=5
                Const TimeChan2 = 6
        Const FFTChan2 =7
        Const BPTimeChan2=8
        Const BPFFTChan2=9
        Const ABSChan2=10
        Const FILTchan = 11
        Const RealChan = 12
                Const FILTERPOINTS = 0
        Const CUTOFFLower = 1
        Const CUTOFFUpper = 2
        Const FILTER_SIZE =201
        Const BUFFSIZE =2048
        Const FFT_SIZE =2048
'+-----------+
'| Arrays |
'+------------+
        Dim ADbuff As FloatArray
        Dim BPFilter0 As FloatArray
        Dim BPFilter1 As FloatArray
        Dim TimeData As FloatArray
        Dim FFTdata As ComplexArray
        Dim ImpulseResponse         As FloatArray
        Dim XX As ComplexArray              ' Tricoherence calculation
        Dim PSD As FloatArray
        Dim AA As ComplexArray
        Dim BB As FloatArray
        Dim CC As FloatArray
        Dim BC As FloatArray
'+----------------+
'| Variables |
'+----------------+
        Dim Frequency As Single
        Dim FMax As Single
        Dim dbScale As Single
        Dim MaxFreq As Single
        Dim MinFreq As Single
        Dim NumOfAvg As Single
'          **********************
'          **** Function Main ****
'          **********************
Function Main()
'+-------------------+
'| Declaration |
'+-------------------+
Dim cmdbuff  As FloatArray
'Dim Utilization As FloatArray
'Dim LossRatio As FloatArray
Dim DAdata As FloatArray
'Dim BHWnd As FloatArray
'Dim WndData As FloatArray
Dim MagData As FloatArray
Dim ADREADY As Integer
'Dim SampleTime As Single
'Dim RetrieveTime As Single
Dim Parameterchecked As Integer
Dim i, j, k, kount,m,n, ecode, Length, start,t ,d,p,DSize As Integer
Dim ref1, ref2  As  Single
```

```
'+------------------------------------------+
'| get the initial A/D frequency |
'+------------------------------------------+
'kount = 0
Do While Not Host.GetDataReady(CTRLchan)
Loop
cmdbuff = Host.GetBuffer(CTRLchan)
Frequency = cmdbuff(0)
'+----------------------------------+
'| Define A/D parameters |
'+------------------------------------+
AD.Frequency = Frequency
AD.ClockSource = DTDSP_ICLK
AD.TriggerSource = DTDSP_ITRG
AD.NumChannels = 2
AD.AcqMode = DTDSP_WRAP
AD.InternalChannelBufferSize = BUFFSIZE*2
'+------------------------------------+
'| Define D/A parameters |
'+------------------------------------+
'DA.Frequency = Frequency
'DA.ClockSource = DTDSP_ICLK
'DA.TriggerSource = DTDSP_ITRG
'DA.NumChannels = 1
'DA.AcqMode = DTDSP_WRAP
'DA.InternalChannelBufferSize = BUFFSIZE*2
'+------------------------------------------------+
'| initialize global array to all zeros |
'+------------------------------------------------+
ImpulseResponse = GenLine(0,0,BUFFSIZE)
ImpulseResponse(BUFFSIZE/2) = FFT_SIZE
Do While Not Host.GetDataReady(CTRLchan)
Loop
'+---------------------+
'| initialize filter |
'+---------------------+
ProcessControl()
'+------------------------------------+
'| generate Window Data |
'+------------------------------------+
'BHWnd = GenWindow(BLACKMAN_HARRIS_61,BUFFSIZE)
dBScale = -20 * log10(10*FFT_SIZE) + 6.0
'SampleTime = 1.0/AD.Frequency * FFT_SIZE
'RetrieveTime = 0
'Utilization = GenLine(0,0,1)
'LossRatio = GenLine(0,0,1)
'+-------------------------------+
'| start acquiring data |
'+-------------------------------+
AD.ChannelType=DTDSP_DI
AD.Filter=AD.Frequency / 2
AD.Start()
'            ***************
'           * Main looping *
'           ****************
ref1=1000
ref2=3000
' ref1 = first reference frequency specified by user
' ref2 = second reference frequency specified by user
Start =0
Length = BUFFSIZE/2
NumOfAVG=10
ParameterChecked = 1
Do While  ParameterChecked = 1
        If Host.GetDataReady(Col5chan) Then
                cmdbuff=Host.GetBuffer(Col5chan)
                NumOfAvg=cmdbuff(0)
                ref1=cmdbuff(1)
                ref2=cmdbuff(2)
        End If
' +-----------------------------------------------------
' |    Calculate  tricoherence
' +-----------------------------------------------------
     TriCoherence(ref1, ref2)
'+----------------------------------------+
'| Upgrade filter if necessary
'+----------------------------------------+
         If (Host.GetDataReady(CTRLchan)  And (AD.DataReady(0) < BUFFSIZE))
Then
                ProcessControl()
        End If
Loop   'End of Do While loop
End Function
'           *********************************
'           **** Function ProcessControl ****
'           *********************************
Function ProcessControl()
Dim ctrlbuff As FloatArray
Dim FilterDataTemp As FloatArray
Dim FFTDataTemp As ComplexArray
Dim ready As Integer
Dim lowercut, uppercut As Single
Dim length As Integer
ready = TRUE
Do While  ready
  ctrlbuff = Host.GetBuffer(CTRLchan)
  ready = Host.GetDataReady(CTRLchan)
Loop
'+-----------------------------------+
'| generate a band pass filter object |
'+-----------------------------------+
lowercut = ctrlbuff(CUTOFFLower)
uppercut =ctrlbuff(CUTOFFUpper)
length   = ctrlbuff(FILTERPOINTS)
BPFilter0 = InitBandPassFilter(lowercut,uppercut,Frequency,length)
BPFilter1 = InitBandPassFilter(lowercut,uppercut,Frequency,length)
FilterDataTemp = Filter(BPFilter0,ImpulseResponse)
```

```
FFTDataTemp = fft(FilterDataTemp)
FilterDataTemp = dB(FFTDataTemp)
FilterDataTemp = FilterDataTemp - 20*log10(FFT_SIZE) + 6.0
'+--------------------------------------+
'| Send data to host for filter plotting |
'+--------------------------------------+
Host.PutBuffer(FILTchan) = FilterDataTemp
End Function
'                    ********************************************
'                    **           Function ApplyFFT( )         **
'                    ********************************************
Function ApplyFFT(SomeData As FloatArray) As FloatArray
Dim Temp1 As FloatArray
Dim Temp2 As ComplexArray
Dim BHWnd As FloatArray
        BHWnd=GenWindow(BLACKMAN_HARRIS_61,BUFFSIZE)
        Temp1=Window(BHWnd, SomeData)
        Temp2=fft(Temp1)
        ApplyFFT=dB(Temp2)+dBScale
End Function
'           ******************************************
'           **      Function  TriCoherence()        **
'           ******************************************
Function TriCoherence( ref11 As  Single, ref22 As  Single )
Dim ProcessTime As Single
Dim SampleTime As Single
Dim RetrieveTime As Single
Dim LossRatio As FloatArray
Dim BHWnd As FloatArray
Dim WndData As FloatArray
Dim Utilization As FloatArray
Dim  kref, kref1, kref2, kmax As Single
Dim bw,  nmax, Fs As Single
Dim  R1, R2, R3 , R4 As  Single
Dim  JJ1 As Complex
Dim  JJ2 As  Complex
Dim J1 As Complex
Dim J2 As Complex
Dim J3 As Complex
Dim d, j, i, m  As Integer
'+-----------------------------------+
'| generate Window Data |
'+-----------------------------------+
BHWnd = GenWindow(BLACKMAN_HARRIS_61,BUFFSIZE)
SampleTime = 1.0/AD.Frequency * FFT_SIZE
RetrieveTime = 0
Utilization = GenLine(0,0,1)
LossRatio = GenLine(0,0,1)
'+---------------------------------------------------+
'| acquire parameters from  buffer
'+---------------------------------------------------+
Fs=Frequency/2
bw=Fs/BUFFSIZE
nmax=BUFFSIZE/2
```

```
XX=GenComplexCos(0,0,10000,FFT_SIZE/2+1)
kref1=ref11/bw+0.5
kref2=ref22/bw+0.5
kref=kref2-kref1
kmax=nmax-kref
ProcessTime = 0
j=AD.NumChannels - 1
For  i = 0  To  j
'' +-----------------------------------------------------
' |   Initialize  global  array    |
' +-----------------------------------------------------
              PSD = GenLine(0,0,FFT_SIZE/2+1)
              AA = GenComplexCos(0,0,10000,FFT_SIZE/2+1)
          AA = AA * 0
          BB = GenLine(0,0,FFT_SIZE/2+1)
             CC = GenLine(0,0,FFT_SIZE/2+1)
             BC = GenLine(0,0,FFT_SIZE/2+1)
           For m = 1 To NumOfAVG
                         TimeStart()
                         ADbuff = AD.Scan (BUFFSIZE)
                         RetrieveTime = TimeRead()
                        TimeStop()
'+---------------------------------------+
'| Extract data from buffer
'+---------------------------------------+
                           TimeData = Extract(ADbuff, BUFFSIZE,i*BUFFSIZE)

'+---------------------------------------------------+
'| calculate average of % utilization
'+---------------------------------------------------+
                          Utilization(0) = .6* Utilization(0)+.4*(((SampleTime-
RetrieveTime)/SampleTime)*100)
                          TimeStart()
'+-------------------------------------------------------------+
'| Put FFT data to host for plotting
'+-------------------------------------------------------------+
                           WndData=Window(BHWnd, TimeData)
                           FFTData=fft(WndData)
'+--------------------------------------------------------------
'         Calculate tricoherence
'+--------------------------------------------------------------
                              XX=FFTData
                              JJ1=XX(kref1)
                              JJ2=XX(kref2)
                              R1= -Imag(JJ1)
                              R2= Real(JJ1)
                              J2=MakeComplex( R2, R1)                      'conjg (JJ1)
                              J1=J2*JJ2
                              For d=0 To nmax
                                        R1=Abs(XX(d))
                                        PSD(d)= PSD(d)+ R1*R1
                              Next d
'                             PSD = PSD + Abs(XX)*Abs(XX)
                              For d=0 To kmax
                        R1= -Imag(XX(d+kref))
                        R2= Real(XX(d+kref ))
                        J2=MakeComplex( R2, R1)
                        AA(d)=AA(d)+J1*XX(d)*J2
                        R3=Abs(J1*XX(d))
                        BB(d)=BB(d)+R3*R3
                        R4=Abs(XX(kref+d))
                        CC(d)=CC(d)+R4*R4
                Next d
'+-----------------------------------------------------------------------------+
'| put time history data to host for plotting
'+-----------------------------------------------------------------------------+
                If (Host.PutDataReady(TimeChan1+i*5)) Then
                        Host.PutBuffer(TimeChan1+i*5)=TimeData
                        If Host.PutDataReady(PERCENTchan) Then
                                Host.PutBuffer(PERCENTchan) = Utilization
                        End If
                End If

                ProcessTime =  TimeRead()
                TimeStop()
                LossRatio(0)=SampleTime/(RetrieveTime+ProcessTime) * 100
                If Host.PutDataReady(Realchan) Then
                        Host.PutBuffer(Realchan) =LossRatio
                End If
        Next m
        For d=0 To kmax
                BC(d)=Abs(AA(d)) * Abs(AA(d)) /(BB(d)*CC(d))
        Next d
        For d= Kmax+1 To  Nmax
                BC(d)=0
        Next d
'               set tri-coherence =0 at w=ws1 and its two surrounding bins
                BC(kref1)=0
                BC(kref1+1)=0
                BC(kref1-1)=0
        If  Host.PutDataReady(BPTimechan1+i*5) Then
                Host.PutBuffer(BPTimechan1+i*5)=BC
        End If
        For d= 0 To nmax
                        PSD(d)=10*log10(PSD(d)/NumOfAvg)+ dBScale
        Next d
        If( Host.PutDataReady(ABSchan1+i*5)) Then
                 Host.PutBuffer(ABSchan1+i*5) = PSD
        End If
Next  i
End  Function
```

2.2.5 Envelope Detection Method - Demodulation of envelope signal using Hilbert Transform

2.2.5.1 General background, algorithm and simulation example of high frequency envelope analysis

The envelope detection method for bearing fault detection is based on the observation that the bearing characteristic frequency may modulate structural or sensor resonant frequencies excited by defect impacts. This impact will produce transient ringing pulses at the impact repetition rate (the modulating frequency). The harmonics of such impact excitation extend well into the high frequency region (10 KHz - 100 KHz) and excite the structural or the sensor resonant frequencies. The resonant response of this motion is detected and refereed to as the carrier frequency. What is of interest however is not the carrier frequency, rather the modulating frequency of the carrier (impact rate). Use of an amplitude demodulation algorithm such as the Hilbert Transform will retrieve the modulating frequency in the resulting envelope signal, and as a result, indicate the faulty component.

Consider a vibration signal due to periodic impact on a structure with resonant frequency w_c. Assuming that, each impact will ring the structure at a dominant frequency w_c, and the waveform will repeat itself with a repetition rate equal to the periodic impact rate. In the time domain, this signal can be modeled as an envelope signal e(t) multiplied by a carrier signal as:

$$x(t) = e(t) \cos[\, w_c t+f(t)] \qquad (5.1)$$

The envelope represents the periodic impact with a dominant frequency w_e, and the carrier represent the structural response at natural frequency wc. In the frequency domain, its PSD will have a side-band structure with left, center and right sideband. Notice that, it is not sufficient to determine the envelope signal just from this measured signal alone. This signal x(t) can be treated as the real part of a complex signal whose imaginary part y(t) is equal to the envelope e(t) multiplied by this sine wave with identical phase drifting. If such imaginary part signal can be found, then the envelop signal is just equal to the amplitude of the complex signal. It can be shown that, if the frequency of the envelop signal is much smaller then the carrier frequency w_c, then this imaginary part signal y(t) is just equal to the Hilbert transform of the real part signal x(t), which is equivalent to perform a 90 degree phase shift of x(t).

$$y(t) = HT\,[x(t)\,] = x(t) * h(t) = IFT\,[\,X(w)\,H(w)] \qquad (5.2)$$

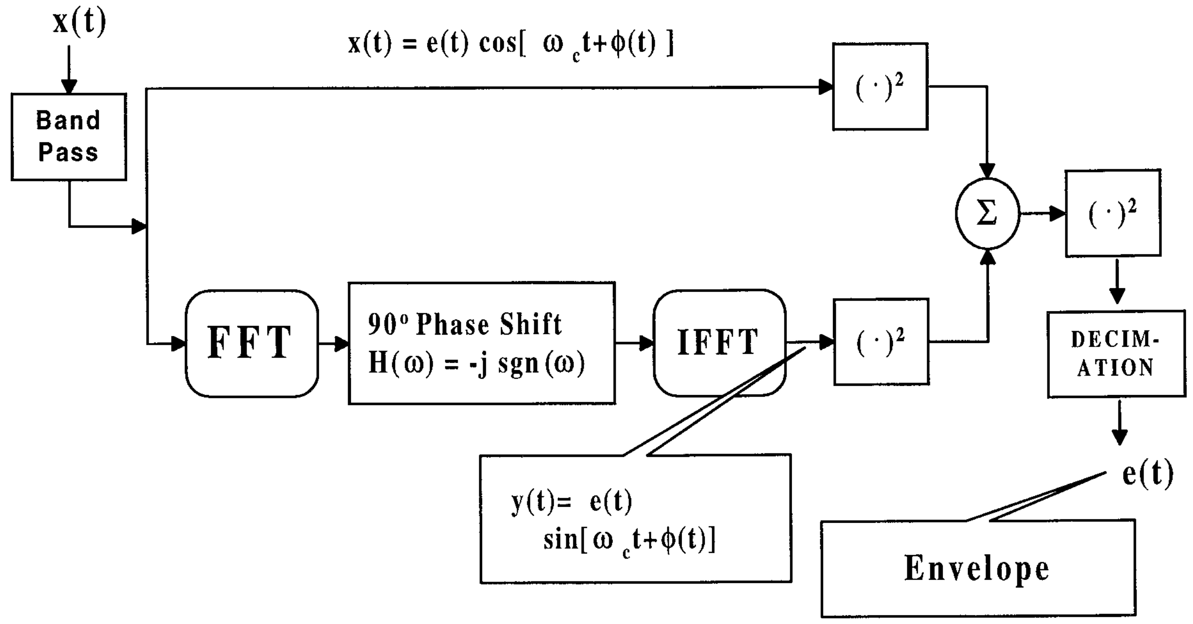
Where : H(w) = Transfer Function of 90 degree phase shiftor = -j sgn(w) (5.3)
HT = Hilbert Transform
IFT = Inverse Fourier Transform

Therefore, in the time domain, y(t) is equal to the convolution of x(t) and the impulse response function (IRF) of the HT h(t). In the frequency domain, Y(w) is simply equal to X(w)H(w). The transfer function H(w) will have unity amplitude with 90 degree phase at positive frequency and -90 degree phase at negative frequency. Therefore, the imaginary signal can be estimated by taking the FFT of the real part signal x(t), multiplied by the transfer function H(w) of HT, and then followed by an IFFT to generate the imaginary part signal y(t). The envelop signal then can be recovered through the amplitude of x(t) & y(t).

2.2.5.2. Algorithm/Code of Envelope Analysis Program

Figure 5.1 shows the signal processing algorithm for high frequency envelope analysis using Hilbert transform technique. The input measurement signal x(t) after bandpass filtered can be modeled as a low frequency envelope signal e(t) multiplied by a high frequency carrier signal with a center frequency wc and phase drifting f(t).. The purpose here is to recover the low frequency envelop signal e(t) from the high frequency measurement signal. The major function of the Hilbert Transform within this algorithm is to find its corresponding imaginary part signal from the real part measurement signal. Since the amplitude of the transfer function of HT is one across the entire frequency, and the phase of the transfer function of HT is -90 degree at all positive frequency and 90 degree phase at all negative frequency. Therefore, the desired imaginary part signal y(t) is simply equal to the HT of the real part signal x(t). This is equivalent to performing a 90 degree phase shifting of x(t). The HT operation can be effectively implemented in the frequency domain, as shown in the algorithm, and the envelop signal is obtained from the amplitude of the complex analytical signal z(t).

Figure 5.2 shows an example of the envelope analysis. The raw time history shown in figure 5.2(a) is taken from a dynamic pressure measurement during an ATD HPOP Inducer test at NASA/MSFC's ITL (Inducer Test Leg) water flow test facility. The "bursting" waveform in the raw time history is generated due to cavitation-generated bubble collapsing process. Such cavitation phenomenon can be detected by performing spectral analysis on the envelope signal of the time history instead of the time history itself. Figure 5.2(b) shows the resulting envelope signal which "envelopes" the original time history. The periodicities of the bursting waveform in the original time history can be detected from the PSD of the envelope signal. Figure 5.3(a) shows the RAW PSD of the original signal which has been bandpass filtered from 6 KHz to 10 KHz. Figure 5.3(b) shows the resulting ENVELOPE PSD, where strong 4N component (280 Hz) along with its harmonics are recovered as an indication of full-blade (4-blade) cavitation condition.


x(t)
x(t) = e(t) cos[ω_c t+φ(t)]
Band Pass
(·)²
Σ
(·)²
FFT
90° Phase Shift H(ω) = -j sgn(ω)
IFFT
(·)²
DECIM-ATION
y(t)= e(t) sin[ω_c t+φ(t)]
e(t)
Envelope

- Measured Bandpass Signal $x(t) = e(t)\cos[\omega_c t + \phi(t)]$

  90° Phase Shifted Signal $y(t) = e(t)\sin[\omega_c t + \phi(t)]$

=> Analytical Signal $z(t) = x(t) + j\,y(t) = e(t)\exp j[\omega_c t + \phi(t)]$

- $H(\omega)$ = Transfer Function of 90° Phase Shiftor = $-j\,\mathrm{sgn}(\omega)$

$$H(\omega) = \begin{cases} \exp(-j\,\pi/2) & \text{for } \omega > 0 \\ \exp(+j\,\pi/2) & \text{for } \omega < 0 \end{cases}$$

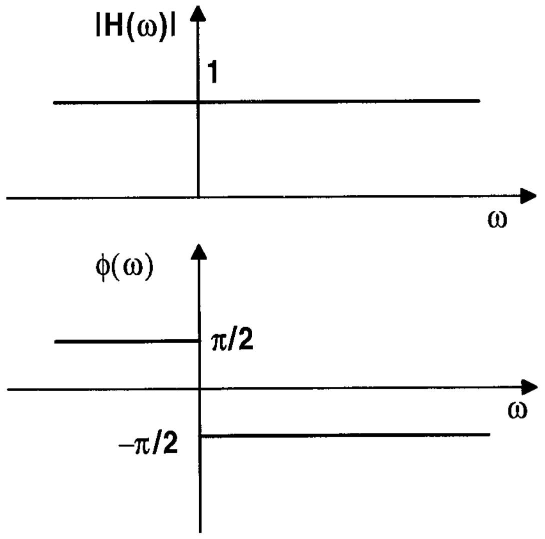


Figure 5.1 Envelope Analysis Algorithm using Hilbert Transform

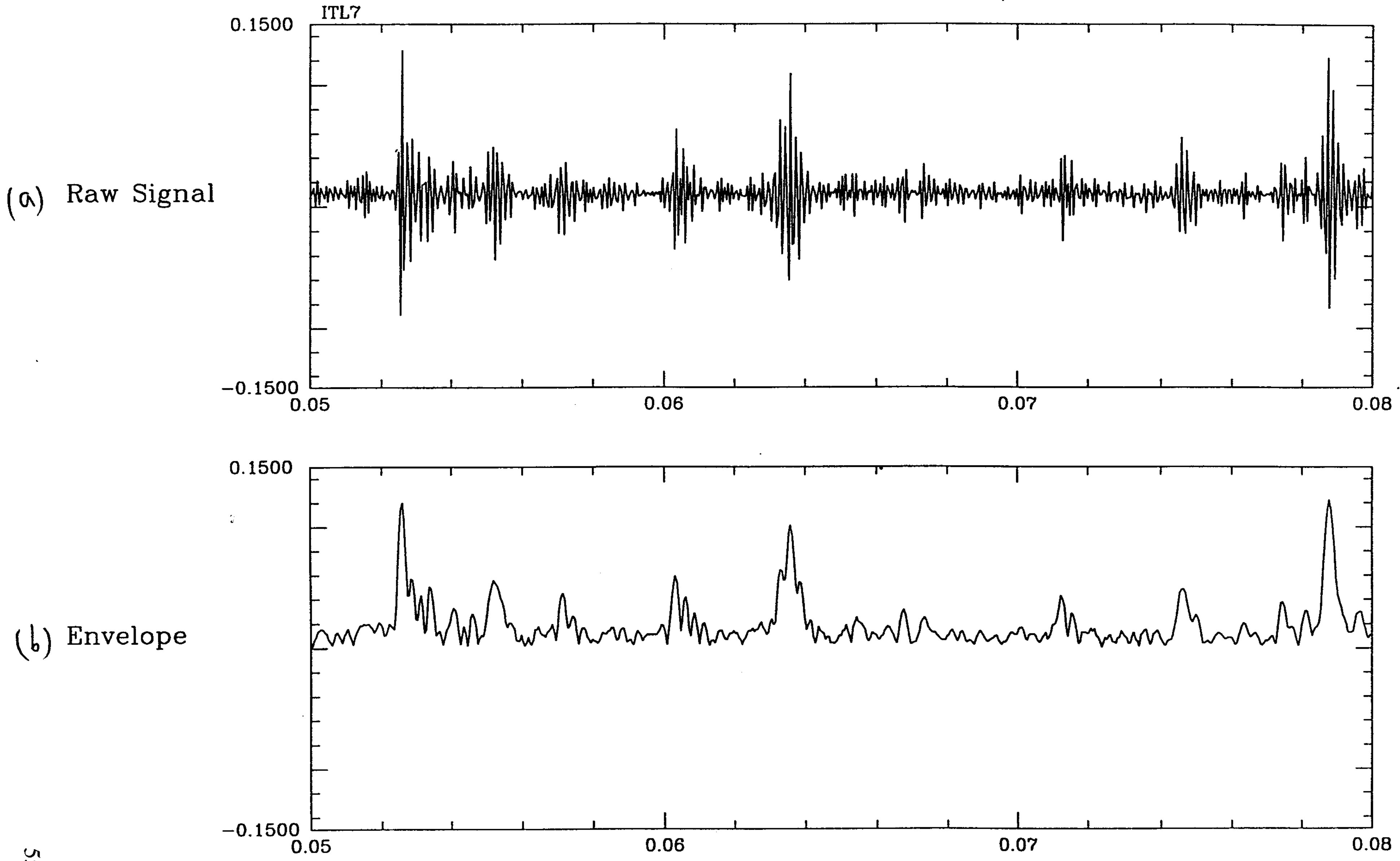


Figure 5.2 Example of Envelope Analysis using I T L Test Data

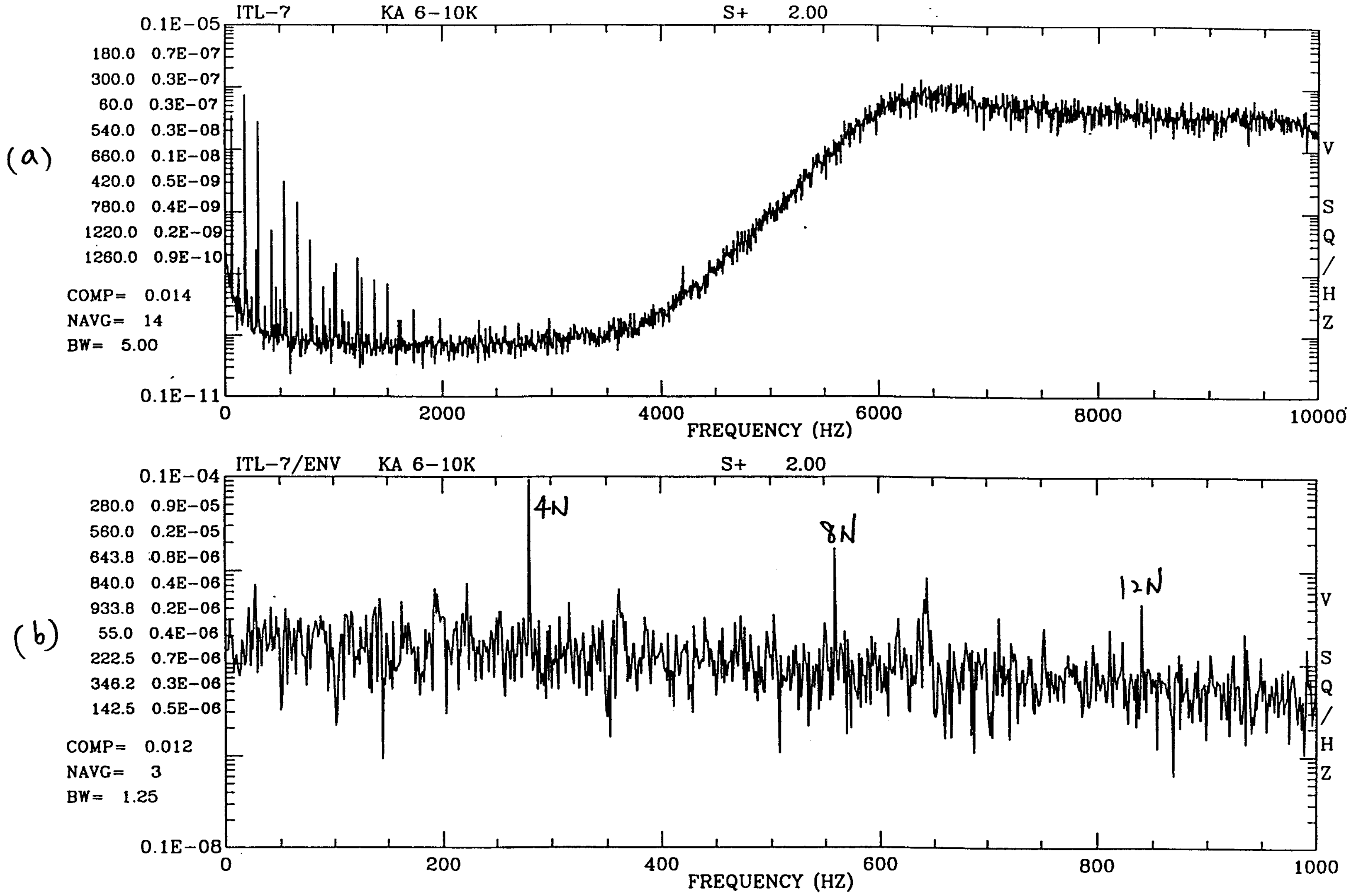


Figure 5.3 (a) RAW PSD (b) ENVELOPE PSD of ITL test data

2.2.5.3 DSP Code Overview of envelope analysis using Data Translation's DSP-EZ programming language running on C40 DSP

The complete codes for the envelope analysis program running on C40 DSP using Data Translation's DSP-EZ programming language is listed below.

```
''
'+------------------------+
'|    Envelop.DSP         |
'+------------------------+
'
'         ***********************************************
'         ****  DSP Basic Program Global Declaration ****
'         ***********************************************
        Option Explicit
'+----------------+
'| Constants |
'+----------------+
        Const CTRLchan = 0      'channel for getting frequency & filter info.  from user
        Const Userchan=1           'channel for getting user defined parameters
        Const FILTChan = 16

        Const FILTERPOINTS = 0
        Const CUTOFFLower = 1
        Const CUTOFFUpper = 2
        Const FILTER_SIZE =201

'+-----------+
'| Arrays |
'+------------+
        Dim ADbuff As FloatArray
        Dim ImpulseResponse      As FloatArray
        Dim BPFilter0 As FloatArray                  'Global for ProcessControl
        Dim BPFilter1 As FloatArray
        Dim TimeData As FloatArray
        Dim FilterData As FloatArray
        Dim PSD As FloatArray
'+---------------+
'| Variables |
'+---------------+
        Dim  Frequency As Single
        Dim  BUFFSIZE  As Integer
        Dim  FFT_SIZE   As Integer            'FFT_SIZE >= BUFFSIZE
        Dim  nmax As Integer
        Dim Decimation As Integer

'          **********************
'          **** Function Main ****
'          **********************
'

Function Main()
```

```
'+------------------+
'| Declaration |
'+------------------+
Dim cmdbuff  As FloatArray
Dim DAdata As FloatArray
Dim Channel As FloatArray
Dim  FWArray As FloatArray
Dim TFWArray As FloatArray
Dim SFW As FloatArray
Dim Sum As FloatArray
Dim T As FloatArray

Dim SCount As Integer
Dim NumOfAVG As Integer
Dim AVG2 As Integer
Dim ADREADY As Integer
Dim NumPSDChd As Integer
Dim i, j, k, n,  Length, start As Integer
Dim bw, ref, Fs As Single

FFT_SIZE =2048
BUFFSIZE =2048
nmax=FFT_SIZE/2 +1
Start=0
Length=nmax
NumOfAvg=1
AVG2=1
Decimation = 1
Channel =GenLine(0,0,16)


'+------------------------------------------+
'| get the initial A/D frequency |
'+------------------------------------------+
Do While Not Host.GetDataReady(CTRLchan)
Loop
cmdbuff = Host.GetBuffer(CTRLchan)
Frequency = cmdbuff(0)
FFT_SIZE = cmdbuff(1)

If Host.GetDataReady(UserChan) Then
        cmdbuff=Host.GetBuffer(UserChan)
        NumPSDChd = cmdbuff(0)
        For i = 1 To NumPSDChd
                Channel(i-1) = cmdbuff(i)
        Next i
End If
'+-----------------------------------+
'| Define A/D parameters |
'+-----------------------------------+
AD.Frequency =Frequency
AD.ClockSource = DTDSP_ICLK
AD.TriggerSource = DTDSP_ITRG
AD.NumChannels = 8
AD.AcqMode = DTDSP_NORMAL
AD.InternalChannelBufferSize = BUFFSIZE * 4
'+----------------------------------+
'| Define D/A parameters |
'+----------------------------------+
'DA.Frequency = Frequency
'DA.ClockSource = DTDSP_ICLK
'DA.TriggerSource = DTDSP_ITRG
'DA.NumChannels = 1
'DA.AcqMode = DTDSP_Normal
'DA.InternalChannelBufferSize = BUFFSIZE*3
'+----------------------------------------------+
'| initialize global array to all zeros |
'+----------------------------------------------+
ImpulseResponse = GenLine(0,0,BUFFSIZE)
ImpulseResponse(BUFFSIZE/2) = FFT_SIZE
Do While Not Host.GetDataReady(CTRLchan)
Loop
'+---------------------+
'| initialize filter |
'+---------------------+
ProcessControl()
'+------------------------------+
'| start acquiring data |
'+------------------------------+
AD.ChannelType=DTDSP_DI
AD.Filter=AD.Frequency / 2
AD.Start()
'           ****************
'           * Main looping *
'           ****************
'+------------------------------------------------+
'| acquire parameters from  buffer
'+------------------------------------------------+
Do While   1
                If Host.GetDataReady(Userchan) Then
                        cmdbuff=Host.GetBuffer(Userchan)
                        NumOfAVG=cmdbuff(0)
                        Start=cmdbuff(1)
                        Length = cmdbuff(2)
                        Decimation = cmdbuff(3)
                        Sum = Sum * 0
                        SCount = 0
                End If
'           +------------------------------------+
'           | Extract data from buffer
'           +------------------------------------+
                Sum = Sum * 0
                SFW = SFW * 0
                If NumOfAVG < Decimation Then
                        AVG2 = 1
                Else
                        AVG2 = NumOfAVG/Decimation
                End If
                For j = 1 To AVG2
                        For i = 1 To Decimation
                                ADbuff = AD.Scan(FFT_SIZE)
                                TimeData = Extract(Adbuff, FFT_SIZE, Channel(0) *
FFT_SIZE)
                                FilterData = Filter(BPFilter0, TimeData)
                                FilterData = HilbertTransform(FilterData)
                                TFWArray =ApplyDecimation(Channel(0), FilterData )
                                If  i = 1 Then
                                        FWArray = TFWArray
                                Else
                                        FWArray = FWArray  & TFWArray
                                End If
                                PSD = ApplyFFT(TimeData)
                                Sum = Sum +PSD

                        Next i
                        SFW=SFW +  Abs(FWArray)
                Next j
                If NumOfAVG + Decimation < 4 Then
                        T = AD.Scan(FFT_SIZE)
                End If
                If Host.PutDataReady(Channel(0)) Then
                        Host.PutBuffer(Channel(0)) = Sum/NumOfAVG
                End If
                If Host.PutDataReady(Channel(0) + 8) Then
                        Host.PutBuffer(Channel(0) + 8) = ApplyFFT(SFW/AVG2)
                End If
                If (Host.GetDataReady(CTRLchan)  And (AD.DataReady(0) <
FFT_SIZE)) Then
                        ProcessControl()
                End If
Loop    'End of Do While loop

End Function
'
'          *********************************
'          **** Function ProcessControl ****
'          *********************************
'

Function ProcessControl()
Dim ctrlbuff As FloatArray
Dim FilterDataTemp As FloatArray
Dim FFTDataTemp As ComplexArray
Dim ready As Integer
Dim lowercut, uppercut As Single
Dim length As Integer
ready = TRUE
Do While  ready
  ctrlbuff = Host.GetBuffer(CTRLchan)
  ready = Host.GetDataReady(CTRLchan)
Loop
'+--------------------------------------------+
'| generate a band pass filter object |
'+----------------------------------------------------+
length   = ctrlbuff(FILTERPOINTS)
lowercut = ctrlbuff(CUTOFFLower)
uppercut =ctrlbuff(CUTOFFUpper)
BPFilter0 = InitBandPassFilter(lowercut,uppercut,Frequency,length)
BPFilter1 = InitBandPassFilter(lowercut,uppercut,Frequency,length)
FilterDataTemp = Filter(BPFilter0,ImpulseResponse)
FFTDataTemp = fft(FilterDataTemp)
FilterDataTemp = dB(FFTDataTemp)
FilterDataTemp = FilterDataTemp - 20*log10(FFT_SIZE) + 6.0
'+-------------------------------------+
'| Send data to host for filter plotting |
'+-------------------------------------+
Host.PutBuffer(FILTchan) = FilterDataTemp
End Function
'
'
'          *********************************************
'          **        Function ApplyFFT( )                **
'          **                                                **
'          **        Calls:  Window, fft, dB             **
'          *********************************************
'
Function ApplyFFT(SomeData As FloatArray) As FloatArray
Dim Temp1 As FloatArray
Dim Temp2 As ComplexArray
Dim BHWnd As FloatArray
Dim dBScale As Single

        dBScale = -20 * log10(10*FFT_SIZE) + 6.0
        BHWnd=GenWindow(BLACKMAN_HARRIS_61,FFT_SIZE)
        Temp1=Window(BHWnd, SomeData)
        Temp2=fft(Temp1)
        ApplyFFT=dB(Temp2)+dBScale
End Function
'          ********************************
'          **        Function Conjugate      **
'          ********************************
Function Conjugate( A As Complex) As Complex
Dim r1, r2 As Single
        r1 = Real(A)
        r2 = -Imag(A)
        Conjugate = MakeComplex( r1, r2)
End Function
'          ******************************************************
'          **          Function ApplyDecimation( Channel_ID, InArray)
'          **          return decimated partial array
'          ******************************************************
Function ApplyDecimation(Channel_ID As Integer,  InArray As FloatArray ) As
FloatArray
Dim TArray As FloatArray
Dim PArray As FloatArray
Dim i,  Size   As Integer
```

```
        Size = FFT_SIZE/Decimation
        TArray = GenLine( 0,0, FFT_SIZE)
        PArray = GenLine (0,0, Size)
        TArray = InArray

        If Decimation= 1 Then
                ApplyDecimation  = TArray
        Else
                For i= 0 To Size-1
                        PArray(i) = TArray(Decimation * i)
                Next i
                ApplyDecimation = PArray
        End If
End Function
'
'               **********************************************
'               **              HilbertTransform(InArray As FloatArray)        **
'               **      FFT(InArray) Then HilbertTransfor Then IrFFT           **
'               **      InArray = FilterData
'               **********************************************
Function HilbertTransform(InArray As FloatArray) As  FloatArray
Dim XX As ComplexArray
Dim YY As ComplexArray
Dim RR As FloatArray
Dim ZZ As FloatArray
Dim Xr As FloatArray
Dim Xi As FloatArray
Dim X2 As FloatArray
Dim Y2 As FloatArray
Dim i As Integer

        ZZ = GenLine(0,0, FFT_SIZE)
        XX = fft(InArray)
        Xr =- Real(XX)
        Xi = Imag(XX)
        YY = MakeComplex(Xi, Xr)
        RR = irfft(YY)
        X2 = InArray * InArray
        Y2 = RR * RR

        For i = 0 To FFT_SIZE - 1
                ZZ(i) = (X2(i) + Y2(i)) ^ .5
        Next i
        HilbertTransform= ZZ
End Function
```

2.2.6 GHC - Generalized Hyper-Coherence Analysis

2.2.6.1 General background, algorithm and simulation example of Generalized hyper-coherence function

In a rotor system, the fundamental shaft rotational component can be treated as a driving source which drives some other rotational mechanisms and generates new spectral components at different frequencies such as cage frequency or outer ball pass frequency associated with bearing defect. These two different frequency components are correlated to each other in a particular nonlinear fashion due to its kinematic geometry. Since the frequency ratio of these two components is arbitrary, the nonlinear higher order cumulant spectral analysis, such as bi-spectrum and tri-spectrum would not be able to identify such correlation. The Generalized hyper-coherence (GHC) can identify their correlation by correlating their instantaneous frequency signal since the rate of change of their instantaneous phases should be synchronize with each other. Consider a gear box example with non-integer gear ratio R. Assuming that the first gear with M1 teeth is driving the second gear with M2 teeth. The angular displacements (phases) at frequencies N1 and N2 will be proportional to each other, and these two spectral components are correlated in such a way that their phase variation are synchronous to each other. Assuming perfect gear mesh with no broken or worn gear tooth. If gear 1 goes through one cycle of rotation with angular displacement a(t), then gear 2 will finish R cycle of rotation with angular displacement R*a(t). In other word, their angular displacement will be proportional to each other by a factor of R. As a result, certain coherent phase relationship would exist between the spectral components at frequencies N1 and N2. However, in general the gear ratio R (M1/M2) would be an non-integer number. Therefore, such phase coherence become very difficult to identify due to the ambiguity introduced by the Riemann surface phase wrapping in the phase estimation. Based on this observations, the GHC was developed to identify such phase coherence by correlating the rate of change of their phases, which is also called the instantaneous frequency. By taking the rate of change of phase, the ambiguous phase term will be eliminated, and their phase correlation will now be reflected in the frequency domain as frequency synchronization.

If a vibration signal is treated as an FM signal with different spectral components at different center (carrier) frequencies. Assuming that, there is some intelligence being frequency modulated in the vibration signal as the instantaneous frequency at these different carrier frequency. In order to recover the intelligence, we need to demodulate the FM signal to estimate its instantaneous frequency signal. A narrow band random process can be modeled as a sine wave with slowly varying amplitude A(t) and phase p(t):

$$x(t) = A(t) \cos[\ \omega_c t + \varphi(t)\ ] \qquad (6.1)$$

The instantaneous frequency Wi(t) is defined by:

$$\omega_i(t) = \frac{d\,\varphi(t)}{d\,t} \qquad (6.2)$$

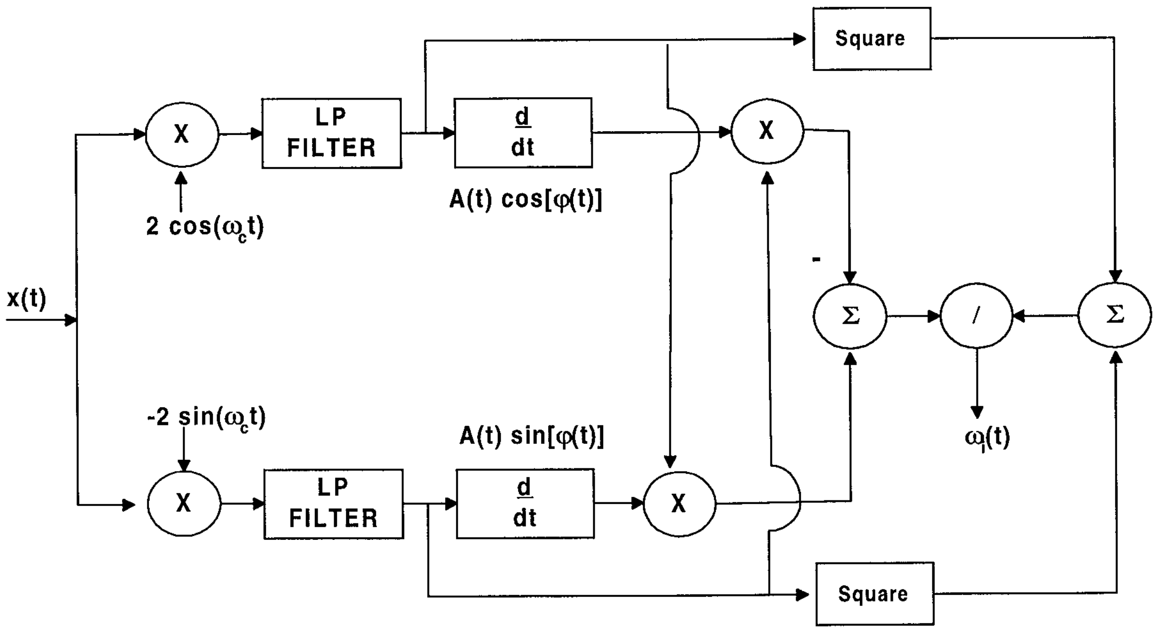


Figure 6.1 Frequency Demodulation Using Synchronous Receiver

Figure 6.1 shows the block diagram of an FM demodulator using the Synchronous Receiver method. The input signal may contain several spectral components (FM signal) at different carrier frequencies. Here, the 90 degree phase shifting is performed through the multiplication of a locally generated sinusoid and cosine waves (synchronous receiver) at the desired carrier frequency wc. This will effectively shift the spectral component from the carrier frequency wc to both zero frequency and two times of that frequency. A low-pass filter is then used to remove any high frequency component. The output of the low-pass filters represent the slowly varying amplitude and phase (i.e. A(t) exp[j φ(t)]). The instantaneous frequency signal wi(t) then can be obtained by,

$$w_i(t) = \dot{\phi}(t) = \frac{u(t)\dot{v}(t) - \dot{u}(t)v(t)}{u^2(t) + v^2(t)} \tag{6.3}$$

Where u(t) = A(t) cos [φ(t)]
v(t) = A(t) sin [φ(t)]

For the purpose of identifying frequency synchronization for machinery diagnostics. Just imagine that the vibration signal is composed of several FM signal modulated at different carrier frequency. Each component is then passing through an FM-demodulator to generate a series of new random signal representing the instantaneous frequency signal at each carrier frequency. A linear correlator is then used to correlate among these instantaneous frequency signal with a specified reference carrier frequency. Their correlations are then summarized in the frequency domain to generate the GHC, which indicates the degree of frequency lock-in between the reference component and any other spectral component at arbitrary frequency. Figure 6.2 shows the flow diagram of the GHC program.

GHC Menu : NDEC=GHC Decimation Cycle
for each GHC tracking component, Request 3 parameters:
(1) Fc (2) LPBW (3) Filter Order=Order

- Fs= sampling frequency in Hz
- NN=block size (or buffersize)
- Fc=Center frequency in Hz
- LPBW = low pass bandwidth in Hz
- Define Pinc=2p*Fc*NN/Fs
- Define MM=NN/NDEC (Note: NN/NDEC must be integer)

<u>Program start (or re-start):</u>

- *initial phase by setting phs=0*
- *initial z1(0) =z2(0)=0*
- greate two filter object filter1 & filter2 with identical filter setting:
  lower cutoff frequency = 0
  higher cutoff frequency = LPBW
  filter order = order

1. get one block of data raw(t) array
2. generate (cos + j sin) array using GENCOSCOMPLEX with:
   Phase: phs = phs + Pinc
   frequency = Fc
   Sampling frequency =Fs
   Arraysize = NN
   break complex (cos +j sin) array into xx() & yy() array for real & imaginery pary
3. generate xx= 2*raw*xx
   yy= -2*raw*yy
4. use filter1 to perform low-pass filter of xx: xx=lowpass filter1 of xx
   use filter2 to perform low-pass filteri of yy: yy=lowpass filter2 of yy
5. average/decomation:
   Array z1 (1 to MM) =xx(1 to NN) with NDEC point average and decimation
   Array z2 (1 to MM) =yy(1 to NN) with NDEC point average and decimation
6. Calculate Instantaneous frequency Array IFQ() using equation (3)

$$w_i(t) = \dot{\phi}(t) = \frac{u(t)\dot{v}(t) - \dot{u}(t)v(t)}{u^2(t) + v^2(t)}$$

Note for New Phase update: T= time period of one clock
1st block = $\cos(\omega_c t + f_0)$
2nd block = $\cos[\omega_c t + (w_c T + f_0)]$
=> phase increment = $\omega_c T$ = 2p*Fc*NN/Fs

Figure 6.2 GHC Program Flow Diagram

<u>2.2.6.2. Simulation Example for GHC Analysis Program</u>

A set of simulation data has been generated which will be used by both ASRI and Vanderbilt to test and evaluate the performance of the GHC program in the MPP system. The simulation signal is composed of three spectral components:

1. Reference component
2. Correlated Component
3. Uncorrelated Component

The first component represents the Synchronous frequency component at 1000 Hz. The second one represents a Sync-related component at 2600 Hz whose instantaneous phase/frequency is synchronous to the reference component. The third component represents an independent component at 3800 Hz with an independent instantaneous phase/frequency variation. The sampling frequency of the simulation is 10240 Hz. A total amount of 40 seconds of data was generated. Figure 6.3 shows the PSD of the simulation signal with 25 average of 4096-point FFT block. Its is composed of three spectral components marked by "R", "C", and "U", which represent the Reference, Correlated and Un-correlated component. Figure 6.4(a), (b) and (c) show the demodulated instantaneous frequency signal with carrier frequencies set at "R", "C", and "U" respectively. Strong frequency correlation or synchronization can be clearly visualized between the reference and correlation component, while no correlation between the reference and uncorrelated component.

<u>SSME Test Example for Generalized Hyper-Coherence Analysis</u>

An example encountered from SSME hot firing test can best demonstrate the practical application of hyper-coherence for machinery diagnostics. Figures 6.5 shows the PSD taken from a HPFTP accelerometer during SSME test 901-471. The peak marked "N" is corresponding to the fundamental RPM component, and the remaining major peaks are its apparent harmonics 2N, 3N and 4N. Notice that, the PSD shows an abnormally high 3N PSD amplitude. The hyper-coherence analysis as discussed in the last month progress report was able to identify this spectral peak at the 3N frequency is not a true third harmonic. The signature turned out to be the so-called pseudo 3N component which is due to some independent source in the HPFTP not related to the machinery rotational process. Figure 6.6 shows the instantaneous frequency signal for (a) N, (b) 4N, and (c) 3N components. The IF signal of 4N component is highly correlated with N, while the IF signal of 3N is only weakly correlated with N.

The complete software codes in a conversion format from ASRI's DSP system to Vanderbilt MGA Model Based Programming environment will be developed and tested in the next reporting period. Due to the schedule slippage of Vanderbilt tasks in converting the other programs (e.g. Tri-coherence, bi-coherence, hyper-coherence, etc.) to the MPP system, ASRI has requested a no cost extension of this contract until 31 August 1996 to the Contracting Officer (COR). No other major problem which would impact the contract cost has been encountered.

2.2.6.3 DSP Code Overview of GHC function using Data Translation's DSP-EZ programming language running on C40 DSP.

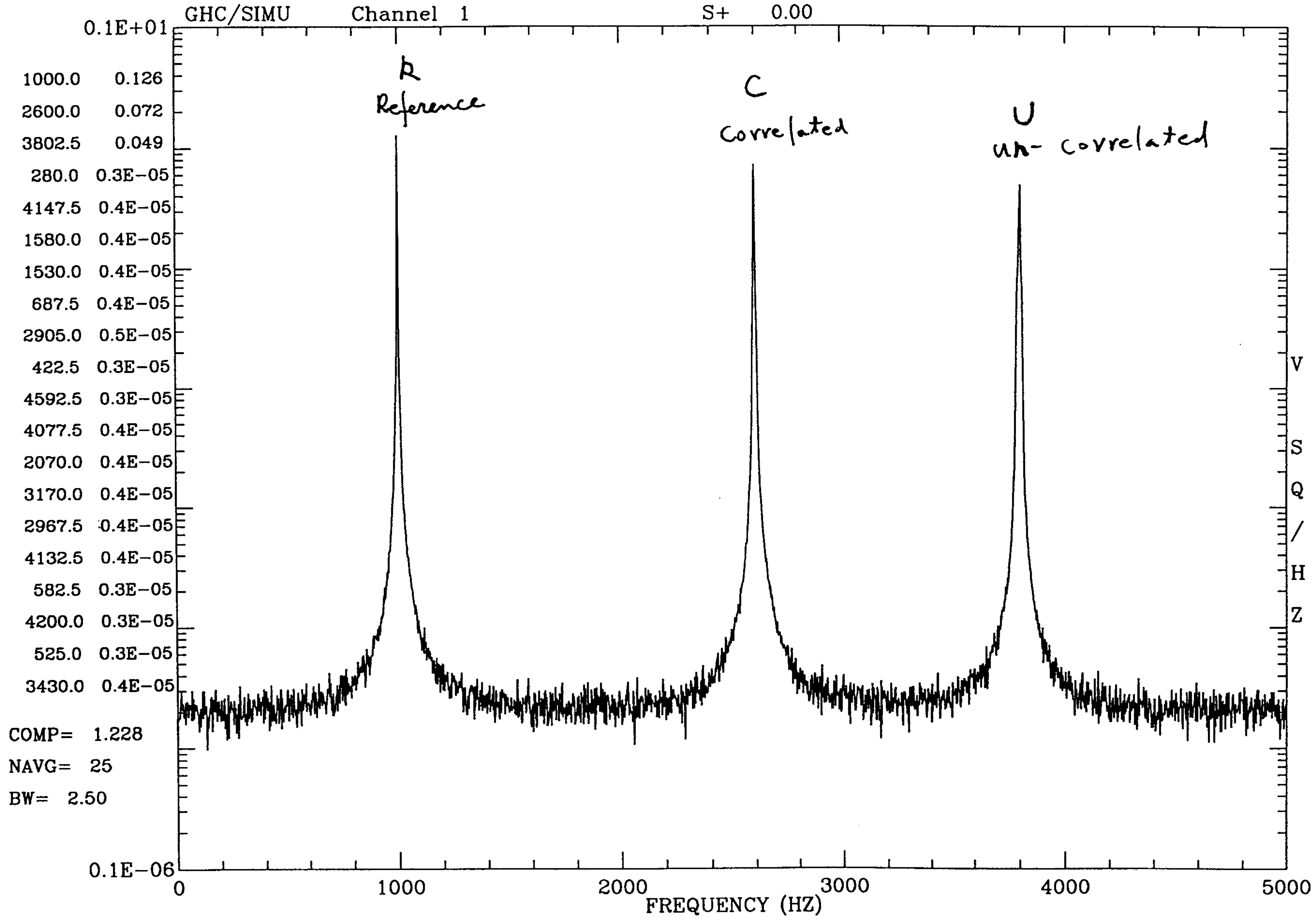


Figure 6.3 PSD of Simulation Signal

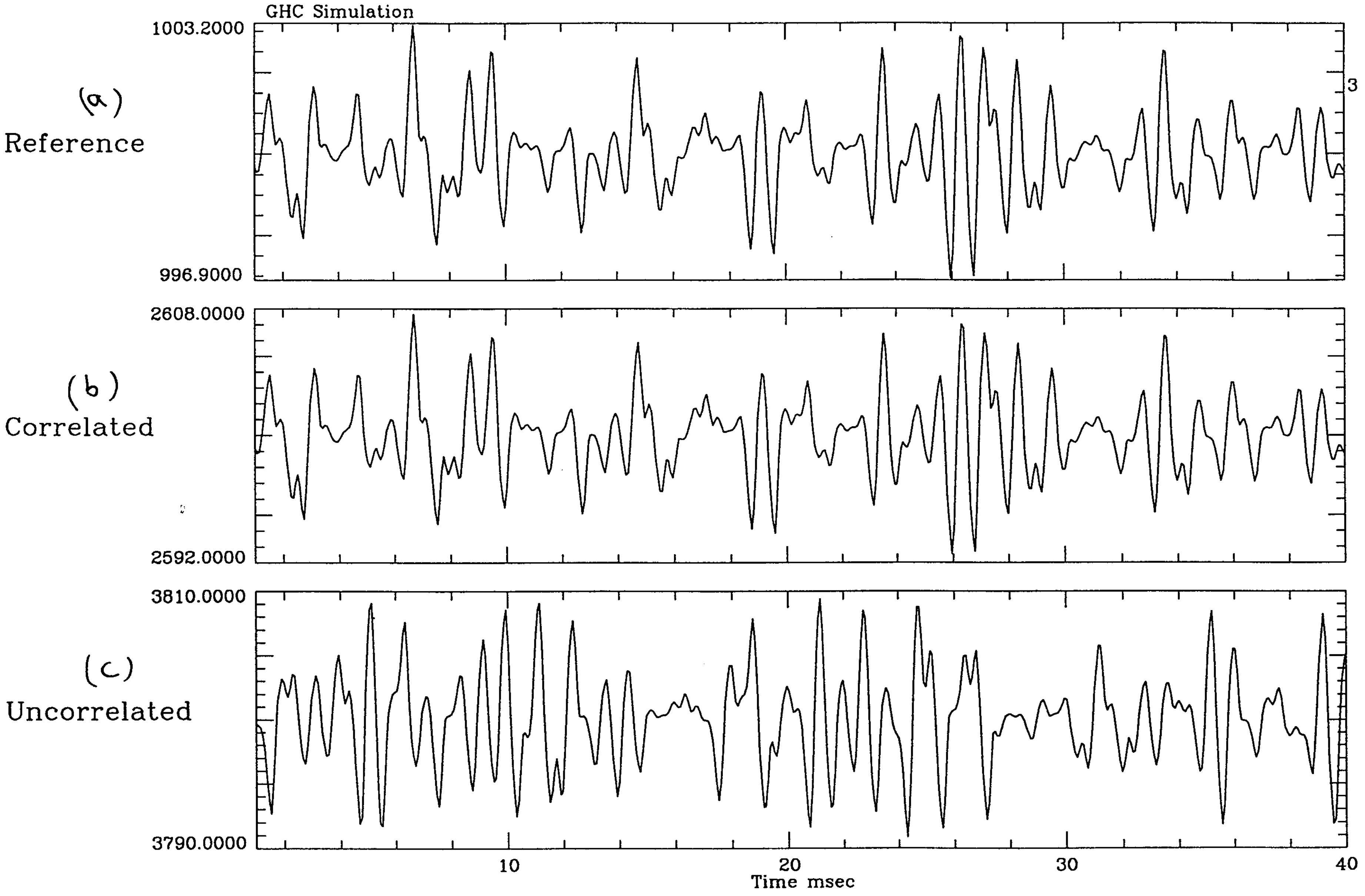
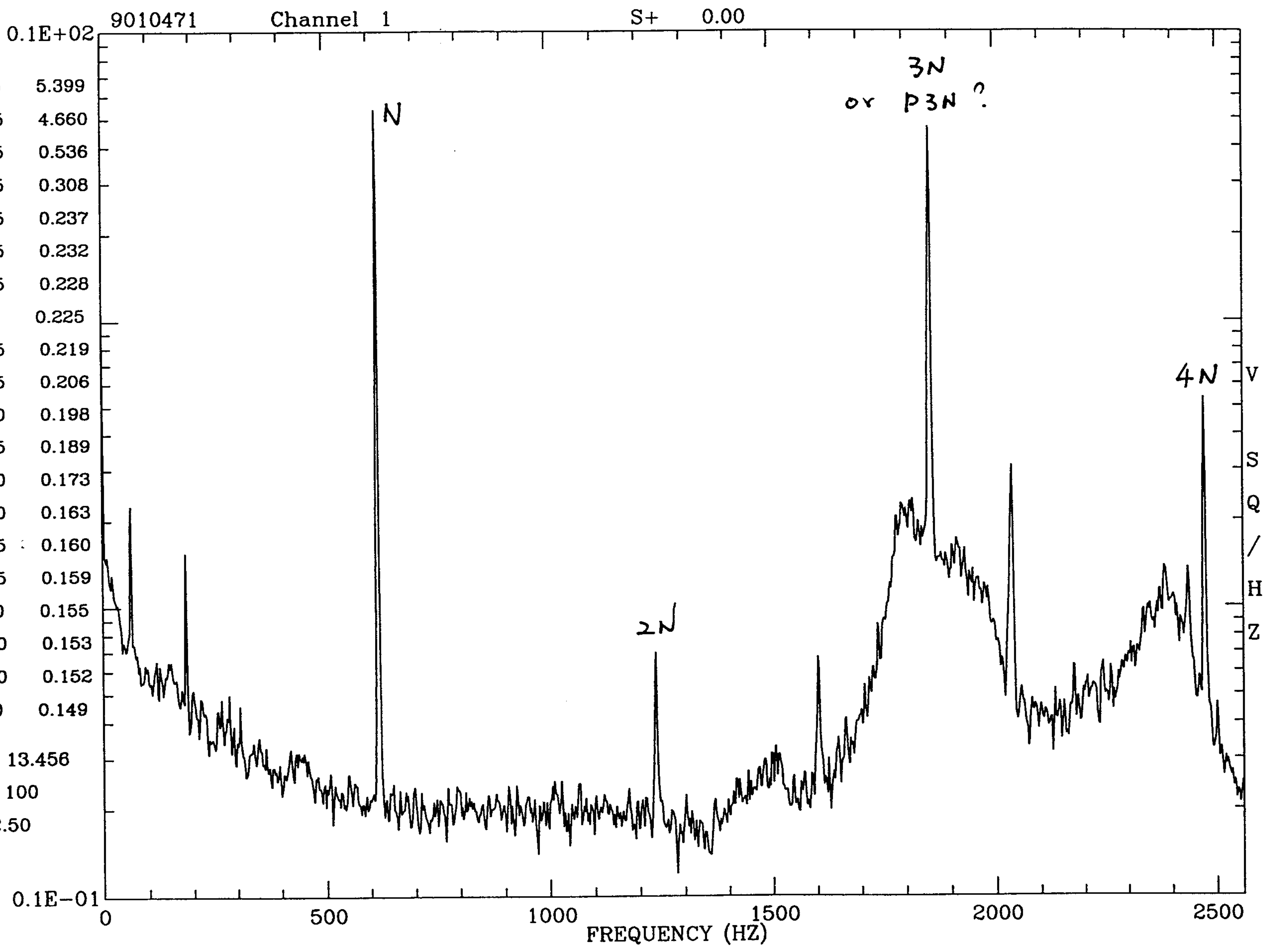


Figure 6.4 cor(1,2)=1.00 cor(1,3)=-.09 cor(2,3)=-.09



617.5	5.399
1857.5	4.660
2472.5	0.536
2037.5	0.308
1817.5	0.237
1812.5	0.232
1792.5	0.228
62.5	0.225
1802.5	0.219
1782.5	0.206
1830.0	0.198
1837.5	0.189
1915.0	0.173
1905.0	0.163
1932.5	0.160
1922.5	0.159
185.0	0.155
1880.0	0.153
1890.0	0.152
10.0	0.149

COMP= 13.456
NAVG= 100
BW= 2.50

Figure 6.5 PSD of HPFP

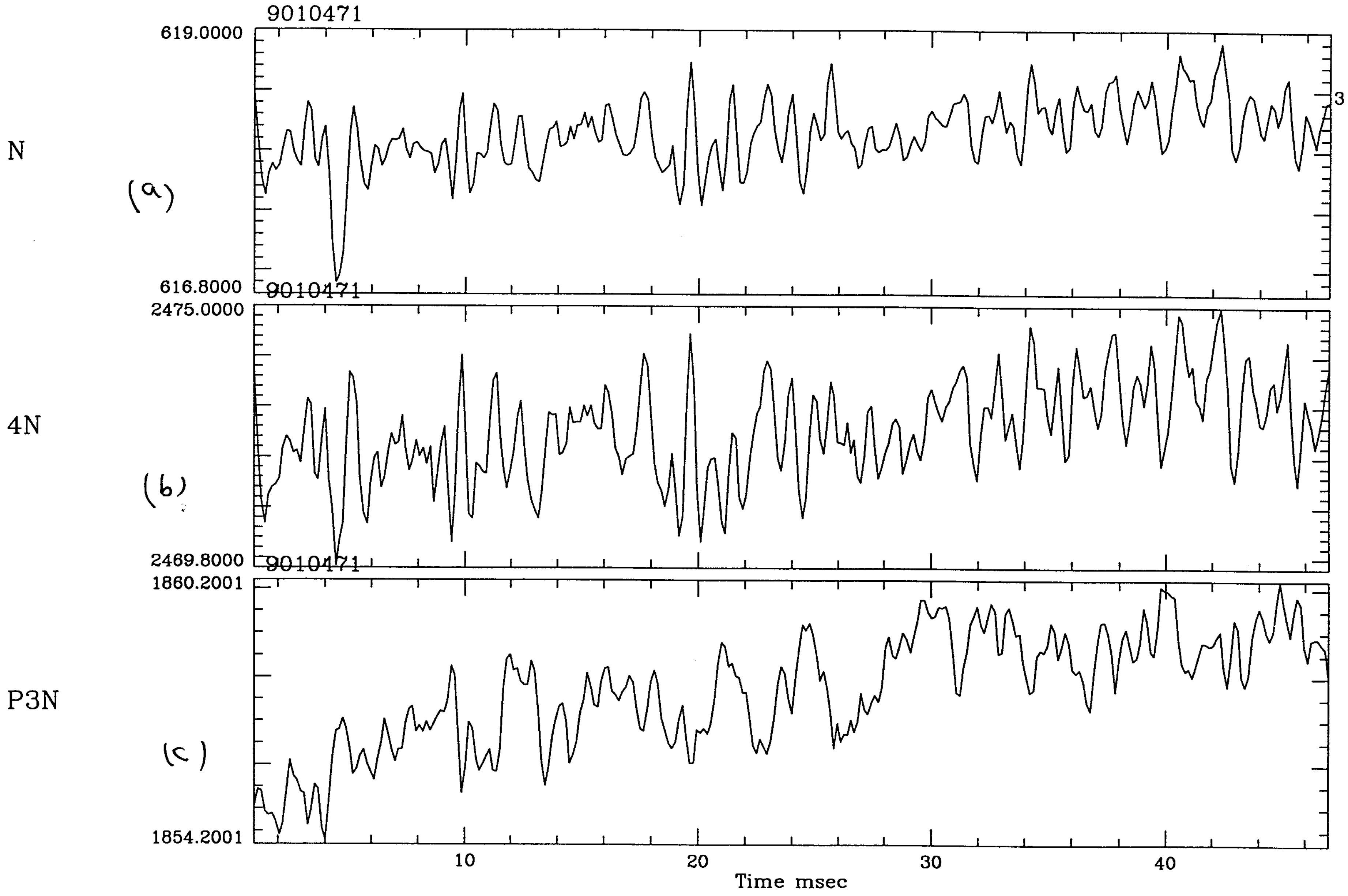


Figure 6.6 cor(1,2)=0.94 cor(1,3)=0.25 cor(2,3)=0.23

```
+--------------------------------+
'|    Ghc.DSP        |
'+--------------------------------+
'Originated from topo.dsp
'GHC redone on 7/12/96
'Revised on 7/25/96
'          ***********************************************
'          ****  DSP Basic Program Global Declaration  ****
'          ***********************************************
        Option Explicit
'+-----------------+
'| Constants |
'+-----------------+
        Const CTRLchan = 0     'channel for getting frequency & filter info.  from user
        Const Userchan=1       'channel for getting user defined parameters
        Const GHchan=2         'channel for getting Fc, LPBW, and Order
        Const DiskChan=3
        Const FILTChan = 9
        Const FILTERPOINTS = 0
        Const CUTOFFLower =0
        Const CUTOFFUpper =2
        Const FILTER_SIZE =201
'+-----------+
'| Arrays |
'+------------+
        Dim ADbuff As FloatArray
        Dim ImpulseResponse As FloatArray
        Dim BPFilter0 As FloatArray          'Global for ProcessControl
        Dim BPFilter1 As FloatArray
        Dim TimeData As FloatArray
        Dim PSD As FloatArray

        Dim IFQ As FloatArray                'For GHC
        Dim  U0 As ComplexArray
        Dim  U1 As ComplexArray
        Dim  U2 As ComplexArray
        Dim  U3 As ComplexArray
        Dim Filter0x As FloatArray
        Dim Filter0y As FloatArray
        Dim Filter1x As FloatArray
        Dim Filter1y As FloatArray
        Dim Filter2x As FloatArray
        Dim Filter2y As FloatArray
        Dim Filter3x As FloatArray
        Dim Filter3y As FloatArray
        Dim  LPBW As FloatArray   'low pass bandwidth in Hz
        Dim  Fc As FloatArray
        Dim  Order As FloatArray
'+----------------+
'| Variables |
'+----------------+
        Dim  Frequency As Single
        Dim  BUFFSIZE  As Integer
        Dim  FFT_SIZE   As Integer
        Dim  nmax As Integer

        Dim  MM As Integer  'For GHC
        Dim  NDEC  As Integer
        Dim RCFactor As Single
        Dim Coeff As Single
        Dim const1 As Single
'          **********************
'          **** Function Main ****
'          **********************
'Function Main()
'+-------------------+
'| Declaration |
'+-------------------+
Dim NumBuffers As Integer
Dim NumOfAVG As Integer
Dim ADREADY As Integer
Dim NumPSDChd As Integer
Dim i, j, k, n,  Length, start As Integer
Dim bw  As Single
Dim Decimation As Integer
Dim DecLength As Integer
Dim DecStart As Integer
Dim PSDChan As Integer
Dim Phs0, Phs1, Phs2,Phs3 As Single
Dim Interval As Single
Dim cmdbuff  As FloatArray
Dim cmdbuffer As FloatArray
Dim DAdata As FloatArray
Dim Channel As FloatArray
Dim S0 As FloatArray
Dim AA As FloatArray
Dim BB As FloatArray
Dim CC As FloatArray
Dim DD As FloatArray
Dim G0 As FloatArray
Dim G1 As FloatArray
Dim G2 As FloatArray
Dim G3 As FloatArray

FFT_SIZE =2048
BUFFSIZE =2048
nmax=FFT_SIZE/2 +1
Fc=GenLine(0,0,4)
LPBW=GenLine(0,0,4)
Order=GenLine(0,0,4)
G0 = GenLine(0,0,3)
G1 = GenLine(0,0,3)
G2 = GenLine(0,0,3)
G3 = GenLine(0,0,3)
cmdbuffer=GenLine(0,0,19)
'+---------------------------------------------+
'| get the initial A/D frequency |
'+--------------------------------------------+
Do While Not Host.GetDataReady(CTRLchan)
Loop
cmdbuff = Host.GetBuffer(CTRLchan)
Frequency = cmdbuff(0)
FFT_SIZE = cmdbuff(1)

If Host.GetDataReady(UserChan) Then
        cmdbuff=Host.GetBuffer(UserChan)
        PSDChan = cmdbuff(0)
End If
'+-------------------------------------+
'| Define A/D parameters |
'+--------------------------------------+
AD.Frequency =Frequency
AD.ClockSource = DTDSP_ICLK
AD.TriggerSource = DTDSP_ITRG
AD.NumChannels = 2   ' 8test
AD.AcqMode = DTDSP_WRAP
AD.InternalChannelBufferSize = BUFFSIZE * 8
'+--------------------------------------+
'| Define D/A parameters |
'+--------------------------------------+
'DA.Frequency = Frequency
'DA.ClockSource = DTDSP_ICLK
'DA.TriggerSource = DTDSP_ITRG
'DA.NumChannels = 1
'DA.AcqMode = DTDSP_WRAP
'DA.InternalChannelBufferSize = BUFFSIZE*3
'+-------------------------------------------------+
'| initialize global array to all zeros |
'+-------------------------------------------------+
ImpulseResponse = GenLine(0,0,BUFFSIZE)
ImpulseResponse(BUFFSIZE/2) = FFT_SIZE
'+---------------------+
'| initialize filter |
'+---------------------+
'Do While Not Host.GetDataReady(CTRLchan)
'Loop
'ProcessControl()
'+--------------------------------+
'| start acquiring data |
'+--------------------------------+
AD.ChannelType=DTDSP_DI
AD.Filter=AD.Frequency / 2
'AD.Start()

'           ****************
'           * Main looping *
'           ****************
Phs0 = 0
Phs1 = 0
Phs2 = 0
Phs3 = 0
```

```
InitFilters(LPBW, Order)
const1 = Frequency / (2*PI*NDEC)
Interval = 2 * PI * FFT_SIZE/Frequency

Do While  1
        If Host.GetDataReady(GHchan) Then
                cmdbuffer=Host.GetBuffer(GHchan)
                For j=0 To 3
                        Fc(j)=cmdbuffer(1+j*4)
                        LPBW(j)=cmdbuffer(2+j*4)
                        Order(j)=cmdbuffer(3+j*4)
                Next j
                NDEC=cmdbuffer(16)
                NumBuffers = cmdbuffer(17)
                RCFactor = cmdbuffer(18)
                MM = FFT_SIZE/NDEC
                Coeff = exp( - PI * RCFactor)
                const1 = Frequency / (2*PI*NDEC)
                InitFilters(LPBW, Order)
                'Host.DebugPrint(" Coeff=" & Str(Coeff))
        End If


        AA = GenLine(0,0,0) 'initialized for concatenation
        BB = GenLine(0,0,0)
        CC = GenLine(0,0,0)
        DD = GenLine(0,0,0)
'          +--------------------------------------+
'          | Extract data from buffer
'          +--------------------------------------+
        S0 = S0 * 0
        For k = 1 To NumBuffers
                'For  j = 1  To NumOfAvg
                        TimeData =Host.GetBuffer(DiskChan)
                        PSD = ApplyFFT(TimeData)
                        S0 = S0 + PSD
                'Next  j
                Host.PutBuffer(PSDChan-1) = PSD  'S0/NumOfAvg


                For i=0 To 3
                        If i=0  Then
                                Phs0 = Phs0 + Interval* Fc(i)
                                U0 = GenComplexCos(phs0,Fc(i),Frequency,FFT_SIZE)
                                G0=GHC( G0, U0, Filter0X, Filter0Y)

                                AA = AA & IFQ
                        ElseIf i=1 Then
                                Phs1 = Phs1 + Interval * Fc(i)
                                U1 = GenComplexCos(phs1,Fc(i),Frequency,FFT_SIZE)
                                G1=GHC(G1, U1, Filter1X, Filter1Y)

                                BB = BB & IFQ
                        ElseIf i=2 Then
```

```
                                Phs2 = Phs2 +Interval * Fc(i)
                                U2 = GenComplexCos(phs2,Fc(i),Frequency,FFT_SIZE)
                                G2=GHC(G2, U2, Filter2X, Filter2Y)

                                CC = CC & IFQ
                        Else
                                Phs3 = Phs3 +Interval * Fc(i)
                                U3 = GenComplexCos(phs3,Fc(i),Frequency,FFT_SIZE)
                                G3 =GHC(G3, U3, Filter3X, Filter3Y)

                                DD = DD & IFQ
                        End If
                Next i
        Next k
        If Host.PutDataReady(10) Then
                Host.PutBuffer(10)=AA
        End If
        If Host.PutDataReady(11) Then
                Host.PutBuffer(11)=BB
        End If
        If Host.PutDataReady(12) Then
                Host.PutBuffer(12)=CC
        End If
        If Host.PutDataReady(13) Then
                Host.PutBuffer(13)=DD

        End If
        If (Host.GetDataReady(CTRLchan)  And (AD.DataReady(0) < FFT_SIZE)) Then
                ProcessControl()
        End If

Loop    'End of Do While loop

End Function
'
'           *********************************
'           **** Function ProcessControl ****
'           *********************************
'

Function ProcessControl()
Dim ctrlbuff As FloatArray
Dim FilterDataTemp As FloatArray
Dim FFTDataTemp As ComplexArray
Dim ready As Integer
Dim lowercut, uppercut As Single
Dim length As Integer
ready = TRUE
Do While  ready
  ctrlbuff = Host.GetBuffer(CTRLchan)
  ready = Host.GetDataReady(CTRLchan)
Loop
'+---------------------------------------------+
'| generate a band pass filter object |
'+-----------------------------------------------------+
lowercut = ctrlbuff(CUTOFFLower)
uppercut =ctrlbuff(CUTOFFUpper)
length   = ctrlbuff(FILTERPOINTS)
BPFilter0 = InitBandPassFilter(lowercut,uppercut,Frequency,length)
BPFilter1 = InitBandPassFilter(lowercut,uppercut,Frequency,length)
FilterDataTemp = Filter(BPFilter0,ImpulseResponse)
FFTDataTemp = fft(FilterDataTemp)
FilterDataTemp = dB(FFTDataTemp)
FilterDataTemp = FilterDataTemp - 20*log10(FFT_SIZE) + 6.0
'+--------------------------------------+
'| Send data to host for filter plotting |
'+--------------------------------------+
Host.PutBuffer(FILTchan) = FilterDataTemp
End Function
'
'                    ********************************
'                    **           Function InitFilters              **
'                    ********************************
Function InitFilters(LP As FloatArray, Od As FloatArray) As Integer
Dim i As Integer
            Filter0x = InitLowPassFilter(LP(0), Frequency, Od(0))
            Filter0y = InitLowPassFilter(LP(0), Frequency, Od(0))
            Filter1x = InitLowPassFilter(LP(1), Frequency, Od(1))
            Filter1y = InitLowPassFilter(LP(1), Frequency, Od(1))
            Filter2x = InitLowPassFilter(LP(2), Frequency, Od(2))
            Filter2y = InitLowPassFilter(LP(2), Frequency, Od(2))
            Filter3x = InitLowPassFilter(LP(3), Frequency, Od(3))
            Filter3y = InitLowPassFilter(LP(3), Frequency, Od(3))
End Function
'
'              ********************************************
'              **        Function ApplyFFT( )                  **
'              **                                                 **
'              **        Calls:  Window, fft, dB               **
'              ********************************************
'
Function ApplyFFT(SomeData As FloatArray) As FloatArray
Dim Temp1 As FloatArray
Dim Temp2 As ComplexArray
Dim BHWnd As FloatArray
Dim dBScale As Single

       dBScale = -20 * log10(10*FFT_SIZE) + 6.0
       BHWnd=GenWindow(BLACKMAN_HARRIS_61,FFT_SIZE)
       Temp1=Window(BHWnd, SomeData)
       Temp2=fft(Temp1)
       ApplyFFT=dB(Temp2)+dBScale
End Function
'
'
'              ********************************
'              **        Function Conjugate        **
'              ********************************
'
```

```
Function Conjugate( A  As Complex) As Complex
Dim r1, r2 As Single
        r1 = Real(A)
        r2 = -Imag(A)
        Conjugate = MakeComplex( r1, r2)
End Function
'
'         **********************************************
'         **          Function   ApplyCPWBD                    **
'         **********************************************
Function ApplyCPWBD(AA  As FloatArray ) As FloatArray
Dim TData As FloatArray        'Function parameter by value only
Dim FData As ComplexArray
Dim PhData As FloatArray
Dim AmpData As FloatArray
Dim IM As FloatArray
Dim R As FloatArray
Dim DC As Single
        TData =AA
        FData=fft(TData)
        PhData = Phase(FData)
        AmpData = Abs(FData)
        AmpData = GenLine(0, 1, 1024)
        IM = AmpData* sin(PhData)
        R= AmpData * cos(PhData)
        TData = irfft(MakeComplex(R,IM))
        TData= Abs(TData)
        DC = Sum(TData) / FFT_SIZE
        TData = TData- DC
        ApplyCPWBD = TData
End Function
'
'         ***********************************
'         **          Function ApplyReduction
'         ** Return T4(0,1,2)=RMS,Skew, kurt
'        ** T4(0) = RMS, T4(1) = Skew, T4(2) = Kurt
'         ***********************************
'
Function ApplyReduction( AA As FloatArray,   n As Integer) As FloatArray
Dim S1, S2, S3, S4, Skew, Kurt  As Integer
Dim T1 As FloatArray
Dim T2 As FloatArray
Dim T3 As FloatArray
Dim T4 As FloatArray
Dim RR As FloatArray
        RR = GenLine( 0, 0 , 6)
        S1 = Sum(AA)/n
        T1 = AA - S1
        T2 = T1 * T1
        T3 = T2 * T1
        T4 = T2 * T2
        S2 = Sum(T2)/n
        S3 = Sum(T3)/n
        S4 = Sum(T4)/n
        RR(0) = S2 ^ .5
        If  S2 >0 Or S2 <= -18  Then
                RR(1) = S3/(S2 ^ 1.5)
                RR(2) = S4/(S2 ^ 2)
        Else
                RR(1) = 0
                RR(2) = 0
        End If
        RR(3) = Max(AA)
        RR(4) = Min(AA)
        RR(5) = S1
        ApplyReduction = RR
End Function
'               *******************
'               ** Function atn2       **
'               *******************
Function atn2(xx As Single, yy As Single) As Single
Dim  phi As Single
        phi = atn(yy/xx)
        If  (phi >0 And xx <0) Or  (phi <0 And xx <0) Then
                atn2 = phi + PI
        Else
                atn2 = phi
        End If
End Function
'                       ******************************************
'                       ***     Function  GHC                   ***
'                       ***     Global: Coeff, const1,MM, IFQ        ***
'                       ******************************************
Function GHC(GL As FloatArray,  UU As ComplexArray, FX As FloatArray, FY As
FloatArray ) As FloatArray
Dim  i, j,k  As Integer
Dim ZZx As FloatArray
Dim ZZy As FloatArray
Dim Tx As FloatArray
Dim Ty As FloatArray
Dim env As FloatArray
Dim ps As FloatArray
Dim TIFQ As FloatArray
Dim TG As FloatArray
        IFQ = GenLine(0,0,MM)
        ZZx = GenLine(0,0,MM)
        ZZy = GenLine(0,0,MM)
        env =GenLine(0,0,MM)
        ps =GenLine(0,0,MM)
        TIFQ = GenLine(0,0,MM)
        TG = GenLine(0,0,3)  'hold  values of changed GL array

        Tx = 2* TimeData * Real(UU)
        Ty = -2 * TimeData * Imag(UU)
        Tx= Filter(FX, Tx)
        Ty = Filter(FY, Ty)
        j = 1
        For i =0 To FFT_SIZE -1
```

```
                k=j*NDEC
                If  (i<k)  Then
                        ZZx(j-1) = ZZx(j-1) +Tx(i)
                        ZZy(j-1) = ZZy(j-1) +Ty(i)
                Else
                        j = j+1
                End If
        Next i
        ZZx = ZZx / NDEC
        ZZy = ZZy / NDEC
        For i =0 To MM-1
                env(i) = (ZZx(i)^2 + ZZy(i)^2)^0.5
                ps(i) = atn2( ZZx(i) , ZZy(i))
                ZZx(i) = cos(ps(i))
                ZZy(i) = sin(ps(i))
        Next i
        TG(0) = ZZx(MM-1)
        TG(1) = ZZy(MM-1)
        TIFQ(0) = const1*(ZZx(0)*(ZZy(0)-GL(1))-(ZZx(0)-GL(0))*ZZy(0))
        For i=1 To MM-1
                TIFQ(i) = const1*(ZZx(i)*(ZZy(i)-ZZy(i-1))-(ZZx(i)-ZZx(i-1))*ZZy(i))
        Next i
        IFQ(0) = (1- Coeff) * TIFQ(0) +Coeff* GL(2)
        For i = 1 To MM-1
                IFQ(i) = (1-Coeff) *TIFQ(i) +Coeff* IFQ(i-1)
        Next i
        TG(2) = IFQ(MM-1)
        GHC = TG
End Function
'                       *********************
'                       **      RCLowPassFilter**
'                       *********************
'Function RCLowPassFilter( A As Single, XX As FloatArray) As FloatArray
'Dim TT As FloatArray
'Static Pa, Pb, Pc, Pd, PP As Single
'Dim i As Integer
'       TT = GenLine(0,0,MM)
'       TT(0) = (1- A) * XX(0) + A * Pa
'       For i = 1 To MM-1
'               TT(i) = (1-A) * XX(i) + A * TT(i-1)
'       Next i
'       Pa = TT(MM-1)
'       RCLowPassFilter = TT
'
'End Function
```

3.0 Conclusion

This Final Report documents and summarizes the results of the software conversion effort. The six advanced nonlinear diagnostic DSP algorithms discussed in Section 2, have been converted to a format consistent with that required for integration into the Vanderbilt Multigraph Architecture (MGA) Model Based Programming environment. Textual/graphical descriptions of analysis algorithm structures along with their signal flow charts have been established. Simulation examples with various pertinent application scenarios were accomplished for demonstration and performance evaluation. This effort allows the real-time execution of these algorithms using the MSFC MPP Prototype System. In this report, ASRI has provided the complete computer source code, including all FORTRAN/C++ Utilities, and all other utilities/supporting software libraries that are required for operation.

Consultation/coordination on the details of these algorithms with both MSFC dynamic analysts and Vanderbilt University's Measurement and Computing Group has also been provided by ASRI. This activity include face-to-face meetings to provide discussions/briefings of the details of these algorithms with both MSFC dynamic analysts and Vanderbilt University's Measurement and Computing Group

In the performance of this contract, ASRI retains “unpublished-all rights reserved under the copyright laws of the United States” regarding the algorithms computer software rights. The government has unlimited use as governed by the Federal Acquisition Regulation.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE November 31, 1996	3. REPORT TYPE AND DATES COVERED Final Report/November 1996

4. TITLE AND SUBTITLE Conversion-Integration of MSFC Nonlinear Signal Diagnostic Analysis Algorithms for realtime Execution on MSFC's MPP Prototype System	5. FUNDING NUMBERS NAS8-40341
6. AUTHOR(S) Jen-Yi, Jong	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AI SIGNAL RESEARCH, INC. 3322 South Memorial Parkway, Suite 67 Huntsville, AL 35801	8. PERFORMING ORGANIZATION REPORT NUMBER TR-4017-96-FR
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Ms. Patrica Fundum, GP26-V George C. Marshall Space Flight Center NASA Marshall Space Flight Center, AL 35812	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report was prepared by AI Signal Research, Inc. (ASRI) for the George C. Marshall Space Flight Center, National Aeronautics and Space Administration. The work was performed under contract NAS8-40341, entitled "Conversion-Integration of MSFC Nonlinear Signal Diagnostic Analysis Algorithms for realtime Execution on MSFC's MPP Prototype System " over the time period from June 7, 1995 through December 2, 1996.

The work was carried out by Dr. Jen-Yi Jong serving as Program Manager. Dr. Jong was responsible for the software development, conversion and evaluation of Nonlinear Signal Diagnostic Analysis Algorithms. Ms. Polly Lu serving as programmer was responsible for implementing the software on C40 DSP using Data Translation's DSP-EZ programming language.

14. SUBJECT TERMS			15. NUMBER OF PAGES 76
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

NSN 7540-01-280-5500

Standard Form 298 (Rev-2-89)
Prescribed by ANSI Std. Z39-18
298-102