Order Number 1345379

Phase estimation as applied to group delay measurements

Phillips, Alan Paul, M.S.

The University of Arizona, 1991

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

PHASE ESTIMATION AS APPLIED TO GROUP DELAY MEASUREMENTS

by

Alan Paul Phillips

A Thesis Submitted to the Faculty of the

Department of Electrical & Computer Engineering

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1991

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: Alan P. Phillips

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Dr. Randall K. Bahr
Assistant Professor of
Electrical and Computer Engineering

7/8/91
Date

ACKNOWLEDGEMENT

The research and writing of this thesis was made possible by the support of the Network Measurements Division of the Hewlett-Packard Company in Santa Rosa, California.

My extended thanks to Dr. Randall K. Bahr who derived the correlation receiver method of phase estimation. Dr. Bahr also made numerous useful suggestions in various areas of the thesis.

TABLE OF CONTENTS

PAGE

I. LIST OF ILLUSTRATIONS 6

II. LIST OF TABLES 9

III. ABSTRACT 10

IV. INTRODUCTION 11

V. THEORY OF SIGNAL DELAY AND GROUP DELAY 13

VI. THE MEASUREMENT SYSTEM 17

VII. GROUP DELAY MEASUREMENT METHODS 18

VIII. PHASE ESTIMATION TECHNIQUES 21

IX. SOURCES OF ERROR WITHIN THE MEASUREMENT SYSTEM 24
IX.A. NOISE 24
IX.B. LINEAR DISTORTION 25
IX.C. NONLINEAR DISTORTION 27

X. DIGITAL SIGNAL PROCESSING OF ANALOG SIGNALS 29

XI. INTRODUCTION TO PART I 32
XI.A. DEVICE UNDER TEST SIMULATION 32
XI.B. MEASUREMENT SYSTEM SIMULATION 34
XI.C. GROUP DELAY MEASUREMENT SIMULATION IMPLEMENTATION 37

XII. THE HILBERT TRANSFORM METHOD OF PHASE ESTIMATION 39
XII.A. ANALYTIC SIGNALS 39
XII.B. DEMODULATION AND PHASE ESTIMATION USING THE ANALYTIC SIGNAL 40

XIII. THE CORRELATION RECEIVER METHOD OF PHASE ESTIMATION 44
XIII.A. THE MAXIMUM LIKELIHOOD ESTIMATOR 44
XIII.B. AM CORRELATION RECEIVER 45
XIII.C. PM CORRELATION RECEIVER 49

XIV. THE SYSTEM IDENTIFICATION METHOD OF PHASE ESTIMATION 53
XIV.A. ADAPTIVE FILTER REQUIREMENTS 55
XIV.B. THE WIENER FILTER SOLUTION 57
XIV.C. THE CORRELATION CANCELER APPROACH 59
XIV.D. THE STOCHASTIC GRADIENT ALGORITHM 60

XIV.E. THE RECURSIVE LEAST SQUARES ALGORITHM 62
XIV.F. PERFORMANCE COMPARISON OF THE RLS AND LMS ALGORITHMS 64

XV. INTRODUCTION TO PART II 68
XV.A. NONLINEAR SYSTEM SIMULATION 68

XVI. NONLINEAR SYSTEM IDENTIFICATION 71
XVI.A. THE VOLTERRA SERIES REPRESENTATION OF NONLINEAR SYSTEMS 71
XVI.B. THE SECOND ORDER DISCRETE VOLTERRA SYSTEM 73
XVI.C. SECOND ORDER SYSTEM IDENTIFICATION 74
XVI.D. SECOND ORDER SYSTEM IDENTIFICATION RESULTS 77

XVII. NONLINEAR EQUALIZATION 79
XVII.A. THE P'TH ORDER INVERSE OF A VOLTERRA SYSTEM 79
XVII.B. THE THIRD ORDER INVERSE OF THE SECOND ORDER VOLTERRA SYSTEM 81
XVII.C. SECOND ORDER SYSTEM LINEARIZATION RESULTS 83

PART III

XVIII. MONTE CARLO ANALYSIS OF THE PHASE ESTIMATION METHODS 89

XIX. CONCLUSIONS 103

APPENDIX I: PROGRAMS FOR PART I 105
APPENDIX I.A. AM AND PM MODULATION SIGNAL GENERATION 105
APPENDIX I.B. AM HILBERT TRANSFORM METHOD 109
APPENDIX I.C. PM HILBERT TRANSFORM METHOD 111
APPENDIX I.D. AM CORRELATION RECEIVER METHOD 113
APPENDIX I.E. PM CORRELATION RECEIVER METHOD 116
APPENDIX I.F. LMS ALGORITHM 120
APPENDIX I.G. RLS ALGORITHM 123
APPENDIX I.H. LINEAR SYSTEM IDENTIFICATION METHOD 126

APPENDIX II: PROGRAMS FOR PART II 132
APPENDIX II.A. PROGRAM FOR SECOND ORDER NONLINEAR SYSTEM IDENTIFICATION 132
APPENDIX II.B. SUBROUTINE FOR SECOND ORDER NONLINEARITY EQUALIZER 138

APPENDIX III: PROGRAMS FOR PART III 139
APPENDIX III.A. SAMPLE PROGRAM FOR THE MONTE CARLO ANALYSIS 139

REFERENCES 144

I. LIST OF ILLUSTRATIONS

Page

Figure 1; Measurement System Block Diagram 18

Figure 2; Magnitude Response of Filter Under Test 33

Figure 3; Phase Response of Filter Under Test 33

Figure 4; Group Delay of Filter Under Test 34

Figure 5; Measurement System Simulation Block Diagram 36

Figure 6; Quadrature Filter Response 40

Figure 7; Analytic Signal Generation 40

Figure 8; Block Diagram of the IF Phase Estimation 48

Figure 9; Block Diagram of the AM Modulation Envelope Phase Estimation 48

Figure 10; Bessel Curves as a Function of their Order, n, and PM Modulation Index ß 51

Figure 11; System Identification Method Block Diagram 54

Figure 12; RLS Convergence Error 65

Figure 13; LMS Convergence Error 65

Figure 14; RLS Convergence Error at End of Data Buffer 66

Figure 15; LMS Convergence Error at End of Data Buffer 66

Figure 16; RLS Adaptive Filter Phase and DUT Filter Phase 67

Figure 17; LMS Adaptive Filter Phase and DUT Filter Phase 67

Figure 18; Magnitude Response of the Linear Filter of Equation (54) 69

Figure 19; Phase Response of the Linear Filter of Equation (54) 69

Figure 20; Graph of the Second Order Nonlinearity from Equation (55) 70

Figure 21; Second Order System Composed of Equation (54) and Equation (55) 71

Figure 22; Second Order System Identification Block Diagram 75

Figure 23; Entries of the Second Order Adaptive Coefficient Vector **H** After Convergence 78

Figure 24; Second Order System Cascaded with its P'th Order Inverse 80

Figure 25; Third Order Inverse System, K(3) 83

Figure 26; Power Spectrum of the IF Signal 86

Figure 27; Power Spectrum of the IF Signal after the Second Order Nonlinearity 86

Figure 28; Power Spectrum of the IF Signal after the Nonlinear Part of the Equalization 87

Figure 29; Power Spectrum of the IF Signal after Full Equalization 87

Figure 30; Difference Between the IF Signals, Before and After the Nonlinearity 88

Figure 31; Difference Between the IF Signals, Before the Nonlinearity and After Equalization 88

Figure 32; Relative Group Delay Error, Noiseless Environment, 10Khz AM Modulation 93

Figure 33; Relative Group Delay Error, Noiseless Environment, 10Khz PM Modulation 93

Figure 34; Relative Group Delay Error, Additive Noise, 10Khz AM Modulation 94

Figure 35; Relative Group Delay Error, Additive Noise, 10Khz PM Modulation 94

Figure 36; Coefficient of Variation, Additive Noise, 10Khz AM Modulation 95

Figure 37; Coefficient of Variation, Additive Noise, 10Khz PM Modulation 95

Figure 38; Relative Group Delay Error, Noiseless Environment, 1Khz AM Modulation 96

Figure 39; Relative Group Delay Error, Noiseless Environment, 1Khz PM Modulation 96

Figure 40; Relative Group Delay Error, Additive Noise, 1Khz AM Modulation 97

Figure 41; Relative Group Delay Error, Additive Noise, 1Khz PM Modulation 97

Figure 42; Coefficient of Variation, Additive Noise, 1Khz AM Modulation 98

Figure 43; Coefficient of Variation, Additive Noise, 1Khz PM Modulation 98

Figure 44; Relative Group Delay Error, 25Khz Sinusoidal Interference, 10Khz AM Modulation 99

Figure 45; Relative Group Delay Error, 25Khz Sinusoidal Interference, 10Khz PM Modulation 99

Figure 46; Coefficient of Variation, 25Khz Sinusoidal Interference, 10Khz AM Modulation 100

Figure 47; Coefficient of Variation, 25Khz Sinusoidal Interference, 10Khz PM Modulation 100

Figure 48; Relative Group Delay Error, Second Order Distortion, 10Khz AM Modulation 101

Figure 49; Relative Group Delay Error, Second Order Distortion, 10Khz PM Modulation 101

Figure 50; Relative Group Delay Error, Distortion with Equalizer, 10Khz AM Modulation 102

Figure 51; Relative Group Delay Error, Distortion with Equalizer, 10Khz PM Modulation 102

II. LIST OF TABLES

Page

Table 1; Hilbert Transform Demodulation and Phase Estimation Flowchart, AM Case 42

Table 2; Hilbert Transform Demodulation and Phase Estimation Flowchart, PM Case 43

III. ABSTRACT

This thesis addresses the problem of measuring the group delay of a network with different frequencies at its input and output ports. Three distinct methods of measuring group delay are developed and their performance compared in a Monte Carlo analysis. Also considered is the problem of second order nonlinear distortion and its effect on measuring group delay. A method is developed to identify a specific second order nonlinear structure causing distortion. A nonlinear equalizer is then developed to cancel, or equalize, the undesired second order distortion.

IV. INTRODUCTION

Group delay specifications are an integral part of the standards that govern the quality of a data transmission channel. Group delay is a critical specification because it measures a channel's phase linearity as a function of frequency. Wide bandwidth signals, whether they be digital or analog, depend on a linear phase characteristic for distortionless transmission.

This thesis concerns itself with two problems associated with the measurement of group delay. The first problem, addressed in part I of the thesis, is that of measuring group delay through frequency translation devices (FTD's) such as mixers. Three distinct methods are presented in part I that meet this requirement. To measure FTD's all methods require a carrier either amplitude or phase modulated to measure group delay at the carrier frequency of interest. The methods presented in part I can be used when the frequencies are the same, or are different as is the case with FTD's, at the input and output ports of a device or network under test.

All methods presented in this thesis are implemented as digital programs which post-process the down converted and sampled data as collected from the simulated measurement system. The measurement system simulates a network analyzer with two channels, each available for measuring both amplitude and phase on the devices input and output ports.

The second problem, addressed in part II of the thesis, is that of correcting group delay measurement errors caused by nonlinear distortion occurring within the measurement system itself. Without correction, significant errors are shown to result. A method, called nonlinear system identification, is developed that measures the characteristics of a second order nonlinearity causing the distortion. The measured characteristics are then used to create an inverse system, called a nonlinear equalizer, that equalizes or cancels the undesired nonlinearity.

Part III of the thesis includes a Monte Carlo analysis of the phase estimation methods developed in part I used in conjunction with the nonlinear equalizer developed in part II. Also included in part III is a summary of the relative performance of the methods developed.

The performance of the phase estimation methods is measured under ideal as well as nonideal conditions by the introduction of broadband noise, sinusoidal noise, linear distortion and nonlinear distortion. The effectiveness of the nonlinear equalizer is also verified in part III.

V. THEORY OF SIGNAL DELAY AND GROUP DELAY

A signal, or wave front, propagating through a medium will experience a propagation delay T_{prop}. Given the propagation velocity, V_{prop}, and the physical length of the medium, L_{med}, then:

$$T_{prop} = \frac{L_{med}}{V_{prop}} \tag{1}$$

For an electromagnetic wavefront in a perfect dielectric then:

$$V_{prop} = \frac{1}{\sqrt{\mu * \varepsilon}} \tag{2}$$

Where μ is the permeability constant and ϵ is the permittivity constant of the dielectric medium. For free space $\mu = 1.26*10^{-6}$ H/m and $\epsilon = 8.85*10^{-12}$ F/m giving $V_{prop} = 3*10^{8}$ m/s = c, the speed of light in a vacuum.

If the propagation velocity changes as a function of the signal's frequency then the propagation medium is said to be dispersive and is an imperfect dielectric. A signal propagating through such a medium will have its individual frequency components arrive at a given reference point at different times. Such a medium is dispersive in time. The dispersion of a signal with more than one frequency component leads to a *distorted* version of the signal at a given point in the medium of propagation.

A quantity called the **phase delay** can be defined as the steady state delay of a frequency

component through a network, or:

$$T_{phase} = -\frac{\theta(\omega)}{\omega} \tag{3}$$

This quantity gives a time delay in terms of the phase shift, θ, of a frequency component, ω. This quantity is relative to the input and output reference points of a propagation medium , i.e. an electrical network. Since this is a steady state definition it does not give the actual propagation delay of a given frequency component.

The **group delay** of a network represents the propagation delay through the network of a narrow frequency group $\Delta\omega$, centered at the frequency ω. The defining equation is:

$$T_{group} = -\frac{d[\theta(\omega)]}{d[\omega]} \tag{4}$$

Where $\theta(\omega)$ is in radians and ω is in radians per second. An alternate definition is:

$$T_{group} = -\frac{d[\phi(f)]}{360 \cdot d[f]} \tag{5}$$

Where $\phi(f)$ is in degrees and f is in Hertz.

Since the group delay is the negative derivative of the networks phase with respect to frequency a constant group delay implies a linear phase characteristic with frequency. If a network's phase characteristic is linear **and** has a zero phase intercept at zero frequency then the group delay and phase delay have the same value [1]. Note that group delay is a positive quantity since it is the measure of a signal's propagation time through a medium.

Envelope delay is the steady state delay of the envelope of a modulated signal waveform through a network. In the limit, as the modulation frequency approaches zero, the phase response of the network over the upper to lower sideband interval is approximately linear. Then in the limit, as $\Delta\omega \rightarrow 0$, the envelope delay is equivalent to group delay. This equivalency is derived below in equations (6) through (10). In this example suppressed carrier modulation is used for simplification but the results of the derivation apply to both AM and narrowband PM modulation. The equivalence between AM and narrowband PM modulation is shown at the end of section IX.b.

A network has an amplitude modulated carrier with a sideband phase shift of $\theta[\omega_c+\omega_m]$ and $\theta[\omega_c-\omega_m]$, where $\theta[\omega]$ is the network's phase shift as a function of ω. The modulation sidebands are then described as:

$$A_l \cdot \cos((\omega_c-\omega_m)\cdot t+\theta[\omega_c-\omega_m])+A_u \cdot \cos((\omega_c+\omega_m)\cdot t+\theta[\omega_c+\omega_m]) \qquad (6)$$

For brevity assume that $A_l = A_u = 1/2$, then applying the trigonometric sum rule to equation (6) gives:

$$\frac{1}{2}\cdot 2\cdot \cos\left(\frac{1}{2}\left(\left(\omega_c+\omega_m\right)\cdot t+\theta[\omega_c+\omega_m]+(\omega_c-\omega_m)\cdot t+\theta[\omega_c-\omega_m]\right)\right) \cdot \cos\left(\frac{1}{2}\cdot\left((\omega_c+\omega_m)\cdot t+\theta(\omega_c+\omega_m]-(\omega_c-\omega_m)\cdot t-\theta(\omega_c-\omega_m]\right)\right) \qquad (7)$$

Combine terms in equation (7) and simplify to give:

$$\cdot \cos\left[\omega_m t + \frac{\theta[\omega_c+\omega_m]-\theta[\omega_c-\omega_m]}{2}\right]\cdot\cos\left[\omega_c t + \frac{\theta[\omega_c+\omega_m]+\theta[\omega_c-\omega_m]}{2}\right] \qquad (8)$$

The second argument in the first cosine term in equation (8) gives the phase shift of the modulation envelope through the network. The second argument in the second cosine term gives the phase shift of the carrier through the network. If the network phase shift is linear then:

$$\theta[\omega_c+\omega_m] = -k\cdot(\omega_c+\omega_m) + c \quad \textit{and} \quad \theta[\omega_c-\omega_m] = -k\cdot(\omega_c-\omega_m) + c \qquad (9)$$

Where k and c are constants. Evaluating the above expression in equation (8) gives an envelope phase shift of $-k\omega_m$ and, by applying equation (3), a group delay of $-(-k\omega_m)/\omega_m = k$ sec.

Using equation (9) with the definition of group delay, given in equation (4), results in:

$$T_{group} = -\frac{d[\theta[\omega]]}{d[\omega]} = -\frac{d[-k\cdot\omega+c]}{d[\omega]} = k \text{ sec.} \qquad (10)$$

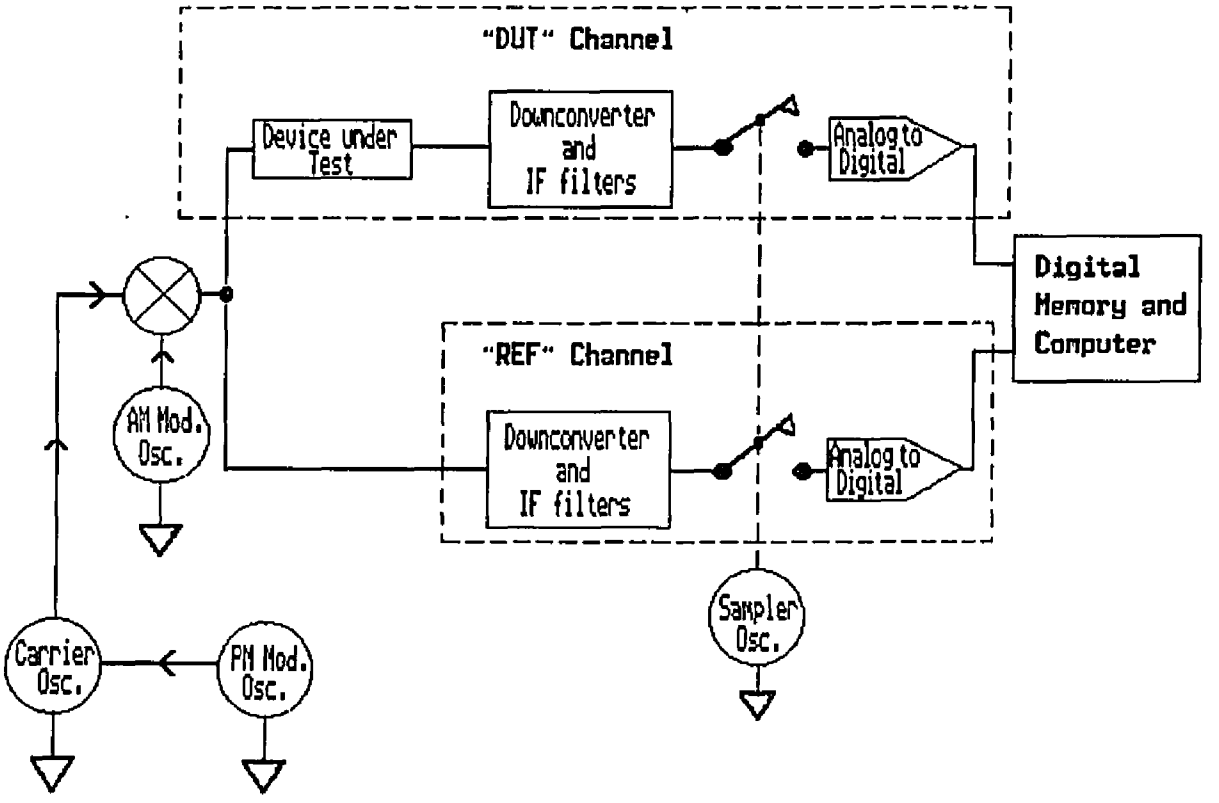
As shown, the envelope delay method gives identical results as the defining equation for group delay when the network's phase shift is linear and the network has a constant amplitude over the frequency range of interest.

VI. THE MEASUREMENT SYSTEM

Since group delay is defined in terms of a network's phase response as a function of frequency then it is obvious that a group delay measurement system must have the capability to measure the network's phase over the frequencies of interest at the network's input and output ports. An instrument with such a capability is called a vector network analyzer.

As shown in figure 1 the network analyzer has a carrier source or oscillator, a modulator source and modulators for AM and PM, all of which make up the source section of the analyzer. Also shown in the figure are two identical receiver channels, one channel is connected to the input port of the device under test as a reference and the other channel is connected to the output port. Following the connection to the device under test the channels are down converted to an intermediate frequency or IF. The channel's IF is sampled, effectively instantaneously, by its respective sample and hold circuit. The sample and hold's output maintains the value of the IF signal at the sampling instant. The analog to digital converters then convert, with finite resolution, the analog signal into a digital number representation. These numbers, or data, are stored in a digital memory accessible by a digital computer. The computer will subsequently digitally demodulate the data and apply the phase estimation algorithms as developed in part I of this thesis.

Figure 1; Measurement System Block Diagram



VII. GROUP DELAY MEASUREMENT METHODS

The defining equation for group delay, as shown in equation (4), can be approximated by its **discrete derivative** using the carrier frequencies ω_1 and ω_2 to give:

$$T_{group} = -\frac{d[\theta[\omega]]}{d[\omega]} \approx \lim_{(\omega_1-\omega_2)\to 0} -\frac{\theta_{\omega_1} - \theta_{\omega_2}}{\omega_1 - \omega_2} \qquad (11)$$

The quantity $\omega_1 - \omega_2$ is called the **frequency aperture**. Too large an aperture gives insufficient frequency resolution for a network with a rapidly changing phase characteristic and results in **aperture error**. This discrete derivative method is used when the frequencies at the input and output ports of a network are the same.

If the network's input and output frequencies are different then the lack of a phase reference between the input and output ports renders the discrete derivative method unusable. An alternative method, called the **envelope delay** method is used for such Frequency Translation Devices (FTD's). This method uses an AM or PM modulated carrier which is passed through the network or Device Under Test (DUT). The modulated carrier is demodulated at both the DUT's input and output ports to give the modulation envelopes. The delay of the output modulation envelope relative to the input modulation envelope is the envelope delay of the network. As shown in section V, by using an arbitrarily small modulation frequency the envelope delay is equivalent to the network's group delay at the carrier frequency. For DUT's with a nonlinear phase response, excessively large modulation frequencies result in aperture error and the envelope delay then no longer approximates the true group delay.

The modulated envelope can be demodulated by digital signal processing techniques, as explored in this thesis, or by an analog demodulator. For broadband signals that are not downconverted by the measurement system to a low frequency IF amenable to digital sampling, an analog AM envelope detector is an alternative. Using digital sampling and processing, however, allows both AM and PM modulation types to be used. A situation where PM modulation has a distinct advantage is a network that has an AM compression characteristic which may reduce the AM modulation index to an unusable level.

An *ad hoc* method, known as the **reference mixer** method, can be used with FTD's and a discrete derivative calculation. This method uses a reference mixer with its RF and LO ports driven with the same RF and LO sources as the mixer under test. Each of the mixer's IF ports drives a separate network analyzer input channel to give a relative delay measurement. The reference mixer is chosen to add a negligible delay to this new composite network. The new network thus provides the same frequency at both its input port and output port, allowing the use of the discrete derivative method without the modulation and demodulation required of the envelope delay method. The disadvantage of this technique is the addition of the reference mixer which may contribute significantly to the group delay measurement uncertainty of the mixer under test.

VIII. PHASE ESTIMATION TECHNIQUES

The envelope delay method, applied in part I of this thesis, requires the measurement of the demodulated envelopes relative delay between a network's input and output ports. This delay is related to the envelopes phase by the phase delay relation shown in equation (3). When the frequency ω is replaced by the modulation frequency ω_{mod} the relative envelope delay is given by:

$$T_{env.\ delay} = \frac{\theta env_{output} - \theta env_{input}}{\omega_{mod}} \qquad (12)$$

Since the modulation frequency is assumed to be known a priori it remains to calculate the envelope's phase at the network's input and output ports. The equation for envelope delay points out the important relationship between the modulation frequency and the phase shift for a fixed envelope delay value. For a small modulation frequency the phase detector needs a high sensitivity which is a tradeoff against the aperture error that results when large modulation frequencies are used.

There are numerous methods by which to calculate or **estimate** the phase of the demodulated envelope. The distinction between the various methods is their relative performance in the presence of additive Gaussian noise. Noise with a Gaussian distribution is always present in a electronic network since its source is thermally generated component noise. The optimum estimation methods set a performance criterion or cost function on what to maximize. A common cost function, for example, is minimum squared error between the actual and estimated values. The qualities that describe an estimator are its bias and variance. The bias, shown in equation

(13) below, is the difference between the actual and estimated mean values. The variance of an estimate, shown in equation (14), is a measure of the sample by sample deviation from the mean, or a measure of the estimate's randomness. The goal of an optimum estimator is to produce an estimate with an unbiased mean and a minimum variance.

$$\textit{Mean of } x = \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}, \quad \textit{Bias} = \bar{x} - \hat{x}, \quad \hat{x} = \textit{estimate of } x \tag{13}$$

$$\textit{Variance of } \hat{x} = \sigma_{\hat{x}}^2 = \frac{\sum_{i=1}^{n} (\hat{x}_i - \bar{x})^2}{n} \tag{14}$$

One estimation method called a **maximum likelihood** estimation is optimum in the presence of additive Gaussian noise and will produce an estimate of the chosen parameter that is the *most likely* or most probable to occur. In this case the parameter is the modulation envelope's phase. A correlation receiver or matched filter receiver are maximum likelihood detectors.

Other optimum estimation methods include **mean square** estimation and **linear mean square** estimation. These methods minimize the mean square difference , or error, between the actual phase value and its estimate. An example of linear mean square estimation is the choosing of a filter such that the filter's output is the best possible estimate, in the least mean square error sense, of a reference signal. Such a filter implementation is called a **Wiener filter.**

An example of a non-optimum method of phase estimation is a **phase lock loop.** Although it is possible for a phase lock loop to give optimum results in a noisy environment the

loop's performance is dependent on its transfer function which may have a multitude of other constraints [2]. The **Fourier transform** is a non-optimum method which gives the phase and magnitude of the signal's frequency domain, or spectral components. Spectral estimation using the Fourier transform, i.e. a periodogram, gives an estimate which is a trade-off between increasing the bias for decreasing the variance through periodogram averaging [13,p58]. Another non-optimum method, the **Hilbert transform**, gives a phasor or complex analytic time domain representation of the signal.

When implementing phase detection using a non-optimum method the signal with noise may require either *ensemble* averaging and/or *time* averaging to reduce the noise variance, or power, to an acceptable level. In the case of additive noise the signal plus noise is the new random variable; $y = s + n_i$, where s is the (deterministic) signal and n_i is the uncorrelated noise from the i'th ensemble average. Given that the variance of n_i is σ^2_n then the sum of N ensemble averages would give a new variance for y of $\sigma^2_y = \sigma^2_n/N$ [3]. Then if ten ensemble averages were taken, the noise power would be reduced by ten decibels. Time averaging is accomplished by filtering the signal to limit the bandwidth, hence the noise power, of the signal plus noise. Time averaging is also referred to as *video* averaging in instrumentation literature.

IX. SOURCES OF ERROR WITHIN THE MEASUREMENT SYSTEM

The phase estimate's accuracy is diminished in the presence of noise, linear distortion, nonlinear distortion and aperture error. All of these nonidealities exist, to some degree, within both the measurement system and the network under test.

IX.A. Noise

Noise is generated from both the measurement system and from the device under test. Probable noise types include thermal noise, phase noise, quantization noise, shot noise and flicker noise. Although intermodulation noise is classified as a noise it is not a random quantity and will be addressed as a nonlinear distortion product.

Shot noise, like thermal noise, has a white spectral density and can be treated as thermal noise insofar as this thesis is concerned. Flicker noise is insignificant at low frequencies and will not be discussed further. Quantization noise is caused by the one half Least Significant Bit (LSB) uncertainty of the analog to digital converter found in the measurement system presented in this thesis. Quantization noise is an additive noise type with a uniform distribution spanning +/- one half of the LSB. Phase noise is a concern since phase is the measured parameter, however it is not an additive noise type.

The phase estimation simulation implemented in part I of the thesis assumes additive white Gaussian noise, i.e. thermal noise, generated by the device under test as the only noise source.

IX.B. Linear Distortion

Linear distortion occurs when a signal with more than one frequency component passes through a network with nonlinear phase and nonconstant amplitude over the range of frequencies contained in the signal. A network's requirements for distortionless transmission of a signal are 1) constant group delay and 2) constant amplitude. Both 1) and 2) need only apply over the range of frequencies present in the signal. The constant group delay requirement enables the relative phasing of the signal's individual frequency components to remain unchanged through the network. The constant amplitude requirement enables the relative amplitudes of the signal's individual frequency components to remain unchanged through the network.

Linear distortion can produce extraneous modulation conversion from AM to PM and from PM to AM. An FM discriminator, for example, makes use of the amplitude change of the discriminator as a function of frequency to convert FM into AM. AM to PM conversion exemplifies the deleterious effect of modulation conversion. The PM modulation results in an extraneous phase shift inseparable from the desired phase shift of the modulation envelope. This conversion takes place when a networks amplitude slope changes the relative symmetry of the modulation sidebands making one sideband higher than the other. This dissymmetry results from two signals of identical frequencies but with dissimilar phase being summed together, where the phasing is different for the upper and lower sidebands. Reducing the frequency aperture will cause a reduction of this effect. A network's nonlinear phase can change the carrier to sideband phase relationship which also can produce modulation conversion. The following derivation shows that nonsymmetrical sideband attenuation can result in simultaneous AM and PM modulation. Given a general AM and PM modulated signal in its phasor description:

$$s(t) = \Re[A\cdot(1+m_{am}\cdot\cos(\omega_m t))\cdot e^{j\cdot(\omega_c t+\beta_{pm}\cdot\sin(\omega_m t))}] \quad (15)$$

Applying the identity $Re\{z_1\cdot z_2\} = Re\{z_1\}\cdot Re\{z_2\} - Im\{z_1\}\cdot Im\{z_2\}$ gives:

$$\begin{aligned} s(t) = A\cdot(1+m_{am}\cdot\cos(\omega_m t))\cdot[&\cos(\omega_c t)\cdot\cos(\beta_{pm}\cdot\sin(\omega_m t)) \\ &- \sin(\omega_c t)\cdot\sin(\beta_{pm}\cdot\sin(\omega_m t))] \end{aligned} \quad (16)$$

Using the approximations for narrowband PM modulation with $\beta_{pm} << 1$ gives :

$$\cos(\beta_{pm}\cdot\sin(\omega_m\cdot t)) \approx 1 \quad \text{and} \quad \sin(\beta_{pm}\sin(\omega_m t)) \approx \beta_{pm}\sin(\omega_m t) \quad (17)$$

Using these approximations reduces equation (16) to the following:

$$s(t) = A(1+m_{am}\cos(\omega_m t))\cdot[\cos(\omega_c t)-\beta_{pm}\sin(\omega_m t)\cdot\sin(\omega_c t)] \quad (18)$$

Expanding equation (18) above gives:

$$\begin{aligned} s(t) \approx\ & A\cdot\cos(\omega_c t)+\frac{1}{2}A(m_{am}+\beta_{pm})\cdot\cos((\omega_c+\omega_m)\cdot t) \\ &+ \frac{1}{2}A(m_{am}-\beta_{pm})\cdot\cos((\omega_c-\omega_m)\cdot t) \\ &- m_{am}\beta_{pm}A\cdot\sin(\omega_m t)\cdot\cos(\omega_m t)\cdot\sin(\omega_c t) \end{aligned} \quad (19)$$

The last term in equation (19) can be ignored since $m_{am}\cdot\beta_{pm}$ is assumed to be $<< 1$. To demonstrate the effects of a network with linear distortion, take a signal for this example with a carrier amplitude of 1.0, an upper sideband amplitude of 0.95 and a lower sideband amplitude of 0.65. This signal, by applying equation (19), will have an AM modulation index of 0.8 and

a PM modulation index of 0.15.

Equation (19) can also be used to show the similarities between AM and narrowband PM modulation. If m_{am} is set to zero in equation (19) the result is the same form as AM modulation, with $ß_{pm}$ replacing m_{am}, except that the lower sideband has a negative amplitude coefficient. The sidebands add in quadrature to the carrier in PM modulation producing the variations in phase.

IX.C. Nonlinear Distortion

Nonlinearities within the measurement system can produce **harmonic distortion, intermodulation distortion** and **cross modulation distortion.** These nonlinearities manifest themselves as new, extraneous frequency terms that can interfere with the measurement process.

The effects of a nonlinearity on the demodulation and phase estimation process depend on where in the measurement system the nonlinearity is located. The measurement system can easily eliminate the DC terms present by AC coupling before sampling or by subtracting the mean of the signal after sampling has occurred. The other frequency terms may or may not be filtered out by the measurement system's IF bandpass filters.

Harmonic distortion results from a nth degree nonlinearity producing new frequency terms that are up to n times higher in frequency than the input frequencies. If n is odd then the odd harmonics, from one up to and including the nth harmonic, will be produced. If n is even then the even harmonics, from zero up to and including the nth harmonic, will result.

Intermodulation distortion results from a nonlinearity operating on two or more input frequencies that are simultaneously present, for example a carrier modulated with AM or low

index PM will have three frequencies, the carrier and two sidebands. A second order nonlinearity will produce new frequencies that are at the sum and difference of the original frequencies. These frequency terms are typically far away in frequency from the original signals and are therefore filtered out by the IF filters in the measurement system if the nonlinearity is located before the down conversion circuitry.

A third order nonlinearity will produce new frequencies, that include, a frequency that is the sum of twice an input signal's frequency with the other input signal's frequency. Also produced is the difference frequency, of twice an input signal frequency with the other input signal frequency. It is this difference frequency that can be close to an input frequency, thus enabling it to pass through the measurement system's IF filters and cause interference with the phase detection process.

Cross modulation can occur with a third order nonlinearity operating on a desired modulated input signal that is present with another, for example extraneous, signal. The extraneous signal will be cross modulated with the modulation of the desired signal. Depending on the extraneous signal's frequency and phase, the incidental demodulation of this extraneous modulation could distort the desired modulation envelope and cause measurement errors.

The cross modulation can also produce modulation conversion from, for example, AM to PM modulation. A third order system with memory, describable using a Volterra series, can demonstrate this effect [4].

X. DIGITAL SIGNAL PROCESSING OF ANALOG SIGNALS

As previously mentioned in section VI. the modulated input and output waveforms of the device under test are downconverted to a low frequency IF. At this point the waveform is sampled and digitized using a sampler followed by an analog to digital converter. The digitized IF samples are then stored in a digital memory where they are computer accessible for further processing. The digital processing of the continuous signal's sampled, discrete time representation, necessitates that digital signal processing theory be applied.

The continuous time signal to be sampled must be bandlimited to one half of the sampling rate to meet the requirements of the **Nyquist sampling theorem**. This theorem states that a continuous time signal may be fully reconstructed from its discrete time samples if the above requirement is met. If the continuous time signal is not bandlimited a condition called **aliasing** is encountered. The spectrum of a sampled signal repeats itself periodically at intervals equal to the sampling rate. Aliasing causes the lower spectral frequencies of the next spectral repetition interval to "fold" back into the previous spectral interval's upper frequencies thus corrupting the signal's spectral content.

Another condition requiring consideration of the Nyquist theorem takes place entirely in the digital domain. If discretely sampled, i.e. digital signals, are multiplied together or are operated on in a nonlinear way, i.e. squared etc. then new frequencies will result as described in section IX under nonlinear distortion. The highest frequency generated "digitally" must again be less than half the sampling rate or aliasing will occur. Since we are operating entirely in the digital domain a digital filter cannot be used to "filter out" the excessively high frequencies. This is due to the fact that the aliasing has already taken place and the digital filter's own transfer

function spectrum repeats itself periodically with the sampling frequency.

The solution to the above problem is to increase the sampling rate, in the digital domain only, of the digital signals. The sampling rate of the continuous time signal remains the same, the continuous time signal's bandwidth meeting the Nyquist sampling theorem requirement for it only. The process of increasing the digital sampling rate is known as **interpolation** and increases the effective sampling rate by interpolating between samples, thus adding in effect another sample point or points, between the original sample points. Obviously the storage requirements increase depending on the degree of interpolation. The interpolation process is done previous to the operations that generate the new frequencies. As much interpolation is done as is needed in anticipation for the highest frequencies to be generated. The interpolation process is described mathematically in the following signal reconstruction equation [5].

$$\hat{x}(t) = \sum_{n=-\infty}^{\infty} x(n) \cdot sinc\left(\frac{n \cdot T - t}{T}\right) \tag{20}$$

The above equation is equivalent to filtering *x(n)* with an ideal low pass filter with a cutoff frequency of one half the sampling rate.

The inverse operation of interpolation is called **decimation** and is used when a lower sampling rate can be used based on the highest frequency contained in the discrete time signal. A lowpass filtering operation may be done before decimating to ensure the Nyquist sampling theorem requirement is met. Decimation simply reduces the sample rate by K by choosing one sample to save out of every K original samples. Decimation is used to reduce the storage and processing requirements of a signal.

The digital filtering operation or discrete convolution sum operation has the following

general form shown in equation (21) for the one dimensional case. This equation applies to the Infinite Impulse Response (IIR) filter as well as the Finite Impulse Response (FIR) filter.

$$y(n) = -\sum_{k=1}^{N} \frac{a_k}{a_0} y(n-k) + \sum_{r=0}^{M} \frac{b_r}{a_0} x(n-r) \qquad (21)$$

x_n is the input sequence and y_n is the output sequence at the discrete time index n. For a finite impulse response, or FIR filter, the a_k coefficients for k greater than zero, are zero. The impulse, or equivalently, the unit sample response for the FIR filter is given by b_n/a_0.

Although the time response of a digital filter is discrete in time, the filter's frequency response is **continuous** in frequency. This means that a continuum of phase shifts and amplitude changes can result from passing a discrete signal sequence through a digital filter.

XI. INTRODUCTION TO PART I

Part I of this thesis addresses the application of the envelope delay method to group delay measurements. The implementation of this method, as mentioned in section VII, requires a modulated carrier. Part I develops three distinct phase estimation methods to measure the envelope phase shift or equivalently, the envelope delay. The first is the Hilbert transform method, the second is the correlation receiver method and the third is the system identification method. All of these methods are tested with, and compared against, the same device for which the group delay is to be measured.

XI.A. Device Under Test Simulation

The device under test (DUT) is a fifth order digital lowpass butterworth filter and was picked for ease of simulation. Although the filter is "digital", whereas an actual device under test would be analog, the digital filter has an analog representation. This distinction is not important for the purposes of the simulation, however, since the digital filter will give a continuous response in the frequency domain. The analog representation can be approximated through the use of a **z plane to s plane transformation** or mapping. Several transformation techniques are available with the bilinear transformation method being invertible, i.e. a two way mapping, between the s and z planes. The bilinear transformation gives an analog filter with a magnitude representation that has a warped frequency axis characteristic. The phase representation suffers even more with a linear phase response representation in the analog, or s, domain resulting from a nonlinear (tangent) phase response in the digital or z domain [6].

The magnitude, phase and actual group delay characteristics of the digital filter tested are shown in figures 2, 3, and 4 respectively. These plots were generated using MATLAB's magnitude, phase and group delay commands operating on the filter's frequency response.

Figure 2: Magnitude Response of Filter Under Test

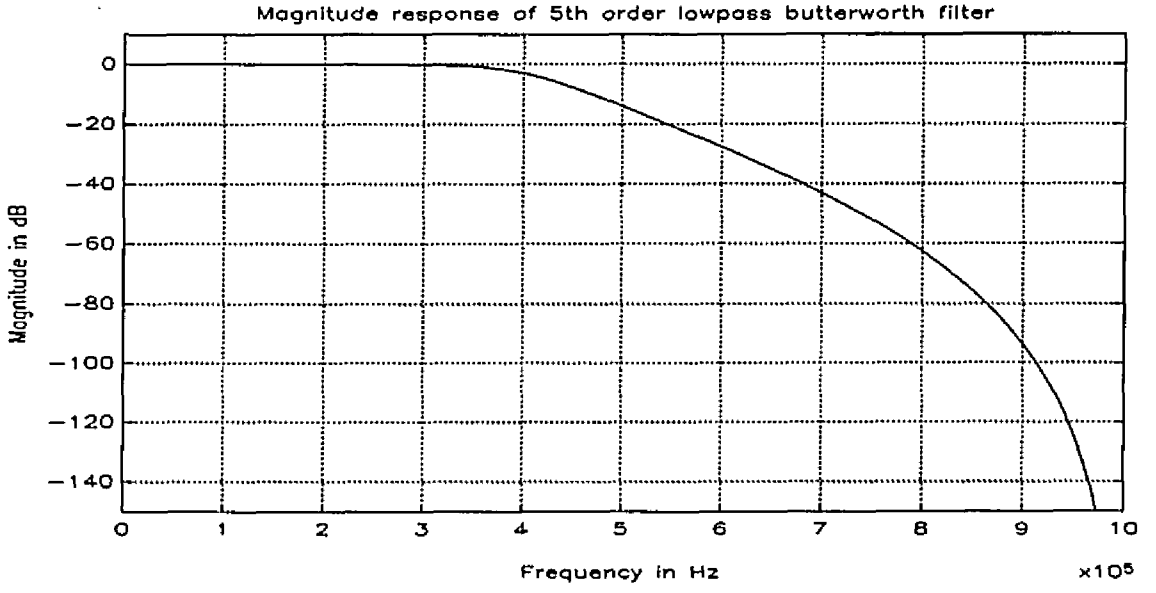


Figure 3: Phase Response of Filter Under Test

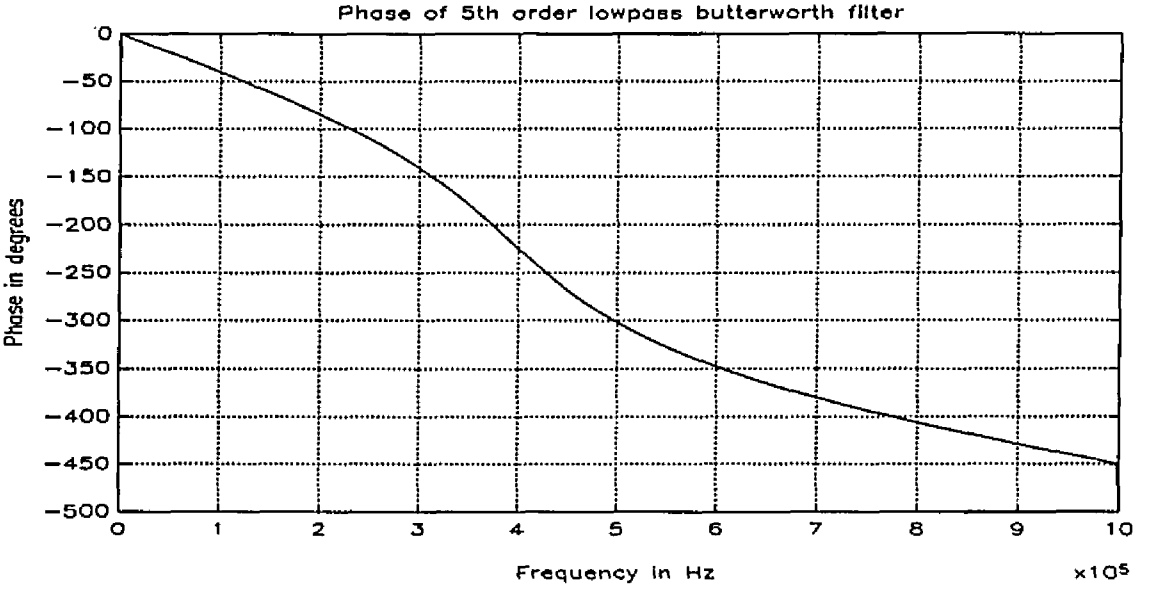
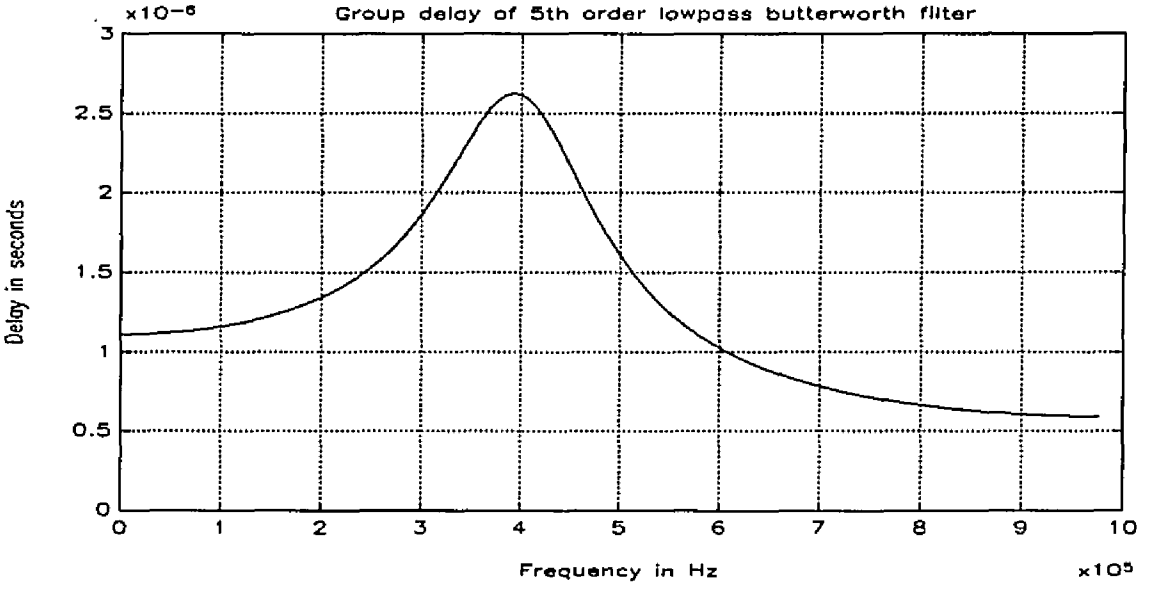


FIGURE 4; Group Delay of Filter Under Test



XI.B. Measurement System Simulation

The measurement system under simulation represents a vector network analyzer with two channels. One channel is referred to as the reference channel and the other channel is referred to as the DUT channel. The connection of these channels to the DUT's input and output ports is shown in figure 5. The use of two channels allows phase shifts relative to the DUT's input and output ports to be measured.

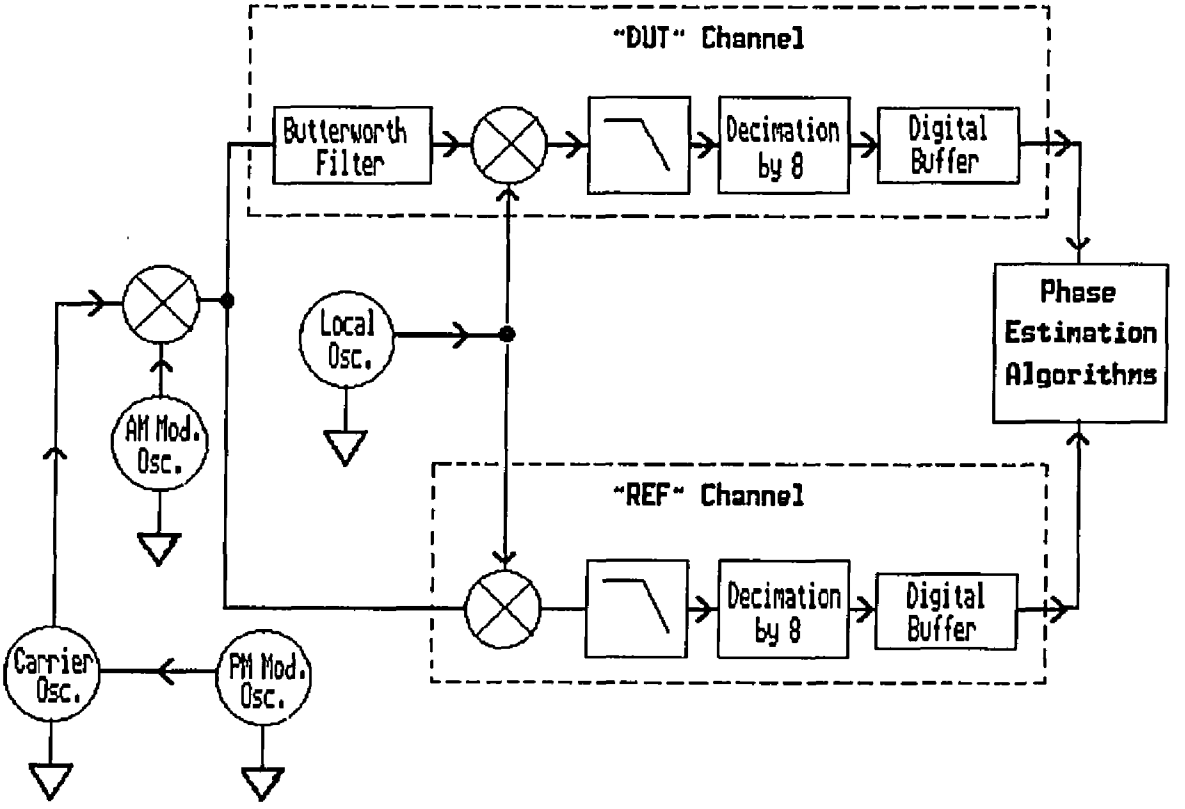
It is assumed, to simplify the simulation, that all extraneous phase shifts have been calibrated out of the measurement system. This would be accomplished in actual practice by initially inserting a DUT with a phase response that is flat with frequency known as a "through line" in place of the actual DUT. The phase shifts are then recorded for each frequency of interest and used to offset the phase measurements taken later with the actual DUT.

In implementing the simulation of the measurement system using digital programs it was necessary to use sampled or discrete versions of the analog signals to be replicated. This necessitates the use of digital signal processing techniques as discussed in section X. The discrepancies between the simulated and the actual measurement system will be indicated as required in the description.

The DUT has its group delay characteristics measured over a carrier frequency range of 100Khz to over 800Khz. A block diagram of the simulation is shown in figure 5. The actual measurement system was shown in figure 1. The sampling frequency is 2Mhz to avoid aliasing up to a carrier frequency of 1Mhz. This was a limitation in the simulation only, since the actual measurement system's block diagram, as shown in figure 1, is still "analog" at this stage and therefore is not sampled. The response of the filter to the modulated carrier is then downconverted, by multiplication or "mixing", to a 50Khz IF frequency. Although the extraneous upper mixing products can exceed the Nyquist rate of 1Mhz the aliasing that results does not "fold" back into the desired low frequency mixing products until a carrier frequency of over 900Khz is reached. At this point the sampling rate of 2Mhz is excessive considering the highest frequency content of the signal is that of the upper modulation sideband only, which is at 50Khz plus the modulation rate. The effective sampling rate is lowered by decimation to 250Khz to reduce the simulation's computational burden. The choice of an IF frequency of 50kHz and the subsequent sampling rate of 250Khz limits the modulation rate to less than 20Khz to meet the antialiasing requirements with a finite order antialiasing filter. This modulation rate limitation is "real" in the sense that it exists in the actual measurement system and not just the simulation. The actual implementation of the simulation does not contain a separate IF filter since the decimation routine in MATLAB includes a prefilter which serves this purpose. The actual

analog measurement system would, of course, require both the IF and antialiasing filters prior to sampling. Obviously no samplers or analog to digital conversion stages are needed in the simulation since the signal is in digital form already. The exact equivalency between the simulation and the actual measurement system occurs at the digital memory or buffer stage. The phase estimation algorithms for both the simulation and the actual measurement system are the same since both operate on digital samples.

Figure 5; Measurement System Simulation Block Diagram



XI.C. Group Delay Measurement Simulation Implementation

The simulation's phase estimation algorithms that measure the modulation envelope's phase operate on the sampled IF signal, from both the reference channel and the DUT channel. The program that generates these signals is called AMGEN or PMGEN depending on whether AM or narrowband PM modulation is desired. AMGEN and PMGEN implement the simulation block diagram up to, and including, the digital memory. They, as for all the programs for part I, are written in the MATLAB simulation language and are located in appendix I.a.

AMGEN and PMGEN generate 14 frequency points chosen so that the signal coming from the DUT; the digital butterworth filter, has its magnitude decreased by 6 dB for each frequency point beyond 300Khz. The phase estimation programs add Gaussian noise at constant power level to the DUT channel data so as to initially have a post-detection Signal to Noise Ratio, or SNR, of 34 dB. The fact that the noise is added after filtering at a constant power level results in the SNR decreasing by 6 dB for each frequency point beyond 300Khz

The post-detection SNR measures the power in the modulation sidebands excluding the carrier power. The choice of a post-detection SNR as opposed to a pre-detection SNR was made since the carrier is not used at all in the system identification method.

The modulation signal power is affected by the following: carrier source amplitude, modulation index, downconversion process, and the DUT filter attenuation. In the simulation the carrier source amplitude was chosen to be unity and the modulation index was chosen to be equal to one half. The downconversion process, with a LO amplitude of unity, results in the desired lowpass signal's amplitude being scaled down by one half. The DUT filter attenuation factor is $A(f)$ where $A(f)$ has a maximum value of unity as seen from figure 2.

The Gaussian noise is digitally generated using a pseudorandom noise generator, as available in MATLAB, and then bandlimited to approximately 80Khz to simulate the lowpass effect of an analog antialiasing filter. The original noise spectrum, as measured from a sampling rate of 250Khz, has a uniform spectrum out to 125Khz. After bandlimiting, the noise power was measured by calculating its variance. Then a suitable noise source multiplying coefficient was selected to give the initial post-detection SNR of 34 dB. The SNR relationship can be described by equation (22) below.

$$\mathrm{SNR}_{\text{Post Det.}} = \frac{\textit{mod. signal power}}{\textit{noise}_{BL}\ \textit{power}} = \frac{\textit{upper and lower sideband power}}{\textit{noise}_{BL}\ \textit{power}}$$

$$= \frac{\left[\frac{1}{2}A(f)\cdot\frac{\textit{mod index}}{2}\right]^2}{\sigma_{BL}^{\,2}} = \frac{\left[\frac{1}{2}\frac{1}{2}\frac{1}{2}\cdot A(f)\right]^2}{6.25\cdot 10^{-6}} = 2.5\cdot 10^{3}\cdot (A(f))^2 \qquad (22)$$

XII. THE HILBERT TRANSFORM METHOD OF PHASE ESTIMATION

In this section a demodulation and phase detection method based on the Hilbert transform is developed and implemented using the MATLAB simulation language. This method is nonoptimum in the sense that in the presence of additive Gaussian noise the resulting phase estimation result will have a larger variance than that obtainable with an optimum method. The performance results of applying this method to group delay measurements can be found in section XIX.

XII.A. Analytic Signals

The discrete complex **analytic signal** $v(n)$ is defined as $v(n) = x(n) + j\,\hat{x}(n)$. The imaginary component $\hat{x}(n)$ and real component $x(n)$ are related by the discrete **Hilbert transform.** A discrete analytic signal has negative frequency components which are zero between $-1/2 < f < 0$ where the normalized frequency f is equal to one at the sampling frequency f_S. This is possible only if the signal is complex since a discrete real signal $x(n)$ has a Fourier transform with Hermitian symmetry, i.e. $X(-f) = X^*(f)$. The signal $\hat{x}(n)$ can be formed by applying the discrete Hilbert transform to x(n), or equivalently, by passing the real signal $x(n)$ through a filter, called a **Hilbert transformer** or **quadrature filter**. The impulse response of this filter has the following definition [7]:

$$h(n) = \int_{-1/2}^{0} j \cdot e^{j2\pi nf}\, df + \int_{0}^{1/2} -j \cdot e^{j2\pi nf}\, df \qquad (23)$$

This integral evaluates to the following:

$$h(n) = \frac{2}{\pi n}\sin^2(\frac{n\pi}{2}) \quad \text{for } n \neq 0\ , \quad \text{at } n = 0;\ \ h(0) = 0 \tag{24}$$

The frequency response of the quadrature filter is shown in figure 6. To form the discrete analytic signal *v(n)* the filter's output is summed with a delayed version of *x(n)*. The delay for the real part of *v(n)* is needed to match the delay that the quadrature filter gives the imaginary part of *v(n)*. This operation is shown in figure 7. The resulting complex analytic signal has zero spectrum between $-1/2 < f < 0$.

Figure 6; Quadrature Filter Response

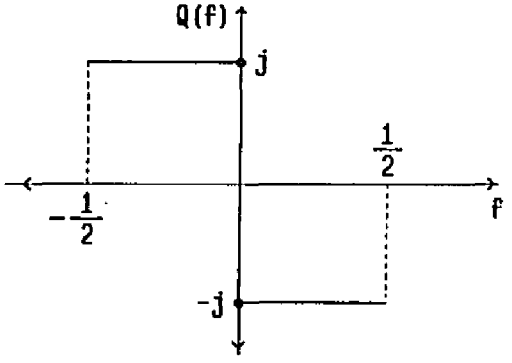
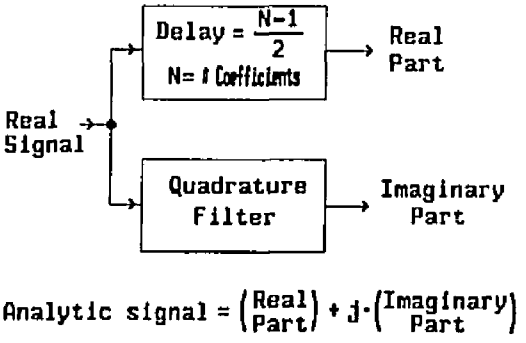


Figure 7; Analytic Signal Generation



XII.B. Demodulation and Phase Estimation using the Analytic Signal

The importance of the phasor representation of the complex analytic signal is shown by representing a general AM and PM modulated carrier as:

$$s(t) = \mathrm{Re}\{\, r(t)\cdot e^{[j\cdot(\omega_c\cdot t + \phi_c + \Phi(t))]}\} \tag{25}$$

This representation allows the modulation to be given by the **complex envelope** defined as:

$$e(t) = r(t) \cdot e^{[j \cdot \Phi(t)]} \tag{26}$$

Where r(t) is the AM modulation and $\Phi(t)$ is the PM modulation [8].

The demodulation of the modulated carrier then amounts to isolating the complex envelope, using the quadrature filter approach. In the case of AM modulation the magnitude of the complex envelope is taken to yield the AM modulation envelope. In the case of PM modulation the argument or phase of the complex envelope is taken to yield the PM modulation signal. Once the demodulated signal is available, in this case a simple sinusoid, then the phase shift of the modulation signal can be found. The phase of the sinusoidal modulation may be found by several methods including finding the phasor representation of the sinusoid through another application of the quadrature filter. The argument of this phasor gives the phase relative to the time the sampling process started on the channel. Since both the reference channel and the DUT channel are sampled synchronously relative to each other the relative phase shift of the modulation envelope through the DUT can be found by subtracting the DUT channel phase result from the reference channel phase result.

The following flowcharts, shown in tables 1 and 2, give the demodulation and phase detection process for the AM case, followed by the PM case respectively.

In the AM and PM case the phase terms $\omega_{mod} \cdot t$ and $\omega_c \cdot t$ appear. These terms are linearly increasing with time t giving a multitude of $2 \cdot \pi$ radian phase cycles that must be "unwrapped" to obtain the true accumulated phase beyond $2 \cdot \pi$ radians.

In the PM case the carrier phase term ϕ_c appears. Since this term is a constant it is subtracted out from the zero mean phase modulation signal $\Phi(t)$ as shown in the PM flowchart

step #5.

The Hilbert transformer is implemented using MATLAB's "hilbert" function. The phase unwrapping is done using MATLAB's "unwrap" function. The MATLAB program that implements the AM flowchart is located in appendix I.b. and the program for the PM flowchart is located in appendix I.c. .

Table 1; Hilbert Transform Demodulation and Phase Estimation Flowchart, AM Case:

Step#	Description	Resulting Equation
	AM modulated carrier:	$A\cdot[1 + m_{am}\cdot\cos(\omega_{mod}\cdot t)]\cdot\cos(\omega_c\cdot t)$
	DUT's magnitude and phase:	$\text{Mag[DUT]} = \lvert H(\omega_c)\rvert$, $\text{Arg[DUT]} = \angle H(\omega_c) = \phi_c$
#1	Take the Hilbert transform	$\rightarrow r(t)\cdot e^{[j\cdot(\omega_c\cdot t + \phi_c)]}$
#2	Take the magnitude of #1	$\rightarrow r(t)$
	Modulation magnitude, phase:	$\text{Mag}[r(t)] = \lvert r(t)\rvert$, $\text{Arg}[r(t)] = \theta_{mod,dut}$
#3	Hilbert transform #2	$\rightarrow \lvert r(t)\rvert\cdot e^{[j\cdot(\omega_{mod}\cdot t + \theta_{mod,dut})]}$
#4	Take the angle of #3	$\rightarrow \omega_{mod}\cdot t + \theta_{mod,dut}$
#5	Unwrap phase of #4	$\rightarrow \omega_{mod}\cdot t + \theta_{mod,dut}$
#6	Subtract $\omega_{mod}\cdot t$	$\rightarrow \theta_{mod,dut}$
#7	Repeat steps 1-6 for ref chan.	$\rightarrow \theta_{mod,ref}$
#8	Find the group delay	$\rightarrow T_{group} = (\theta_{mod,dut} - \theta_{mod,ref})/\omega_{mod}$

Table 2; Hilbert Transform Demodulation and Phase Estimation Flowchart, PM Case

Step#	Description	Resulting Equation
	PM modulated carrier:	$A \cdot \cos(\omega_c \cdot t + \beta_{pm} \cdot \cos(\omega_{mod} \cdot t))$
	DUT's magnitude and phase:	$\text{Mag[DUT]} = \lvert H(\omega_c) \rvert$, $\text{Arg[DUT]} = \angle H(\omega_c) = \phi_c$
#1	Take the Hilbert transform	$\rightarrow r(t) \cdot e^{[j \cdot (\omega_c \cdot t + \phi_c + \Phi(t))]}$
#2	Take the angle of #1	$\rightarrow \omega_c \cdot t + \phi_c + \Phi(t)$
#3	Unwrap phase of #2	$\rightarrow \omega_c \cdot t + \phi_c + \Phi(t)$
#4	Subtract $\omega_c \cdot t$	$\rightarrow \Phi(t) + \phi_c$
#5	Subtract mean of #4 from #4	$\rightarrow \Phi(t)$
	Modulation magnitude, phase:	$\text{Mag}[\Phi(t)] = \lvert \Phi(t) \rvert$, $\text{Arg}[\Phi(t)] = \omega_{mod} \cdot t + \theta_{mod,dut}$
#6	Hilbert transform #5	$\rightarrow \lvert \Phi(t) \rvert \cdot e^{[j \cdot (\omega_{mod} \cdot t + \theta_{mod,dut})]}$
#7	Take the angle of #6	$\rightarrow \omega_{mod} \cdot t + \theta_{mod,dut}$
#8	Unwrap phase of #7	$\rightarrow \omega_{mod} \cdot t + \theta_{mod,dut}$
#9	Subtract $\omega_{mod} \cdot t$	$\rightarrow \theta_{mod,dut}$
#10	Repeat steps 1-9 for ref chan.	$\rightarrow \theta_{mod,ref}$
#11	Find the group delay	$\rightarrow T_{group} = (\theta_{mod,dut} - \theta_{mod,ref})/\omega_{mod}$

XIII. THE CORRELATION RECEIVER METHOD OF PHASE ESTIMATION

In this section the AM and PM correlation receivers are developed and applied to phase estimation. The performance results of applying these methods to group delay measurements can be found in section XIX.

XIII.A. The Maximum Likelihood Estimator

The AM and PM correlation receiver methods are derived from **maximum likelihood** principles. The maximum likelihood estimator, or MLE, will give an estimate of the chosen parameter that is the *most likely* or most probable to occur.

The observed, or measured, signal is of the form $y(t) = s(t) + n(t)$. The signal $y(t)$ represents the desired signal $s(t)$ with additive Gaussian noise $n(t)$. The measured signal, $y(t)$, has a random quality due to the additive noise $n(t)$. The signal $y(t)$ then has a probability density associated with its observation that is conditioned on the occurrence of the signal $s(t)$. This probability density as given in [9] is:

$$p(y|s) = \frac{1}{\sqrt{2\pi}\sigma_n}\exp\left[-\frac{(y-s)^2}{2\sigma_n^2}\right] \qquad (27)$$

The above probability density has its mean shifted by the deterministic part of the signal, $s(t)$, and has a variance given by the random part of the signal, $n(t)$.

The MLE of $s(t)$ is found by maximizing the conditional density as given in equation (27). This will give an estimate of $s(t)$, called $\hat{s}(t)$ that can be interpreted as the value of $s(t)$

which most probably gave rise to the observed quantity *y(t)*. An important property of the MLE is its commutation over nonlinear operations. A MLE of the phase parameter θ will result from a MLE of $s(t,\theta)$ where $s(t,\theta)$ may be a nonlinear function of θ. The MLE solution can be shown in references [9] and [10], to satisfy:

$$\int_{t_0}^{t_f}[y(t,\theta) - s(t,\theta)]\cdot\frac{\partial\, s(t,\theta)}{\partial\,\theta}dt\,|_{\theta=\hat{\theta}} = 0\ ,\ \text{where}\ \hat{\theta} = \text{MLE of}\ \theta \qquad (28)$$

XIII.B. AM Correlation Receiver

For the amplitude modulated case the measured signal $y(t,\theta)$, previously downconverted to an IF frequency by the measurement system, has the following form:

$$y(t,\theta,\phi_{IF}) = A[1+m_{am}\cos(\omega_{mod}t+\theta)]\cos(\omega_{IF}t+\phi_{IF}) + n(t) \qquad (29)$$

An estimate of the carrier phase, $\hat{\phi}_c$ is first recovered by measuring *y(t)* with the modulation turned off. The carrier phase shift is preserved through the downconversion stage, showing up as a phase shift in the IF signal giving $\phi_c = \phi_{IF}$. The carrier phase estimation process was itself implemented as a MLE with the phase estimation block diagram shown in figure 8 below.

After the carrier phase has been estimated at each of the desired carrier frequencies the modulation is then turned on. The modulated signal is then coherently demodulated by downconverting from the IF to baseband by multiplying it with a sinusoid of the IF frequency,

offset with the estimated carrier phase. After lowpass filtering to remove the unwanted upper product this results in the baseband signal:

$$y_{BB}(t,\theta) = \frac{A}{2}[1+m_{am}\cos(\omega_{\mathrm{mod}}t+\theta)] + n(t) \tag{30}$$

Applying the MLE criterion from equation (28) gives, from [10], the following:

$$\int_{t_0}^{t_f}[y_{BB}(t,\theta)-[1+m_{am}\cos(w_{\mathrm{mod}}t+\theta)]]\cdot\frac{\partial\,[1+m_{am}\cos(w_{\mathrm{mod}}t+\theta)]}{\partial\,\theta}dt\,|_{\theta=\hat{\theta}} = 0 \tag{31}$$

For an integration period chosen to be an integral multiple of the modulation period the second term in the integral will be zero giving:

$$\int_{0}^{T_{\mathrm{mod}}}[y_{BB}(t,\theta)\cdot\sin(w_{\mathrm{mod}}t + \theta)]dt\,|_{\theta=\hat{\theta}} = 0 \tag{32}$$

Then applying the sine addition identity $\sin(A+B) = \sin A\cdot\cos B + \cos A\cdot\sin B$ in equation (32) results in:

$$\int_{0}^{T_{\mathrm{mod}}} y_{BB}(t,\theta)\cdot\sin(\omega_{\mathrm{mod}}t)\cdot\cos(\hat{\theta})\,dt + \int_{0}^{T_{\mathrm{mod}}} y_{BB}(t,\theta)\cdot\cos(\omega_{\mathrm{mod}}t)\cdot\sin(\hat{\theta})\,dt = 0 \tag{33}$$

Solving equation (33) for the chosen parameter θ gives the estimate of θ to be:

$$\hat{\theta} = -\,Tan^{-1}\left[\frac{\int_0^{T_{mod}} y_{BB}(t,\theta)\cdot\sin(\omega_{mod}t)\;dt}{\int_0^{T_{mod}} y_{BB}(t,\theta)\cdot\cos(\omega_{mod}t)\;dt}\right] \qquad (34)$$

The implementation of equation (34) results in using the correlation receivers to give the MLE estimate of the demodulated signal *s(t)*. The resulting modulation envelope has its phase determined by taking the arctangent of the signal's DC, or integrated, in-phase and quadrature components. The block diagram of the MLE process in finding the modulation envelopes phase shift is shown in figure 9.

The digital implementation of the algorithm does not provide for continuous integration. Instead integration's discrete counterpart; summation, is used with the trapezoidal rule [11]. The MATLAB implementation of the AM correlation algorithm is found in appendix I.d.

Figure 8; Block Diagram of the IF Phase Estimation

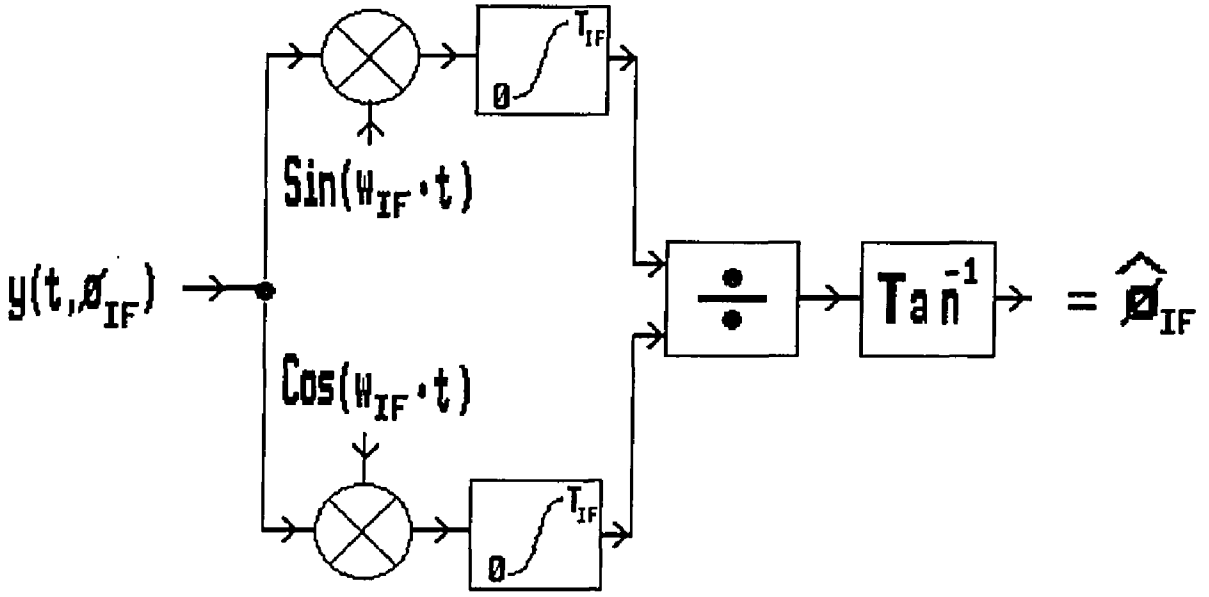
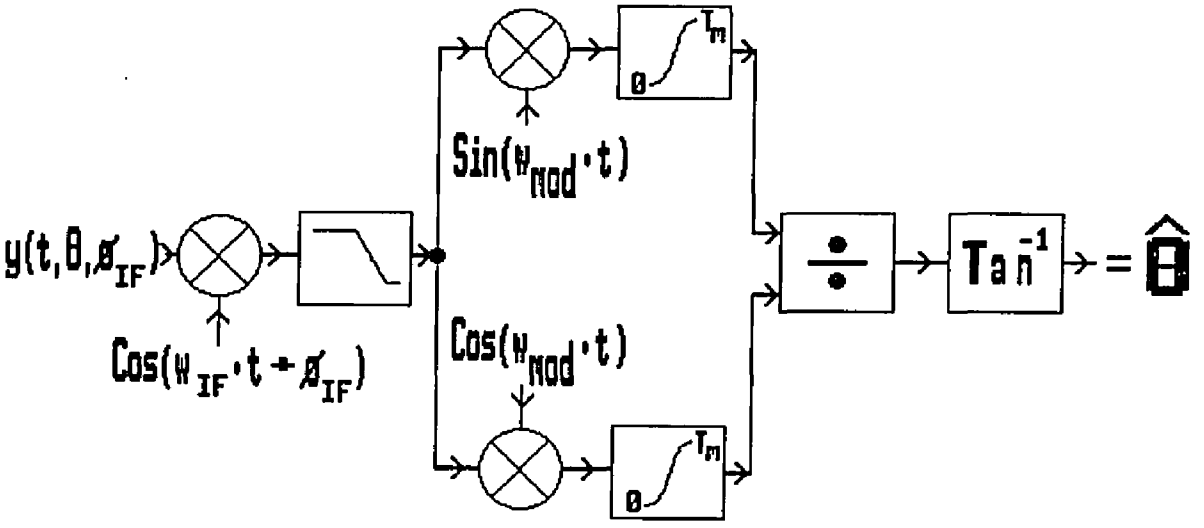


Figure 9; Block Diagram of the AM Modulation Envelope Phase Estimation



XIII.C. PM Correlation Receiver

For the phase modulated case the measured signal $y(t,\theta)$, previously downconverted to an IF frequency by the measurement system, has the following form:

$$y(t,\theta,\phi_{IF}) = A\cdot\cos(\omega_{IF}t+\phi_{IF}+\beta_{PM}\cdot\sin(\omega_m t+\theta)) + n_{noise}(t) \quad (35)$$

Equation (35) may be rewritten in terms of Bessel functions, [10], as:

$$y(t,\theta,\phi_{IF}) = \frac{A}{2}\cdot\sum_{n=-\infty}^{n=\infty} J_n(\beta_{pm})\cdot\cos((\omega_{IF}+n\cdot\omega_{mod})\cdot t + \phi_{IF} + n\cdot\theta) + n_{noise}(t) \quad (36)$$

As in the AM case an estimate of the carrier phase, $\hat{\phi}_c$, is first recovered by measuring $y(t)$ with the modulation turned off. The carrier phase shift is preserved through the downconversion stage, showing up as a phase shift in the IF signal giving $\phi_c = \phi_{IF}$. The IF phase estimation process was itself implemented as a MLE with the IF phase estimation block diagram shown in figure 8 above.

After the IF phase has been estimated at each of the desired carrier frequencies the modulation is then turned on. The MLE solution in its general form was shown in equation (28). For an integration period chosen to be an integral multiple of the modulation period the second term in the integral will be zero giving:

$$\frac{\partial}{\partial\theta}\left[\int_0^{T_{mod}}[y(t,\phi_{IF},\theta)\cdot\frac{A}{2}\cdot\sum_{n=-\infty}^{n=\infty} J_n(\beta_{pm})\cdot\cos((\omega_{IF}+n\cdot\omega_{mod})\cdot t+\hat{\phi}_{IF}+\theta)]dt\right]\Big|_{\theta=\hat{\theta}} = 0 \quad (37)$$

Evaluating the partial derivative of equation (37) gives the result:

$$\sum_{n=-\infty}^{n=\infty} n \cdot J_n(\beta_{pm}) \int_0^{T_{mod}} y(t,\phi_{IF},\theta) \cdot \sin((\omega_{IF}+n\cdot\omega_{mod})\cdot t + \hat{\phi}_{IF} + n\cdot\theta)\ dt\ |_{\theta=\hat{\theta}} = 0 \quad (38)$$

Then applying the sine addition identity $\sin(A+B) = \sin A\cdot\cos B + \cos A\cdot\sin B$ to equation (38) results in:

$$\begin{aligned} &\sum_{n=-\infty}^{n=\infty} n \cdot J_n(\beta_{pm}) \int_0^{T_{mod}} y(t,\phi_{IF},\theta) \cdot \sin(\omega_{IF}+\hat{\phi}_{IF}) \cdot \cos(n\cdot(\omega_{mod}t+\theta))\ dt + \\ &\sum_{n=-\infty}^{n=\infty} n \cdot J_n(\beta_{pm}) \int_0^{T_{mod}} y(t,\phi_{IF},\theta) \cdot \cos(\omega_{IF}+\hat{\phi}_{IF}) \cdot \sin(n\cdot(\omega_{mod}t+\theta))\ dt\ |_{\theta=\hat{\theta}} = 0 \end{aligned} \quad (39)$$

Define the following:

$$\begin{aligned} C_s(n) &\triangleq \int_0^{T_{mod}} y(t,\phi_{IF},\theta) \cdot \sin(\omega_{IF}t+\hat{\phi}_{IF}) \cdot \cos(n\cdot\omega_{mod}t)\ dt \\ C_c(n) &\triangleq \int_0^{T_{mod}} y(t,\phi_{IF},\theta) \cdot \cos(\omega_{IF}t+\hat{\phi}_{IF}) \cdot \cos(n\cdot\omega_{mod}t)\ dt \\ S_s(n) &\triangleq \int_0^{T_{mod}} y(t,\phi_{IF},\theta) \cdot \sin(\omega_{IF}t+\hat{\phi}_{IF}) \cdot \sin(n\cdot\omega_{mod}t)\ dt \\ S_c(n) &\triangleq \int_0^{T_{mod}} y(t,\phi_{IF},\theta) \cdot \cos(\omega_{IF}t+\hat{\phi}_{IF}) \cdot \sin(n\cdot\omega_{mod}t)\ dt \end{aligned} \quad (40)$$

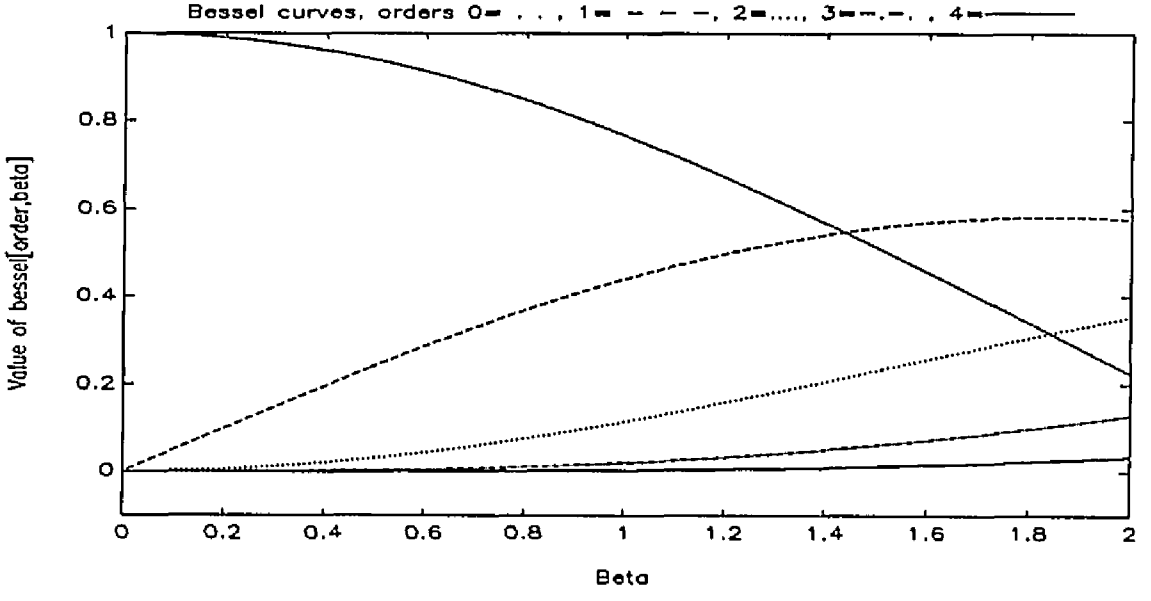
Using the above definitions and the fact that $J_n(ß) = J_{-n}(ß)$ for n even, $J_n(ß) = -J_{-n}(ß)$ for n odd gives the final result:

$$\sum_{n=1\ (n\ odd)}^{\infty} n \cdot J_n(\beta) \cdot [C_s(n) \cdot \cos(n \cdot \hat{\theta}) - S_s(n) \cdot \sin(n \cdot \hat{\theta})] + \sum_{n=2\ (n\ even)}^{\infty} n \cdot J_n(\beta) \cdot [C_c(n) \cdot \sin(n \cdot \hat{\theta}) + S_c(n) \cdot \cos(n \cdot \hat{\theta})] = 0 \quad (41)$$

Equation (41) is solved numerically for the PM modulation envelope's phase shift estimate.

The extent to which the Bessel series is expanded in the above equation depends on the PM modulation index ß. A primary requirement for the group delay measurement is to have a small frequency aperture error. This is obtained by keeping the modulation sidebands in as a narrow region about the carrier as possible. The amplitude of the nth sideband depends on the Bessel function $J_n(ß)$. A graph showing the Bessel curves as a function of n and ß is shown in figure 10 below.

Figure 10; Bessel Curves as a Function of their Order, n, and PM Modulation Index ß



A PM modulation index, ß, of 0.5 will give the following relative spectral amplitudes; Carrier = 0.9385, first sidebands = 0.2423, second sidebands = 0.03060, third sidebands = 0.002564. For a ß = 0.5 this result indicates that using $n_{max} = 2$ will give a very good approximation. In fact, using a larger value of n_{max} would be detrimental since noise would likely dominate the signal of the outermost sidebands giving erroneous results.

In preparation for generating high frequencies that may cause aliasing problems in the PM correlation receiver algorithm the sampled IF data stored in the digital memory is interpolated by a factor of two. The interpolation, as described in section X, is done for both the reference channel and the DUT channel using MATLAB's "interp" function. As in the AM correlation receiver implementation the integration is numerical using the trapezoidal rule. The zero finder, needed to solve equation (41), uses the built in function "fzero" in MATLAB. The MATLAB program implementing the PM correlation receiver method is found in appendix I.e.

XIV. THE SYSTEM IDENTIFICATION METHOD OF PHASE ESTIMATION

The System Identification Method, or SIM, is the last of the three methods presented in part I that determine the phase shift of a modulation envelope through a network. The SIM uses an adaptive filter implementing the Recursive Least Squares, or RLS, algorithm to minimize the error, in the least squares sense, of the signal's estimate $\hat{s}(t)$. The performance results of applying this method to group delay measurements can be found in section XIX.

As shown in section V the envelope delay method works by measuring the phase shift, or equivalently the delay, of the modulation envelope through a DUT. The phase shift of the modulation envelope was given for the AM case in equation (8) repeated below.

$$\cos\left[\omega_m t+\frac{\theta[\omega_c+\omega_m]-\theta[\omega_c-\omega_m]}{2}\right]\cdot\cos\left[\omega_c t+\frac{\theta[\omega_c+\omega_m]+\theta[\omega_c-\omega_m]}{2}\right] \quad (8)$$

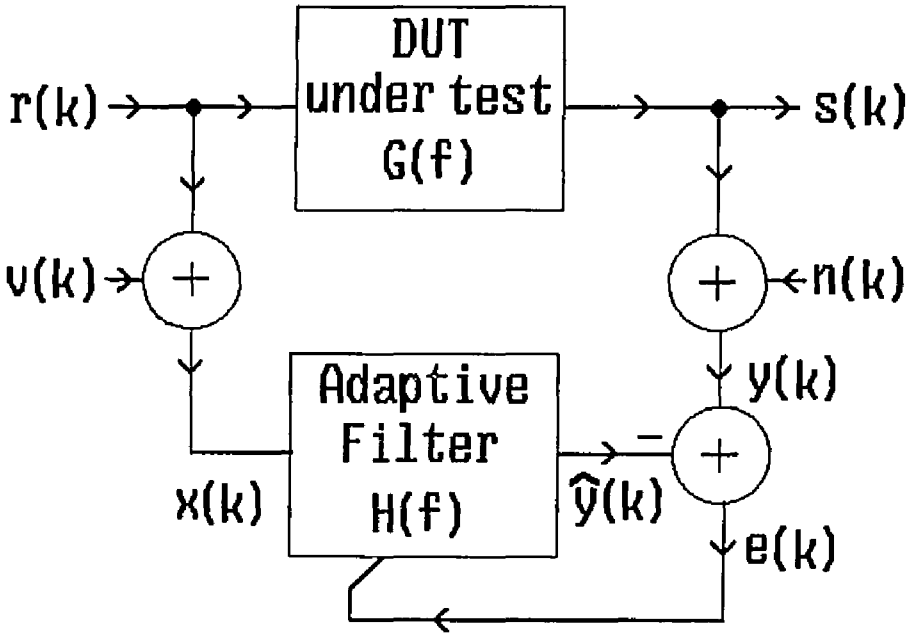
As shown in equation (8) the modulation phase shift is given by $1/2\cdot(\theta[\omega_c+\omega_m] - \theta[\omega_c-\omega_m])$, where $\theta[\omega]$ is the network's phase shift as a function of ω.

The modulation envelope's phase shift can also be determined from the equivalent equation (6), repeated below, which now gives the envelope's phase shift in terms of the phase shift of the upper and lower modulation sidebands through the DUT.

$$A_l\cdot\cos((\omega_c-\omega_m)+\theta[\omega_c-\omega_m]) + A_u\cdot\cos((\omega_c+\omega_m)+\theta[\omega_c+\omega_m]) \quad (6)$$

The SIM attempts to replicate the frequency response of the DUT, at the upper and lower modulation sideband frequencies, by adaptively creating a filter that gives this frequency response. By evaluating the frequency response of the adaptive filter after a suitable degree of convergence has been achieved the modulation envelope's phase shift can be found from the relation $\theta_{env} = 1/2 \cdot (\theta[\omega_c + \omega_m] - \theta[\omega_c - \omega_m])$. The carrier frequency response is not used, implying that the modulation type need only provide a pair of symmetrical sidebands. The block diagram of the SIM is shown in figure 11 below with separate noise sources added to both the reference channel and the DUT channel.

Figure 11; System Identification Method Block Diagram



XIV.A. Adaptive Filter Requirements

The system, or DUT, to be identified has an unknown transfer function G(f). The optimum linear filter, H(f), for producing the best linear mean square error estimate $\hat{s}$, is the Wiener filter. The filter H(f) need match the frequency response of G(f) only in the frequencies of interest, namely at the upper and lower modulation sideband frequencies. Since the system identification method only "identifies" the unknown system at select frequencies the term "pseudo-system identification" may be more appropriate. This relaxed requirement allows the filter H(f) to be chosen to replicate G(f) with many degrees of freedom for filter structure and filter order. If the frequency response was needed continuously over a broad range a highly constrained solution to the required structure would result. This would occur, for example, if the modulated carrier was replaced by white noise. It is possible that a structure for H(f) actually duplicating G(f) may not be implementable as an adaptive filter. This could be due to an excessive number of filter coefficients needed or instabilities occurring during the adaptive convergence.

Two popular filter topologies exist; lattice and transversal. The transversal topology was chosen since it has the advantage of enabling the frequency response of the filter to be evaluated directly through its impulse response coefficients. The lattice filter requires a conversion process, known as Levinson-Durbin recursion, to isolate the filter's impulse response coefficients from the lattice filter's reflection coefficients. The transversal filter itself has two structures; FIR and IIR. The FIR case is unconditionally stable since it has all zeros and no poles and therefore is suitable for use in the implementation of an adaptive filter where the coefficients do not have predetermined values.

The stability and response of a digital filter is best analyzed by using the **z transform** of the filter's impulse response defined by:

$$H(z) = Z[h(n)] = \sum_{n=-\infty}^{\infty} h(n) z^{-n} \tag{42}$$

A test for a (causal) digital filter's stability can be found by examining the poles of the filter's z transform. The poles must lie within the unit circle in the complex z plane.

The Fourier transform of the filter's impulse response gives its frequency domain transfer function. For a FIR filter its impulse response is given directly by the filter coefficients. The Fourier transform is the z transform evaluated on the unit circle in the complex z plane. The frequency response, $H(e^{j\omega T})$, of a digital filter is then found by replacing z with $e^{j\omega T}$, thus evaluating z on the unit circle [12]. The result is a complex number giving the amplitude and phase of the filter at frequency ω.

The order, or degree, of the adaptive FIR filter is chosen based upon the frequency resolution needed. A FIR filter normally produces transmission zeros which give a bandstop characteristic in the filter's magnitude response at the zero frequencies. This means that the FIR filter order may be large to approximate a bandpass characteristic in the filter's magnitude response. Sinusoidal type signals, like a modulated carrier for example, require a filter with a bandpass characteristic for transmission. Another requirement for the order of the filter is that the filter delay, equal to the number of filter taps, N, multiplied by the sampling period, must be at least as long as the delay of the reference signal through the DUT [13,p421]. Yet another influence on the filter order is the fact that the various frequency components, i.e. carrier and sidebands, are simultaneously filtered by the same filter. The discrete frequencies can influence

each others filtering response by a cross-coupling effect on the filter's impulse response solution [14]. The cross coupling effect is reduced by increasing the number of filter coefficients. This effect is noticeable especially when the amplitude of one sinusoid dominates another sinusoid of interest that is near in frequency.

XIV.B. The Wiener Filter Solution

Unlike the adaptive filter which produces the optimum filter through an iterative "learning" process, the Wiener filter is a fixed or nonadaptive filter whose coefficients are calculated through knowledge of the DUT's input/output cross correlation vector and input autocorrelation matrix. The adaptive filter produces the optimal result without knowledge of the statistical properties of the DUT's input and output signals. The Wiener filter solution is approached asymptotically in the adaptive filter with the true solution reached only in the limit, i.e. with infinite data.

The following derivation of the Wiener filter solution to the SIM provides some insight as to the effects of the noise sources on both the reference channel and DUT channel as shown in figure 11.

The transversal filter's input/output relationship was shown in equation (21) and is repeated below for the FIR case with N filter coefficients.

$$y(n) = \sum_{i=0}^{N-1} h_i \cdot x(n-i) \tag{43}$$

The above filtering, or convolution, operation can be equivalently described as an inner product of vectors as $y(k) = \mathbf{h}^T \cdot \mathbf{x}(k)$. Where the data vector $\mathbf{x}$ at sample k is $\mathbf{x}(k) = [x(k)\ x(k-1)\ \cdots\ x(k-N+1)]^T$ and the impulse response or coefficient vector $\mathbf{h} = [h_0\ h_1\ h_2\ \cdots\ h_{N-1}]$.
The input, $x(k) = r(k) + v(k)$, and output, $y(k) = s(k) + n(k)$, to the Wiener filter are shown in figure 11. The noise signals, v(k) and n(k), have variances of σ_v^2 and σ_n^2 respectively. For the purposes of the derivation, the signals r(k), v(k) and n(k) are assumed to be wide sense stationary mutually uncorrelated random processes. This is realistic assuming the noise sources v(k), n(k) are independently generated.

The impulse response vector, h_{opt}, of the Wiener filter is optimized according to the Minimum Mean Square Error (MMSE) criterion given by: $\mathbf{E}[(y(k)-\hat{y}(k))^2]$ [15,p23]. **E** is the discrete expectation operator defined as:

$$E[X] \triangleq \sum_{i=-\infty}^{\infty} x_i \cdot p_X(x_i) \ , \ where\ p_X(x_i) = Prob[X = x_i] \tag{44}$$

It can be shown, [15,p21], that the optimal FIR filter has an impulse response vector of

$$\mathbf{h}_{opt} = \phi_{xx}^{-1} \cdot \phi_{xy}. \tag{45}$$

The autocorrelation matrix ϕ_{xx} is given by:

$$\phi_{xx} = \phi_{rr} + \sigma_v^2 \cdot \mathbf{I}_N \tag{46}$$

Where ϕ_{rr} is defined by $\mathbf{E}[\mathbf{r}(k)\mathbf{r}^T(k)]$. $\mathbf{I}_N$ is the N by N identity matrix.

The cross correlation vector is given by

$$\phi_{xy} = \phi_{rs} \tag{47}$$

This result from equation (47) is due to the zero mean uncorrelated noise sources v(k), n(k) which give zero values when cross correlated. By substituting the equations (46) and (47) into equation (45) the solution can be shown, REF[15,p24], to be:

$$\mathbf{h}_{opt} = \mathbf{g} - \sigma_v^2 \cdot (\phi_{rr} + \sigma_v^2 \cdot \mathbf{I}_N)^{-1} \cdot \mathbf{g}, \tag{48}$$

Where g is the impulse response vector of the DUT under identification.

The results of equation (48) show that the Wiener filter's impulse response, $\mathbf{h}_{opt}$ is a biased estimate of g, the impulse response under identification. This bias is due to the noise source v(k) on the reference channel. Also shown is that the noise source n(k) added to the DUT channel does not affect the estimate.

XIV.C. The Correlation Canceler Approach

An intuitive approach to the adaptive filtering process can be made using the **correlation canceling** concept [13,p9]. The adaptive filter, as shown in figure 11 above, is an optimum filter which converges to the Wiener filter solution. The adaptive filter attempts, through a convergence process, to minimize the error *e(k)* between the DUT channel's signal *y(k)*, and the adaptive filter's output signal $\hat{y}(k)$ which is an estimate of *y(k)*. The process of minimizing this error is called correlation canceling and results in the Wiener filter solution described above. Given the DUT's signal with noise, *y(k)*, and the filter's input from the reference channel, *x(k)*,

then the optimum filter's output, $\hat{s}(k)$, will be a signal maximally correlated with $y(k)$. To create an optimum estimate the error $e(k)$ is formed by subtracting the estimated signal $\hat{s}(k)$ from $y(k)$ which cancels the correlated part of $x(k)$ from $y(k)$. The difference, $e(k)$, is uncorrelated with, or orthogonal to $x(k)$. The filter coefficients, or weights, are adaptively adjusted so as to minimize the signal power in $e(k)$.

XIV.D. The Stochastic Gradient Algorithm

A number of methods exist for producing an optimum estimate of $s(k)$, $\hat{s}(k)$, from $y(k)$. Two of the most popular methods that are implementable using an adaptive filter are the Least Mean Square, or LMS, and the Recursive Least Square, or RLS algorithms.

The LMS algorithm is based on the stochastic gradient descent approach. The main advantage the LMS algorithm has over the RLS algorithm is its simplicity. The LMS algorithm was included for comparison purposes only since its convergence performance is inferior to the RLS algorithm for the modulated carrier signals encountered. This is due to the large range of the eigenvalues of the input signals autocorrelation matrix for correlated, i.e., sinusoidal signals. The convergence time constant for the LMS algorithm is shown in [15,p44] to be, at a minimum, one half times the ratio of the maximum and minimum eigenvalues.

The basic principle behind the LMS algorithm is in the use of the gradient of the variable to be optimized [13,p411]. For the case of the adaptive filter it is the filter coefficients that are optimized, adaptively, to minimize the error e(k). The gradient of the filter coefficients gives the proper direction and magnitude of the incremental step that will minimize the error. The filter coefficient updating equation then becomes:

$$\bar{h}(k+1) = \bar{h}(k) - \mu \cdot \frac{\partial \left[E[(\bar{y}(k) - \bar{h}^T(k) \cdot \bar{x}(k))^2] \right]}{\partial h} \qquad (49)$$

In the implementation of the algorithm the expression for updating the filter coefficients ignores the expectation operator used in equation (49) and replaces it with the instantaneous gradient giving:

$$\bar{h}(k+1) = \bar{h}(k) - 2\mu \cdot e(k) \cdot \bar{x}(k) \ , \quad \textit{where } e(k) = y(k) - \bar{h}^T(k) \cdot \bar{x}(k) \qquad (50)$$

The parameter μ, shown in equation (50), controls the convergence rate and is normally maximized. The upper limit for μ is must not exceed the inverse of the number of coefficients times the adaptive filter's input signal power [16,p99]. Exceeding this value can cause the adaptive filter to become unstable and the solution will diverge instead of converge. Another consideration in choosing the upper limit for μ is the fact that μ also controls the degree that the coefficients fluctuate, or dither, around the optimum, or Wiener, solution [16,p103]. The MATLAB program implementing the LMS algorithm that was used in generating the figures 13,15 and 17 is located in appendix I.f.

XIV.E. The Recursive Least Squares Algorithm

The fast version of the RLS algorithm, found in [13,p545], was implemented for the linear system identification needs of this section. The RLS algorithm does not have the eigenvalue dependency found in the LMS algorithm. The RLS algorithm can be shown to produce an unbiased estimate, and if the noise source $n(k)$ is Gaussian, the result is a minimum variance unbiased estimate. This result is expected since the RLS algorithm produces, at convergence, the Wiener filter solution. The RLS algorithm is based on the Least Squares Error, or LSE, criterion which is similar to the MMSE criterion. Since the data set size is limited to a fixed number of samples, M, the ensemble expectation operator **E** found in the MMSE criterion is replaced by a finite summation. The LSE criterion is then defined [15,p28] as the minimization of:

$$\sum_{k=0}^{M-1} (y(k) - \hat{y}(k))^2 \tag{51}$$

Similarly the sampled autocorrelation matrix and the sampled cross correlation vector can be defined using summations as:

$$r_{xx}(M) = \sum_{k=0}^{M-1} \bar{x}(k)\bar{x}^T(k) \;, \quad r_{xy}(M) = \sum_{k=0}^{M-1} \bar{x}(k)\,y(k) \tag{52}$$

Note that the sampled autocorrelation matrix $\mathbf{r}_{xx}$ is not to be confused with the signal r(t). Analogously, the optimum impulse response is then:

$$\mathbf{h}_{opt}(M) = \mathbf{r}_{xx}^{-1}(M) \cdot \mathbf{r}_{xy}(M) \qquad (53)$$

The fast RLS algorithms exploit the fact that for a filter of length N the data vector x(k) has N-1 common elements from one sample interval to another. This gives redundancy to the sampled inverse autocorrelation matrix, $\mathbf{r}_{xx}^{-1}(M)$, which can then be recursively updated with the new data only. This redundancy is also true of the cross correlation vector $\mathbf{r}_{xy}(M)$. Forward and backward prediction of the data entering and leaving the "window" of N coefficients in the prediction filter give the a priori and a posteriori errors respectively. These prediction errors are then used to reduce the computation of $\mathbf{r}_{xx}^{-1}(M)$. This is possible due to the fact that $\mathbf{r}_{xx}^{-1}(M)$ is completely represented by the forward prediction error power and the prediction filter coefficients [16,p150].

The RLS algorithm has two adjustable parameters called delta and lambda. Delta is the initial prediction value and is used to avoid a divide by zero upon algorithm start-up. Delta should be made as small as possible so as not to cause an initial convergence slowdown. It can be shown that a value for delta that is greater than N·(input signal power)/lambda will guarantee stability during convergence [16,p190]. Lambda is called the "forgetting factor" and is an exponential weighting factor for the input data. Lambda should be set equal to one for signals with stationary statistics. The MATLAB program implementing the fast RLS algorithm used for generating the figures 12,14 and 16 is located in appendix I.g. The MATLAB program implementing the SIM which uses the RLS algorithm is located in appendix I.h.

XIV.F. Performance Comparison of the RLS and LMS Algorithms

The graphs of figures 12 through 15 below show the convergence performance of the RLS verses LMS algorithms operating on the same DUT, a fifth order digital lowpass butterworth filter, with a data buffer length of 4096 points. The "carrier" frequency is 50Khz with a 10Khz AM modulation rate and a modulation index of 0.1. Both adaptive filters have 100 taps. The μ parameter for the LMS algorithm, explained in section XIV.d, was chosen for the run by finding the value of μ at which the algorithm started diverging and then using one half of this value for μ. The lambda parameter for the RLS algorithm was set to one since the data, in this case, is deterministic and therefore has stationary statistics. The delta parameter, which does not have an effect on the final converged value, was set to 0.001.

The graphs of figures 12 and 13 respectively show the RLS and LMS convergence error, e(k), as the algorithm operates on the buffer from beginning to end. The graphs of figures 14 and 15 respectively show the algorithm's final convergence error near the end of the buffer. The RLS error is approximately 10 times smaller. The LMS algorithm could reach the RLS performance level only with a longer data buffer. The graphs of figures 16 and 17 respectively show the phase response for the RLS and LMS adaptive filters after the data buffer has been processed. The filter response shows, for both algorithms, that the true DUT filter response matches the adaptive filter response only at the 40Khz lower modulation sideband, the 50Khz carrier and the 60Khz upper modulation sideband. It is only these frequencies that place constraints on the solution to the adaptive filter coefficients.

Figure 12; RLS Convergence Error

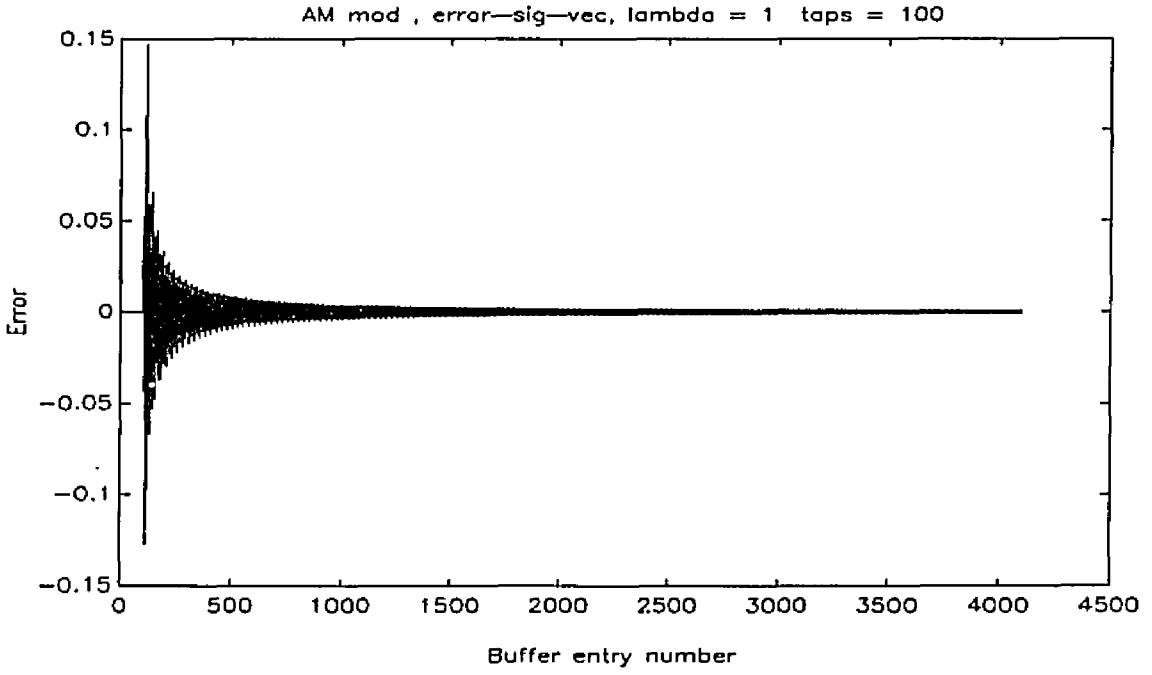


Figure 13; LMS Convergence Error

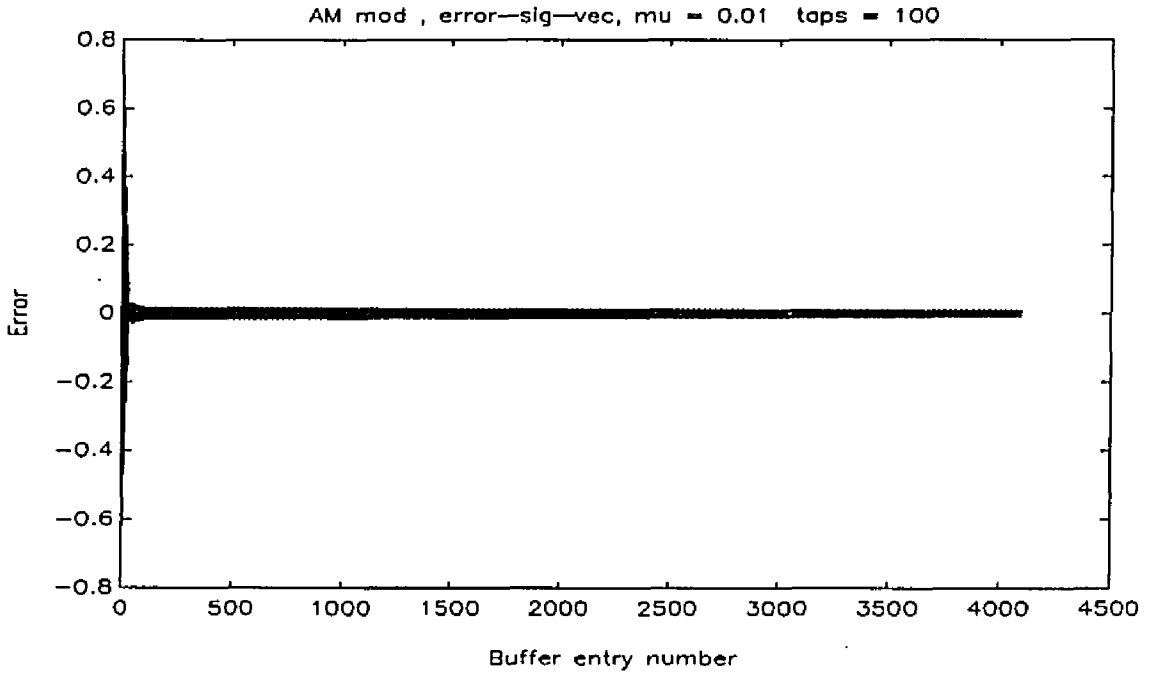


Figure 14; RLS Convergence Error at End of Data Buffer

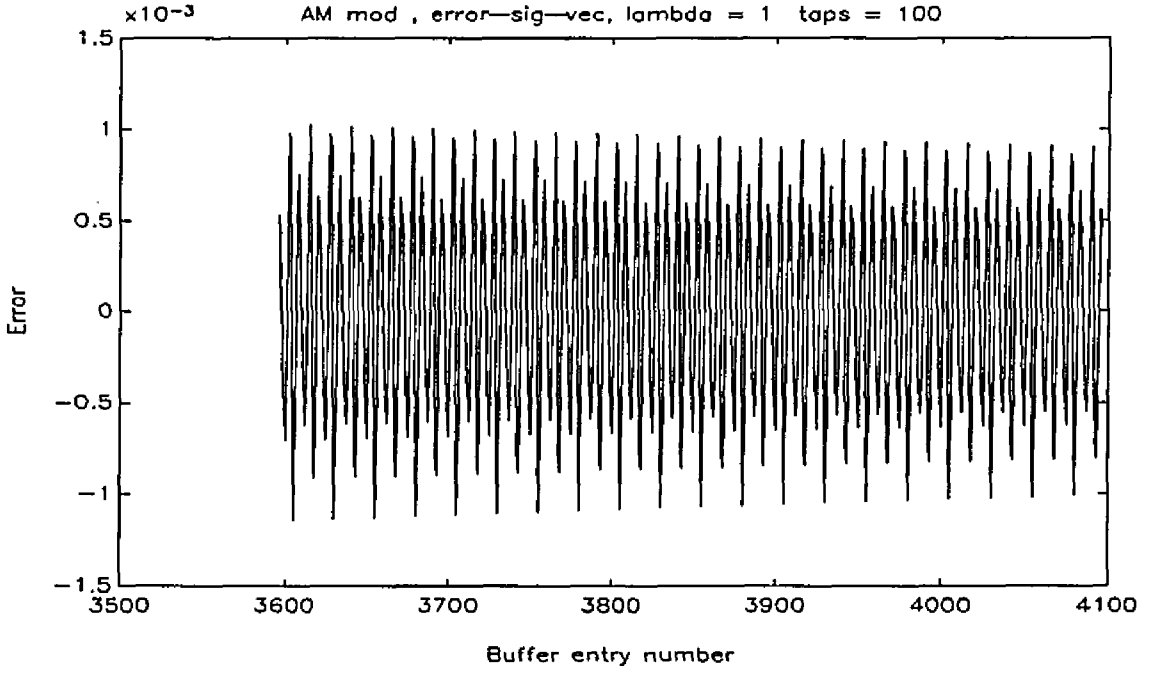


Figure 15; LMS Convergence Error at End of Data Buffer

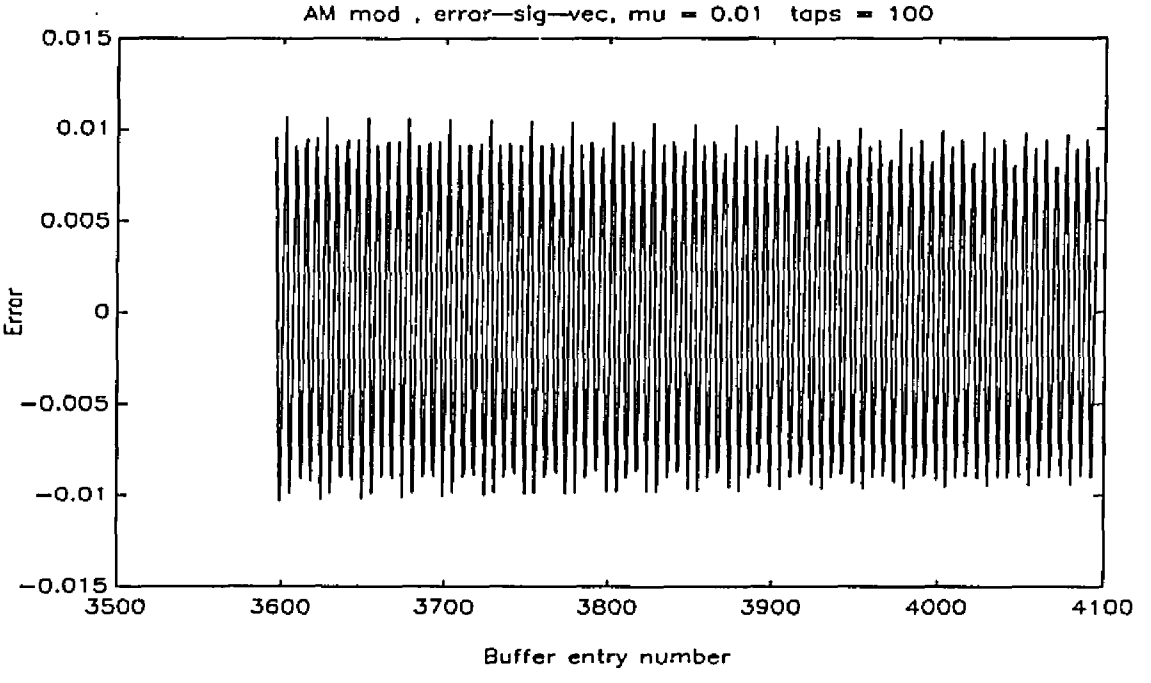


Figure 16; RLS Adaptive Filter Phase and DUT Filter Phase

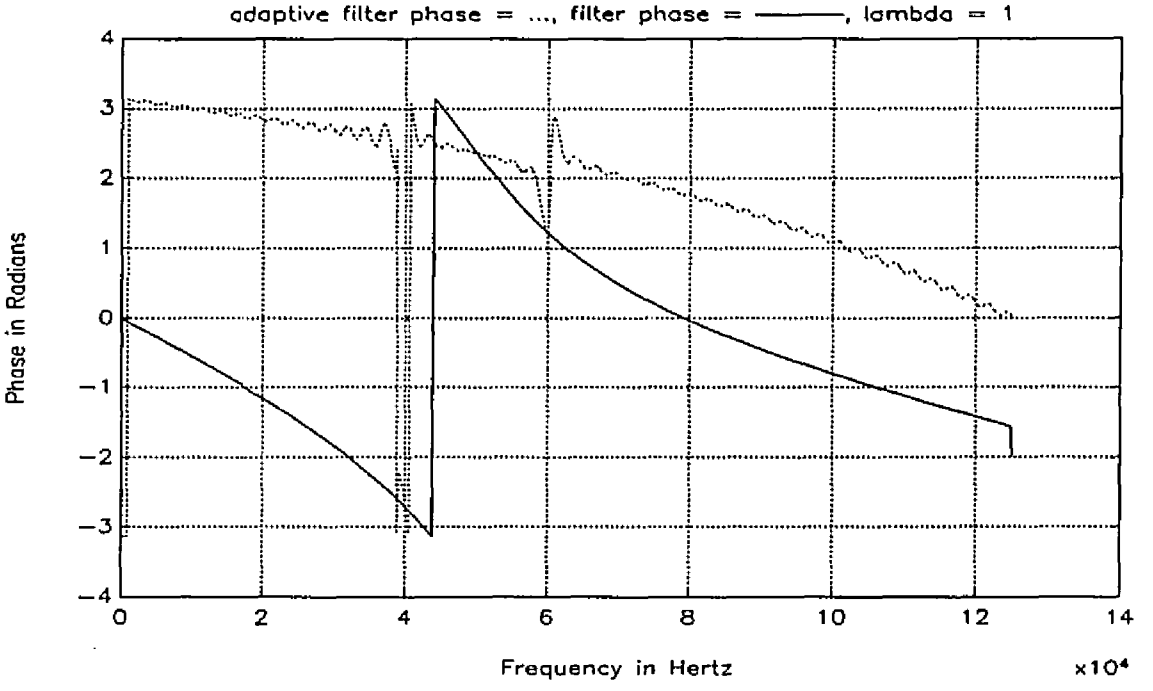
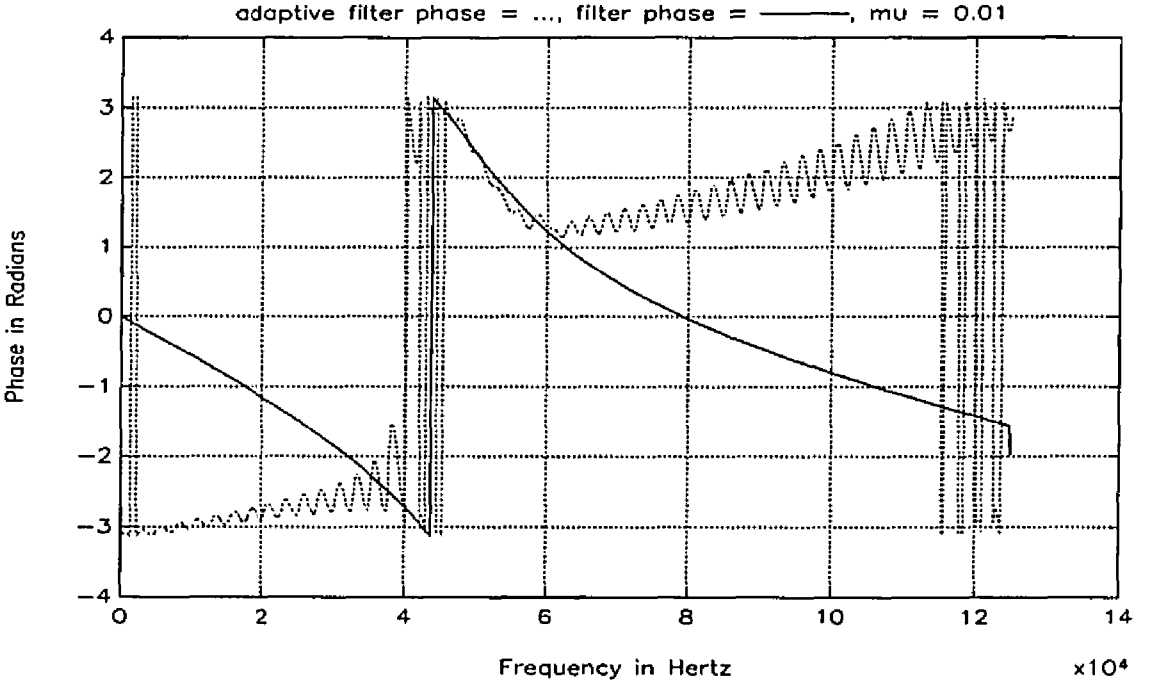


Figure 17; LMS Adaptive Filter Phase and DUT Filter Phase



XV. INTRODUCTION TO PART II

Part II, section XVI of this thesis develops a procedure, or algorithm, called nonlinear system identification capable of measuring the characteristics of a nonlinear system. The nonlinear system of interest is assumed to be present in the measurement system and is the source of unwanted nonlinear distortion.

In section XVII the inverse of the "identified" nonlinear system is created and then used to cancel, or equalize, the deleterious effects of the original nonlinearity.

XV.A. Nonlinear System Simulation

The nonlinear system used for part II of this thesis system represents, hypothetically, the measurement system's antialiasing filter followed by a buffer amplifier. These circuits would be located between the final IF filter and the analog to digital converter section's sample and hold.

The antialiasing filter is modeled by a linear (first order) system that is producing linear distortion as described in section IX.b. Although the antialiasing filter will be an analog filter in the actual measurement system the simulated filter is a digital filter for the reasons given in section XI.a. The filter was chosen to have a lowpass frequency response, characteristic of an antialiasing filter. The magnitude response of this filter was chosen to introduce significant linear distortion to the IF signal. Another requirement of the filter, needed to create a nonlinear equalizer, is that it have a stable inverse. This requirement is met by keeping the magnitude of the zeros of the filter's z transform to less than one. This results in the poles of the inverse filter that are within the unit circle of the z domain's complex plane thus giving a stable inverse filter.

One filter that meets all of the above requirements has a z transform as follows:

$$H(z) = 0.3 + 0.4z^{-1} + 0.3z^{-2} + 0.2z^{-3} + 0.1z^{-4} \qquad (54)$$

This filter has zeros at 0.0564 +/- j·0.7132 and at - 0.7231 +/- j·3584 thus their magnitudes are less than one. The frequency response of this linear filter is shown below in figures 18 and 19.

Figure 18; Magnitude Response of the Linear Filter of Equation (54)

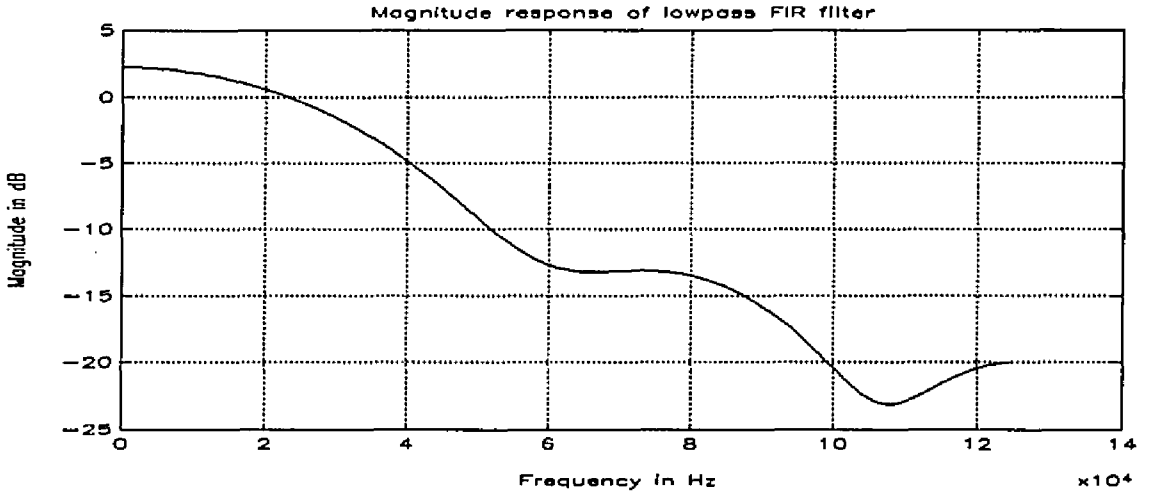
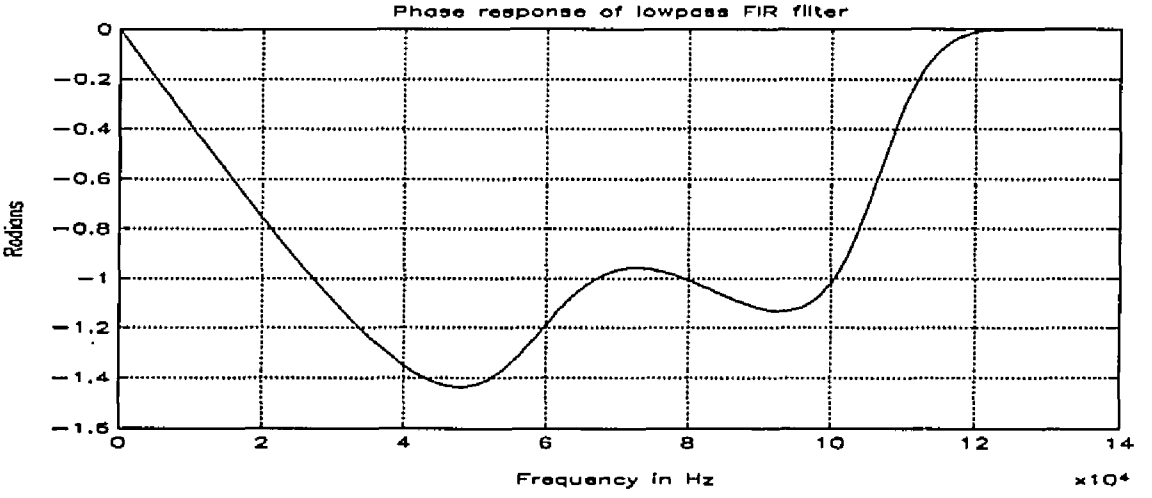


Figure 19; Phase Response of the Linear Filter of Equation (54)

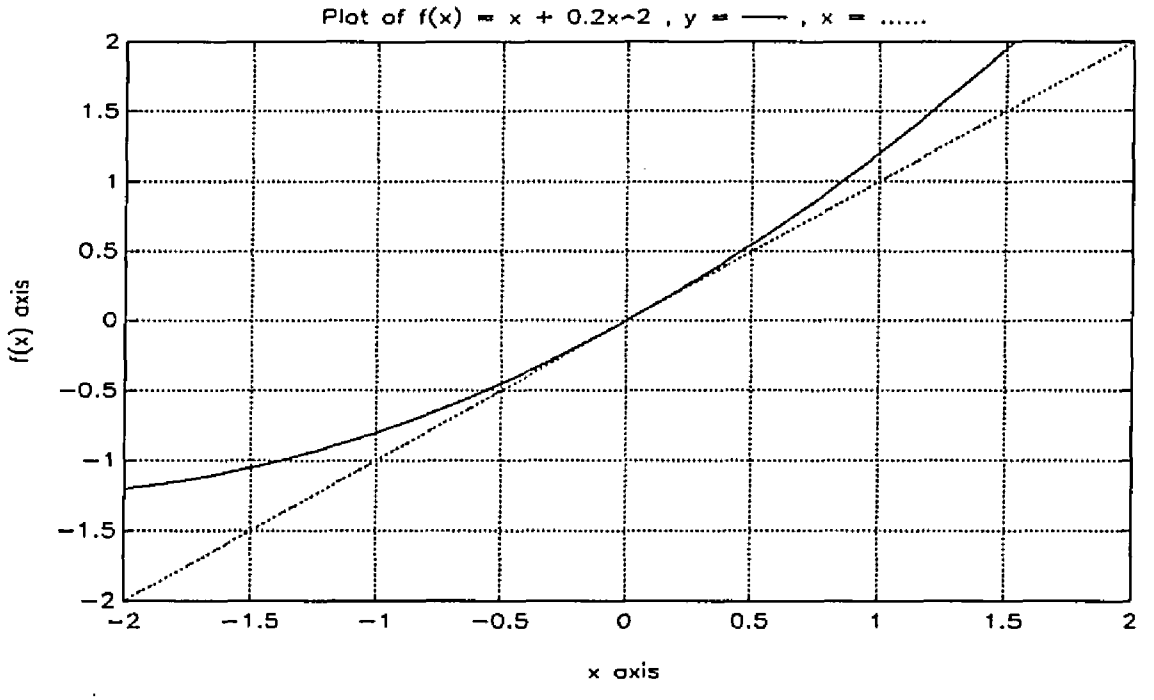


The buffer amplifier is modeled by a second order system of the form:

$$f(x) = x + c \cdot x^2 \qquad (55)$$

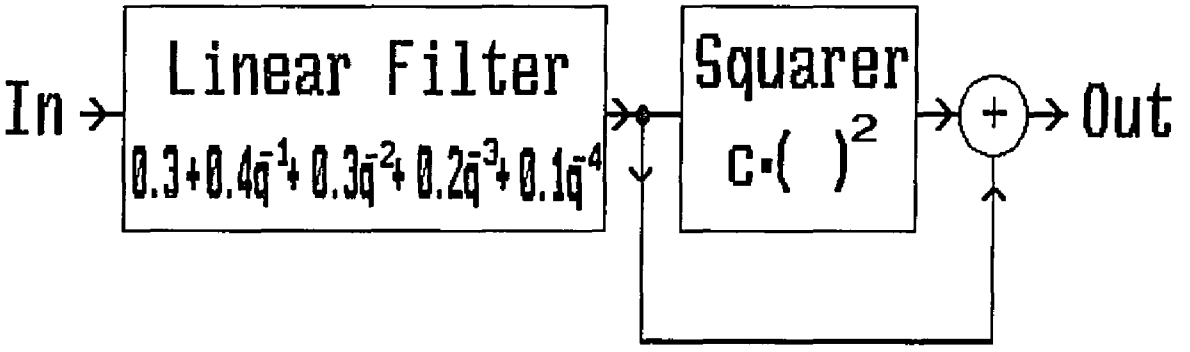
The second order coefficient, **c**, was chosen to be 0.2. The graph of this nonlinearity is shown below in figure 20 plotted along with the linear line $f(x) = x$.

Figure 20; Graph of the Second Order Nonlinearity from Equation (55)



The nonlinear system is formed by cascading the linear filter of equation (54) with the second order system of equation (55). This will produce both linear and second order distortion in the simulated measurement system. This second order system is shown in figure 21 below. The "q^{-1}" operator represents a sample delay in the time domain, thus q^{-2} indicates two delays.

Figure 21; Second Order System Composed of Equation (54) and Equation (55)



XVI. NONLINEAR SYSTEM IDENTIFICATION

XVI.A. The Volterra Series Representation of Nonlinear Systems

Nonlinear systems that have memory, or equivalently, systems with energy storing elements, cannot be adequately represented by a power series such as a Taylor series. The introduction of a power series expressed in terms of convolution integrals gives rise to a power series with memory called a **Volterra series**. The Volterra series is useful for describing time invariant systems with "mild" nonlinearities. The nonlinearities are "mild" in the sense that they must be continuous, cannot be multiple valued such as is present in a system with hysteresis, and the nonlinearity cannot have an infinite memory such as would be found in a digital flip-flop [17]. Most analog circuits encountered, however, were intended to be approximately linear in operation and are therefore well suited for representation by the Volterra series.

Due to computational considerations the order of the series will normally be limited to the minimum number of terms that will adequately approximate the nonlinear system of interest.

The Volterra representation between the input *x(t)* and the output *y(t)* of a time invariant nonlinear system with memory is described as a summation of n convolution integrals of one through n-dimensions:

$$y(t) = \int_{-\infty}^{\infty} h_1(\tau_1)x(t-\tau_1)d\tau_1 + \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_2(\tau_1,\tau_2)x(t-\tau_1)x(t-\tau_2)d\tau_1 d\tau_2$$
$$+ \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_3(\tau_1,\tau_2,\tau_3)x(t-\tau_1)x(t-\tau_2)x(t-\tau_3)d\tau_1 d\tau_2 d\tau_3 \cdots \qquad (56)$$
$$+ \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} h_n(\tau_1,\tau_2,\cdots,\tau_n)x(t-\tau_1)x(t-\tau_2)\cdots x(t-\tau_n)d\tau_1 d\tau_2\cdots\tau_n$$

The nth degree integral is called the nth order **Volterra operator** and is denoted by $\mathbf{H}_n$. A nth order Volterra system is then composed of the sum of the operators $\mathbf{H}_1$ through $\mathbf{H}_n$. The $h_n(\tau_1,\tau_2\cdots,\tau_n)$ found in the operator's integrand are called the **Volterra kernels**. The first order kernel, $h_1(\tau_1)$, is equivalent to the linear system impulse response. The lower order kernels can be derived from the higher order kernels along the line where $\tau_1 = \tau_2$. If the kernel is **symmetric** then the order of the τ's are unimportant. The symmetric kernel is unique in the sense that there is only one symmetric kernel that can represent a given Volterra system's response. If a kernel is **separable** then the nth order kernel can be expressed as the product of n first order kernels or $h_n(\tau_1,\tau_2\cdots,\tau_n) = h_1(\tau_1)h_2(\tau_2)\cdots h_n(\tau_n)$.

Analogous to linear system theory a nth order **kernel transform**, $\mathbf{H}(j\omega_1,j\omega_2,\cdots,j\omega_n)$ can be defined which is the nth dimensional Fourier transform of the nth order Volterra kernel. As expected, the kernel transform $\mathbf{H}_1(j\omega_1)$ gives the familiar linear system frequency response.

XVI.B. The Second Order Discrete Volterra System

For digital signals, as encountered in the digital buffer of the measurement system, the discrete time equivalent of the continuous time Volterra series is needed. The discrete time Volterra series can be found by replacing the integrals, as shown in equation (56), with summations. A second order discrete time Volterra series with symmetric kernels is then:

$$y(k) = \sum_{i=0}^{N-1} a_i x(k-i) + \sum_{i=0}^{N-1} \sum_{j=i}^{N-1} b_{i,j} x(k-i)x(k-j) \qquad (57)$$

The first summation represents the first order operator $\mathbf{H}_1$ and the double summation represents the second order operator $\mathbf{H}_2$. The sum of $\mathbf{H}_1$ and $\mathbf{H}_2$ represents the complete second order system. The discrete indices i and j replace the τ's from equation (56). The first degree kernel of equation (57) is denoted by a_i and the second degree kernel is $b_{i,j}$. These kernels are functions of time since they will be "adapted" to their final value using an adaptive algorithm.

The nonlinear system **G** , whose parameters are to be identified, was shown in figure 21. This system is composed of a linear filter followed by a (memoryless) squarer. The identification of the linear filter coefficients is made from the first order kernel a_i and identification of the second order coefficient, c, is made from the second order kernel $b_{i,j}$. Different second order structures can be generated from the Volterra kernels, a_i and $b_{i,j}$. However, all of these structures will have the same input/output characteristics since they are based on the same Volterra kernels. An alternative structure to figure 21, giving the same kernels, would be composed of two identical first order systems in parallel, described by a_i. The second order output of the system is the sum of the outputs from a first order system and paralleled system.

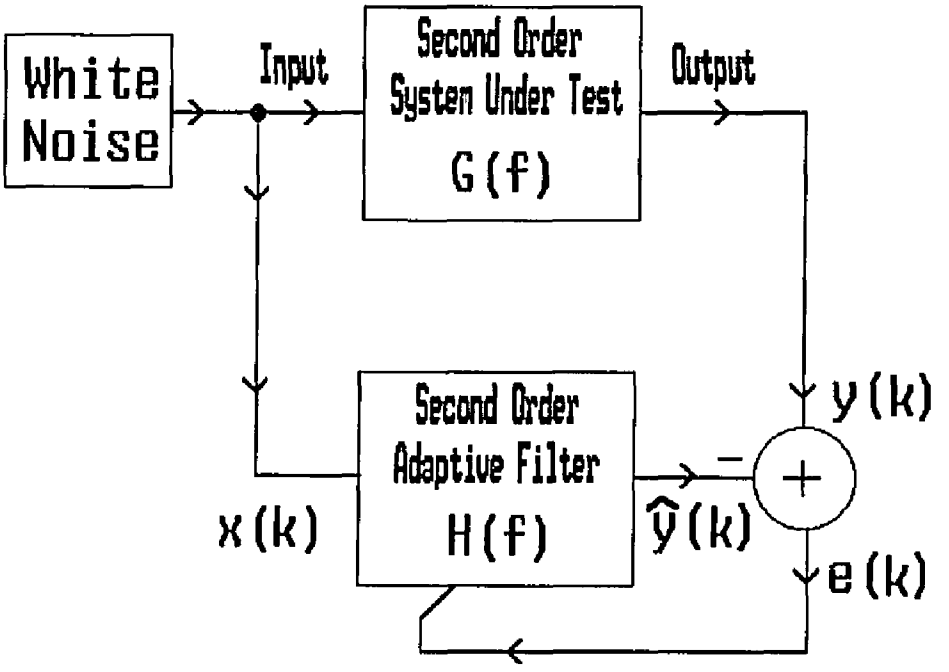
XVI.C. Second Order System Identification

Analogous to the linear system identification method described in section XIV the second order system can be identified using an adaptive filter. The adaptive filter must now also be of second order to "match" the system under identification. Equation (57) which describes a discrete second order Volterra system, is readily implementable as an adaptive filter. The Volterra series, described by equation (57), need only have the limits of the indices extend as long as the memory of the system under identification. This memory is four samples, or lags, deep in the linear FIR filter of equation (54). By contrast, an IIR filter would require a Volterra series with a long "memory" to capture the system's long impulse response.

The adaptive filter, taking the form of equation (57), is connected as shown in figure 22. Note that the linear system identification block diagram of figure 11 in section XIV was connected in the same fashion. The main difference between figure 11 and figure 22 is that the input signal, x(k), is now white noise instead of the modulated carrier. Using white noise as the input signal will force a solution of the Volterra kernels with the best possible match, at all frequencies, between the second order adaptive filter and the second order system under identification.

The LMS algorithm was chosen as the adaptive algorithm for several reasons. The first reason is that since the input signal is white noise the eigenvalues of the signal's input autocorrelation matrix are all the same indicating the LMS algorithm will give good performance as indicated in section XIV.d. Another reason is that for the two dimensional case encountered here the LMS algorithm still retains its simplicity of implementation.

Figure 22; Second Order System Identification Block Diagram



By arranging the processed input data and the adaptive filter coefficients into specially arranged vectors, **X** and **H** respectively, the second order adaptive filter's input/output relationship is simply [18]:

$$\hat{y}(k) = H^T \cdot X \tag{58}$$

The updating expression for the second order adaptive filter coefficients, using the LMS algorithm as described in section XIV.d, is then:

$$H(k+1) = H(k) - 2\mu \cdot e(k) \cdot X(k) \ , \quad where \ e(k) = y(k) - \hat{y}(k) \tag{59}$$

As shown in equations (58) and (59) the input vector **X** is arranged so that a simple dot product between it and the coefficient vector **H** will result in the estimated output $\hat{y}(k)$. As an example, using the case where N = 3, the input data vector would be preprocessed into the following form:

$$\mathbf{X} = [\, x(k),\ x^2(k),\ x(k)\cdot x(k\text{-}1),\ x(k)\cdot x(k\text{-}2),\ x(k\text{-}1),\ x^2(k\text{-}1),\ x(k\text{-}1)\cdot x(k\text{-}2),\ x(k\text{-}2),\ x^2(k\text{-}2)\,] \quad (60)$$

Following the format of the input vector **X**, for N = 3, the adaptation process will force the coefficients vector **H** into the following format:

$$\mathbf{H} = [\, a_0,\ b_{0,0},\ b_{0,1},\ b_{0,2},\ a_1,\ b_{1,1},\ b_{1,2},\ a_2,\ b_{2,2}\,] \quad (61)$$

The coefficients $b_{1,0}$, $b_{2,0}$, and $b_{2,1}$ are not needed because the kernel $\mathbf{h}_2$ formed is symmetric.

Both the **X** and **H** vectors have a length of $N\cdot(N+3)/2$ because the kernels produced are symmetric. N is the number of linear coefficients, or the lower to upper summation index span of the $\mathbf{H}_1$ operator. The number of second order coefficients is then given by $N\cdot(N+3)/2 - N$.

The **X** vector organization algorithm and the LMS adaptive filter algorithm were implemented as a "C" language subroutine for an execution speed advantage over the equivalent MATLAB code. The MATLAB main program implementing the nonlinear system identification algorithm is located in appendix II.a.

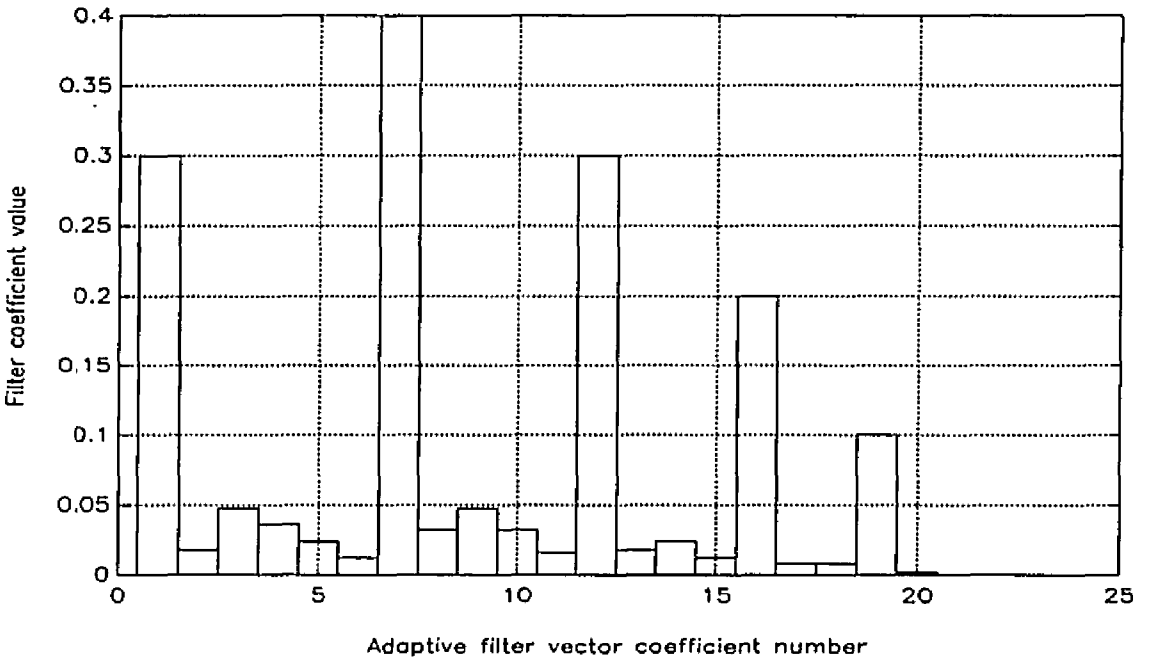
XVI.D. Second Order System Identification Results

The second order system under consideration, shown in figure 21, was used as the DUT in the configuration of figure 22. Using white gaussian noise as the input signal, x(k), the algorithm of section XVI.c was applied to a buffer length of 8192 points. The final convergence error, e(k), was less than 1E-15 indicating an excellent duplication of the DUT filter coefficients by the adaptive filter. The adaptive filter vector, **H**, is shown in figure 23 below after convergence. The number of filter coefficients, $N\cdot(N+3)/2 = 25$ for $N = 5$, was chosen with a priori information. This was done for presentation purposes only since a longer filter vector will have the surplus coefficients converge to zero.

The linear coefficients have specific locations in the vector **H** and represent a_i, where i = 1 to N. For the vector **H** shown in figure 23 the linear coefficients are at **H**[1], **H**[7], **H**[12], **H**[16] and **H**[19] which correspond to the DUT's filter delays of 0, 1, 2, 3, and 4 respectively. Notice that the values of **H** at these locations corresponds to the linear filter of figure 21.

The second order coefficients are located between the linear coefficients in vector **H**. The second order coefficient without a delay, $b_{0,0}$, is at **H**[2]. This coefficient results from multiplying the input signal by the linear filter's first (delayless) coefficient which is equal to 0.3, squaring this product and finally multiplying this result by the second order squarer coefficient, **c**, equal to 0.2. Since the linear coefficients have already been identified it remains to isolate the second order squarer coefficient, **c**, using the relationship $\mathbf{c} = \mathbf{H}[2]/(0.3^2)$. The second order system parameters have now been fully identified at which point this information will be used in section XVII to create an inverse system for equalization purposes.

Figure 23; Entries of the Second Order Adaptive Coefficient Vector H After Convergence



XVII. NONLINEAR EQUALIZATION

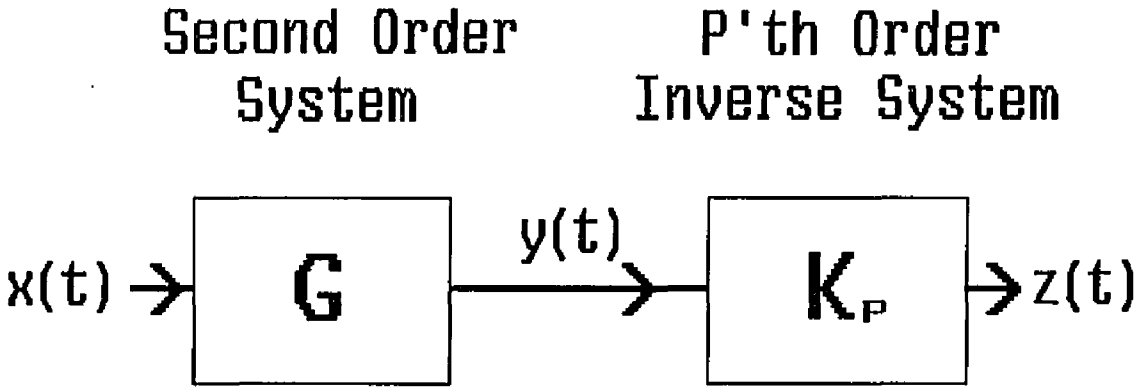
Although the simulated measurement system can be made "perfect" the real measurement system will possess all of the imperfections, to some degree, listed in section IX. This section on nonlinear equalization will develop a method to cancel, or equalize, the effects of the second order nonlinear system **G**, discussed in section XV. This second order system hypothetically models the antialiasing filter followed by a buffer amplifier in the actual measurement system. The antialiasing filter introduces linear distortion while the buffer amplifier introduces second order distortion. The effect of the nonlinear equalizer is to deconvolve the unwanted nonlinear response of this second order system, **G**, by creating a good approximation to the inverse system $\mathbf{G}^{-1}$. This leaves the measurement system nearly distortion free so as to maximize the accuracy of the phase estimation methods.

In section XIX the results of the phase estimation methods developed in part I are examined with the second order system in place. The phase estimation results are compared with, and without, the nonlinear equalizer in use.

XVII.A. The P'th Order Inverse of a Volterra System

The inverse of a Volterra system **G** can be approximated, to any degree desired, by finding the **p'th order inverse** of **G** . This p'th order inverse will be denoted by **K**(p). The true inverse of **G** is approached when the order p, of **K**(p), goes to infinity. The cascade connection of the system and its approximated inverse is shown below in figure 24.

Figure 24; Second Order System Cascaded with it's P'th Order Inverse



The composite system of **G**⊛**K**(p) is denoted by **Q** where ⊛ indicates convolution in the time domain. The desired output *z(t)* of the system of figure 24, for the purposes of linearizing **G**, is simply the input *x(t)*. The desired linear output can be achieved by requiring the first order kernel of **Q** to have a unit impulse response and the second through p'th order kernels of **Q** to be zero. The response of this desired system shown in operator notation is then:

$$z(t) = Q[x(t)] = x(t) + \sum_{n=p+1}^{\infty} Q_n[x(t)] \qquad (62)$$

As shown in equation (62) the effect of the kernels of **Q** higher than p are still present in the output *z(t)*. The extent to which these higher order kernels effect the desired linear output response depends on the amplitude of the input *x(t)* and the approximated inverse, **K**(p).

Satisfying the first condition needed of **Q**, i.e. $\mathbf{Q}_1[x(t)] = x(t)$ gives a requirement that:

$$K_1(s) = (G_1(s))^{-1} \tag{63}$$

Where $K_1(s)$ and $G_1(s)$ denote the Laplace transforms of the kernels k_1 and g_1 respectively [19,p131]. This requires that $G_1(s)$ have a stable inverse, a condition that was met in choosing this particular system **G** in section XVI. The remaining condition is that $\mathbf{Q}_2[x(t)]$ through $\mathbf{Q}_p[x(t)] = 0$. Satisfying this condition requires finding the operators K_2 through K_p.

XVII.B. The Third Order Inverse of the Second Order Volterra System

To find the p'th order inverse, **K**(p), the operators K_1 through K_p must be found. Due to computational reasons the order p is chosen to be just large enough to meet the minimum requirements for the linearization of **G**. In this thesis a third order inverse of **G** will be developed and its linearization effectiveness examined.

The requirements for the operators K_1 through K_p were given above as was the explicit expression for K_1. The required expression for K_2 is shown in [19,p132] to be:

$$K_2 = -K_1 \circledast G_2 \circledast K_1 \tag{64}$$

The above expression for K_2 can be separated into a memoryless nonlinear part, N_2, followed by the linear filter K_1 [19,p134]. For the particular second order system at hand N_2 is simply the negative of the second order nonlinearity itself or $N_2[y(t)] = - c \cdot y^2(t)$. Alternatively K_2 can be expressed as:

$$K_2 = K_1[-N_2[y(t)]] \tag{65}$$

The required expression for K_3 is shown in [20,p34] to be:

$$K_3 = -K_1 \circledast G_3 \circledast K_1 + K_2 + K_2 \circledast G_2 \circledast K_1 - K_2 \circledast (\mathbf{I} + G_2 \circledast K_1) \qquad (66)$$

The above expression for K_3 can be separated into a memoryless nonlinear part N_3 followed by the linear filter K_1 [19,p135]. Substituting K_2, equation (65), into the equation for K_3 and taking the nonlinear part for N_3 will give:

$$N_3 = -N_2[N_2[y(t)]] + N_2[N_2[y(t)] + y(t)] - N_2[y(t)] \qquad (67)$$

Replacing N_2 with $-c \cdot y^2(t)$ and simplifying will give:

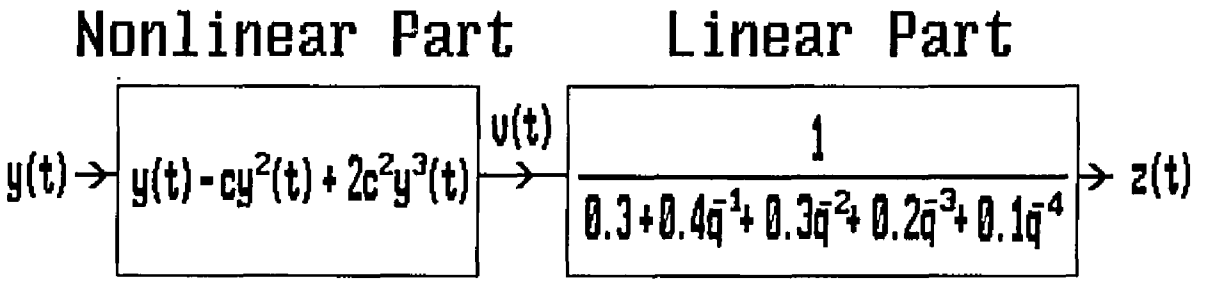
$$N_3[y(t)] = 2 \cdot c^2 \cdot y^3(t) \qquad (68)$$

The third order inverse system, **K**(3), is the sum of its operators K_1,K_2 and K_3. The system **K**(3) can itself be separated into a memoryless nonlinear part and a linear filter. The nonlinear part is expressible as:

$$N_{K(3)}[y(t)] = y(t) - c \cdot y^2(t) + 2 \cdot c^2 \cdot y^3(t) \qquad (69)$$

This nonlinear part is followed by the linear filter K_1 to form **K**(3), the nonlinear equalizer, shown below in figure 25.

Figure 25; Third Order Inverse System, K(3)



The nonlinear part, $N_{k(3)}$, is recognizable as the power series reversal for the input *x(t)* in terms of the output *y(t)*. Continuing the expansion of the power series reversal beyond three terms, as found in [21], will give a "better" inverse of the nonlinear part of **G**. It should be pointed out that the chosen second order system **G** had a memoryless nonlinearity which enabled the algebraic simplification of K_2 and K_3. If this were not the case then the convolution operations shown in equations (64) and (66) would have been applied directly, resulting in a significant computational load.

XVII.C. Second Order System Linearization Results

To test the linearization effectiveness of the nonlinear equalizer a signal representing the modulated and downconverted carrier, known as the IF signal, is generated and passed through the second order system of figure 21. The power spectrum of this signal is taken at each stage in the simulated measurement system, with the nonlinearity and the nonlinear equalizer, to measure the linearization effectiveness.

Figure 26 below represents the power spectrum of the IF signal as could be found after

the final IF stage in the actual measurement system. The carrier has been downconverted to a 50Khz IF frequency. Notice that the modulation sidebands at 40Khz and 60Khz are symmetrical and no intermodulation terms are present that would indicate nonlinear distortion.

Figure 27 below represents the power spectrum of the IF signal after passing through the second order system of figure 21. The modulation sidebands are no longer symmetrical indicating severe linear distortion as discussed in section IX.b. There is also second order distortion.with intermodulation products at the frequencies predicted in section IX.c.

Figure 28 below represents the power spectrum of the IF signal after passing through the second order system of figure 21 and being equalized by the nonlinear part, equation (69), of the nonlinear equalizer. The intermodulation products show approximately a 30 db drop in power compared to figure 27. The linear distortion, however, is still present.

Figure 29 below represents the power spectrum of the IF signal after passing through the second order system of figure 21. This signal is then fully equalized by both the nonlinear part and the linear part of the nonlinear equalizer. The linear distortion has now been corrected as indicated by the symmetrical modulation sidebands. The equalizer's linear filter has a high pass magnitude characteristic due to the filter being the inverse of the antialiasing filter shown in figure 18. This high pass characteristic accentuates the high frequency intermodulation products that were previously attenuated by the equalizer's nonlinear part. However, the high frequency intermodulation products still show more than a 15 dB drop in power compared to the unequalized case of figure 27. The low frequency intermodulation products show a 35 dB drop in power compared to figure 27.

As an alternative way to examine the equalizer effectiveness the IF signal before and after equalization can be compared in the time domain. As a reference, the IF signal that passed

through the nonlinearity is subtracted from the original IF signal. This difference value will reveal any changes in amplitude and/or phase between the two signals. Figure 30 below shows this difference value for a segment of the signal buffers. The difference magnitude is approximately three units peak to peak demonstrating, in this case, a large phase difference between the two signals.

Figure 31 below shows the difference between the original IF signal and the IF signal exposed to the nonlinearity, after full equalization. The difference magnitude is approximately 0.035 units peak to peak indicating nearly two orders of magnitude improvement over the unequalized case of figure 30. The power series reversal was then expanded to four terms instead of three. The four term expansion reduced, by another factor of five, the difference between the original IF signal and the equalized IF signal after the nonlinearity.

Figure 26; Power Spectrum of the IF Signal

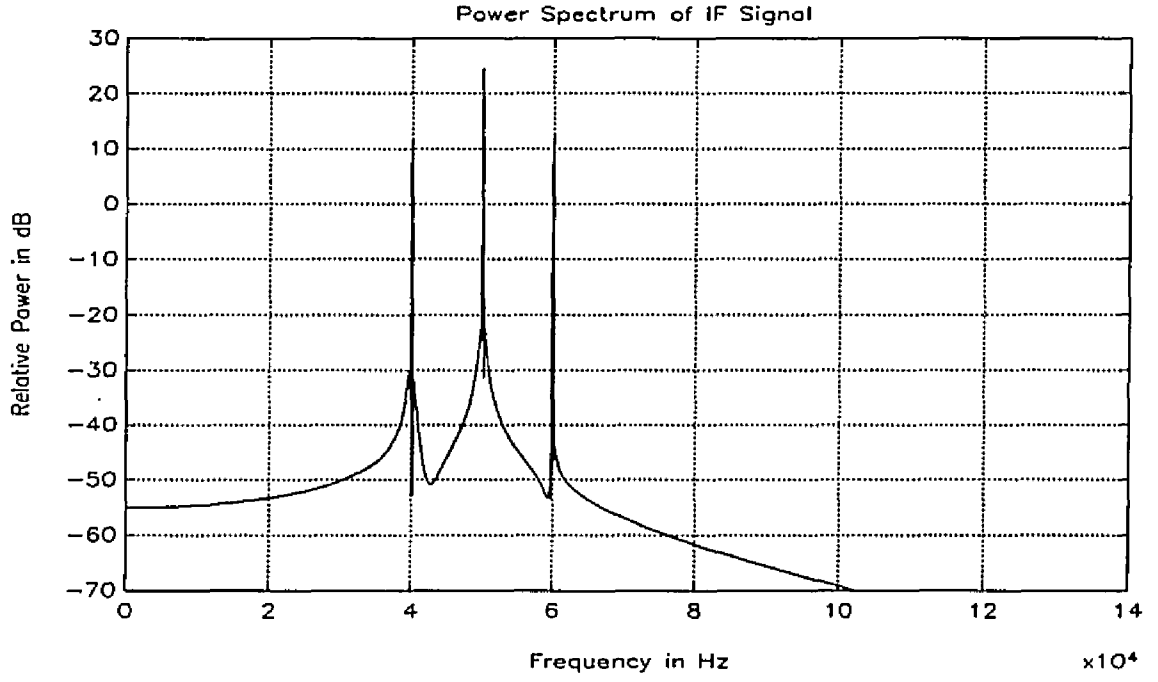


Figure 27; Power Spectrum of the IF Signal after the Second Order Nonlinearity

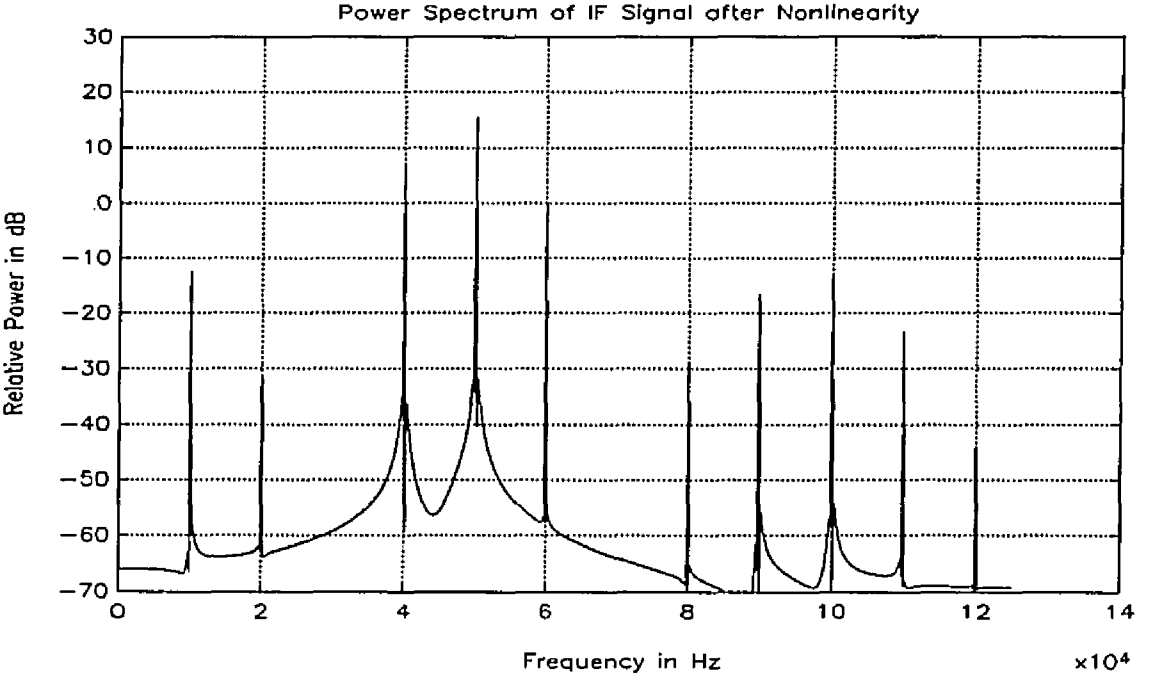


Figure 28; Power Spectrum of the IF Signal after the Nonlinear Part of the Equalization

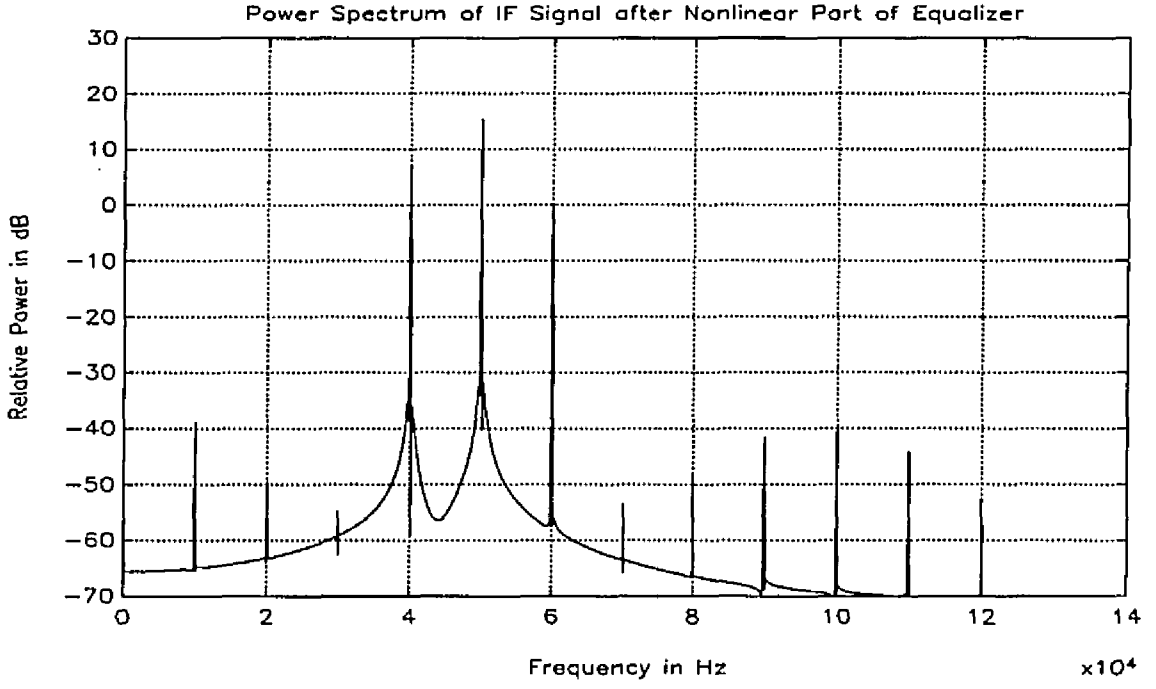


Figure 29; Power Spectrum of the IF Signal after Full Equalization

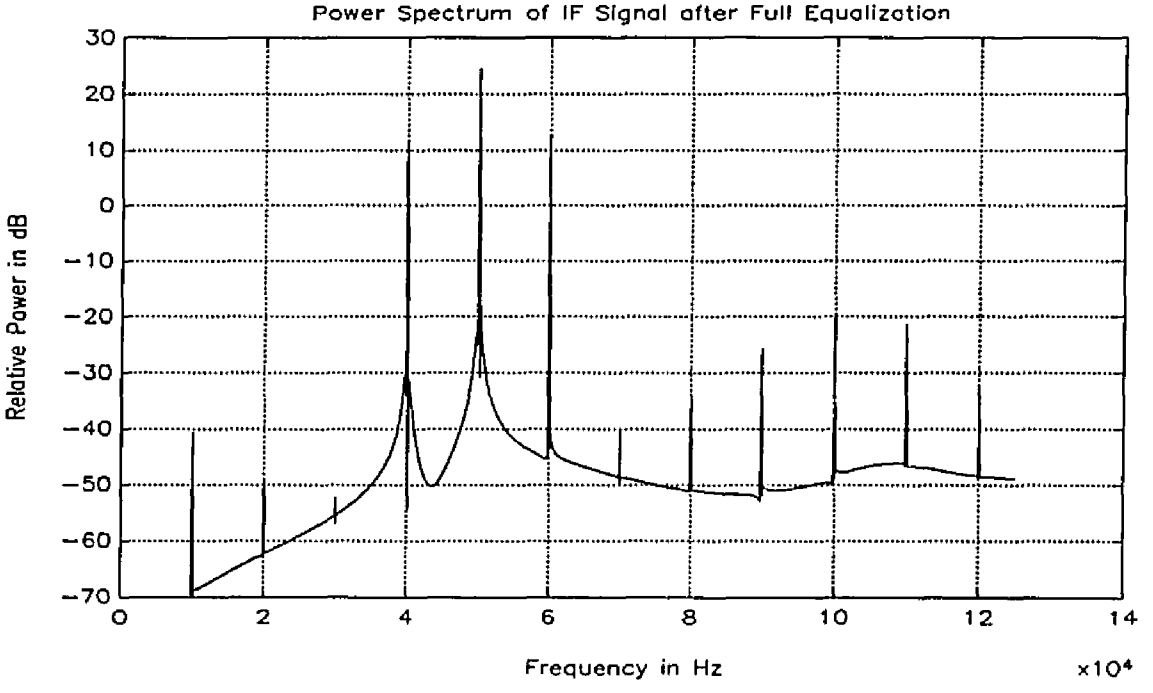


Figure 30; Difference Between the IF Signals: Before and After the Nonlinearity

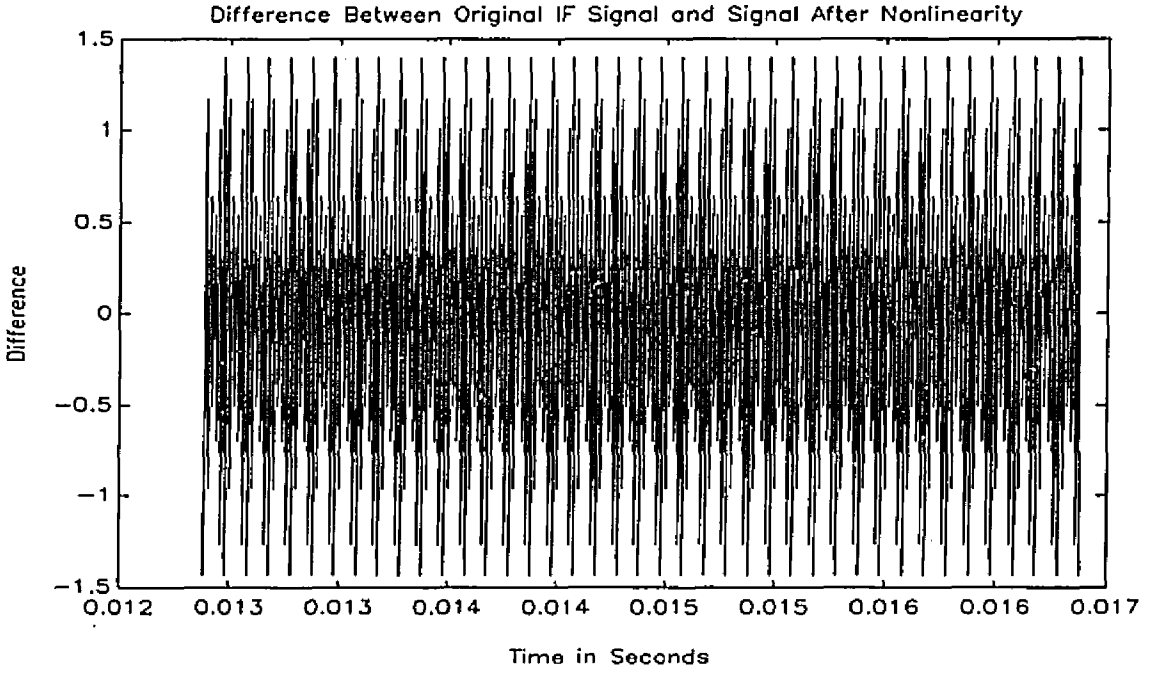
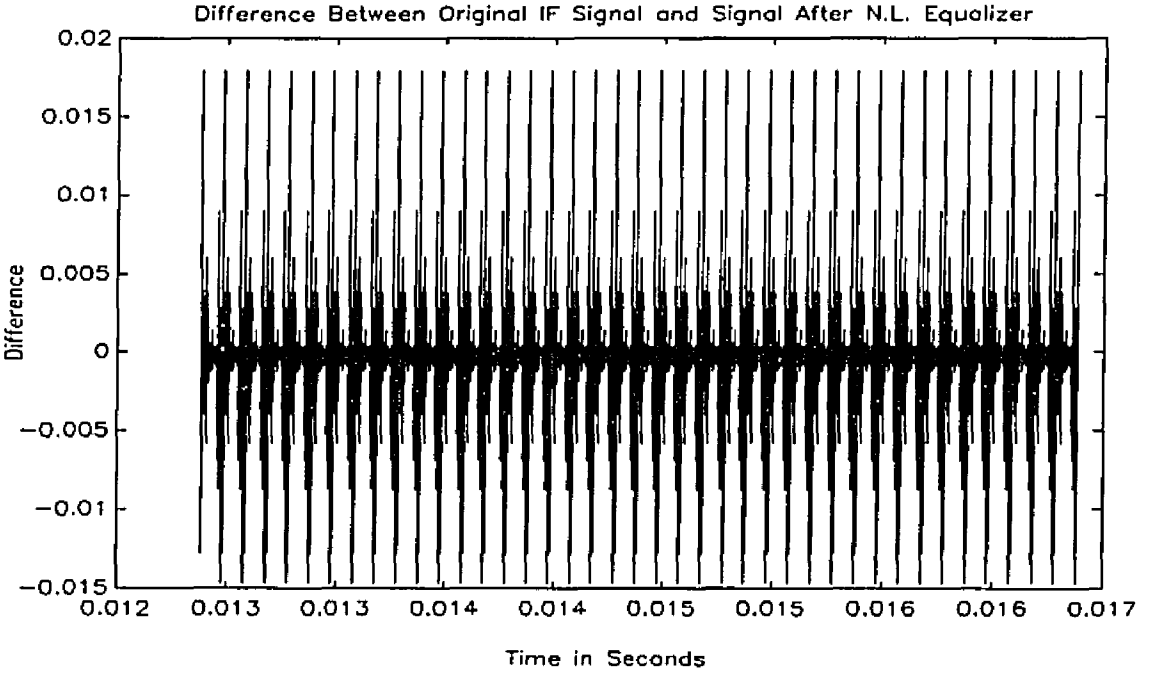


Figure 31; Difference Between the IF Signals: Before the Nonlinearity and After Equalization



XVIII. MONTE CARLO ANALYSIS OF THE PHASE ESTIMATION METHODS

It was mentioned in section VIII that the performance criterion by which to measure the quality of an estimator is by the estimate's mean and variance. The mean of the estimate should be equal to the actual quantity under estimation assuming the noise added is zero mean itself. The variance of the estimate should be a minimum so that the probability of obtaining an estimate close to the actual value is maximized. The simulation involving the random quantity of signal plus noise requires numerous runs to converge on the true values of the estimate's mean and variance. Such a simulation is referred to as a **Monte Carlo Analysis.**

To measure the quality of the estimator's mean a quantity, known as the relative error, is defined as the difference between the ideal value and the measured mean, divided by the ideal value. The relative percentage error then gives the error of the estimate's mean relative to the ideal or theoretical value as a percentage.

To measure the quality of the estimator's variance a quantity, known as the coefficient of variation, is defined as the sampled standard deviation of the estimate divided by the estimate's mean. The percentage coefficient of variation is then a normalized measure of the estimate's deviation, or randomness, in terms of a percentage.

When examining the performance of a phase estimation method both the relative error and the coefficient of variation at the SNR of interest need to be cross examined. A small coefficient of variation is useless if the relative error is large and vice versa.

The performance of the phase estimation methods under several nonideal conditions were simulated and graphs were assembled in figures 31 through 47 below. The graphs use fifty points of data for each of the fourteen carrier frequencies evaluated. The DUT is the linear fifth order

lowpass butterworth filter from part I of the thesis. The nonideal conditions include linear distortion from the DUT, additive broadband Gaussian noise and additive sinusoidal interference. The effectiveness of the equalizer developed in section XVII for canceling second order distortion, was also simulated and the results shown in figures 48 through 51 below.

In all the graphs in the Monte Carlo analysis the Hilbert method is marked by a "+", the correlation receiver method by a "*" and the system identification method by a "o". In some cases the marks are missing in which case they are actually off scale. In some other cases the "+","*" and/or "o" marks are superimposed and require scrutinization.

The first set of graphs, shown in figures 32 through 37, use 10Khz AM and PM modulation. Figures 32 and 33 give the relative group delay error in a noiseless environment for the AM and PM phase estimation methods respectively. The deviation from ideal is due to the linear distortion of the DUT inducing AM to PM conversion. Figures 34 and 35 give the relative group delay error in an additive broadband noise environment for the AM and PM phase estimation methods respectively. The SNR varies from -32 dB to +34 dB in increments of 6 dB. This SNR variation is consistent for all graphs in the Monte Carlo analysis involving noise. Figures 36 and 37 give the percentage coefficient of variation for the group delay estimate with additive broadband noise for the AM and PM phase estimation methods respectively.

The second set of graphs, shown in figures 38 through 43, use 1Khz AM and PM modulation. The 1Khz modulation rate, with its commensurate reduction in linear distortion and aperture error, allows the effects of the linear distortion and aperture error to be compared against the 10Khz modulation case of figures 32 through 37.

The third set of graphs, shown in figures 44 through 47, use 10Khz AM and PM modulation. A 25Khz sinusoid was chosen as the interference frequency since its period is not

a integral multiple of the modulation rate and therefore will not be integrated completely by the correlation receiver method nor averaged completely by the Hilbert transform method. Figures 44 and 45 give the relative group delay error in an additive 25Khz sinusoidal interference environment for the AM and PM phase estimation methods respectively. Figures 46 and 47 give the percentage coefficient of variation for the group delay estimate with additive 25Khz sinusoidal interference for the AM and PM phase estimation methods respectively. The SNR, as in the broadband noise case, varies from -32 dB to +34 dB in increments of 6 dB.

The last set of graphs, shown in figures 48 through 51, use 10Khz AM and PM modulation with the linear butterworth filter DUT and the second order network of figure 21. This second order network is used on both the reference channel and the DUT channel to simulate a channel's antialiasing filter with a nonlinear buffer amplifier as discussed in section XV.a.

The graphs of figures 48 and 49 show the second order distortion effect, without equalization, on the AM and PM phase estimation methods respectively. Figures 48 and 49 give the relative group delay error in a noise free environment. The graphs of figures 50 and 51 show the second order distortion effects after being canceled by the nonlinear equalizer for the AM and PM phase estimation methods respectively. The distortion effects have been effectively canceled in figures 50 and 51 as is shown by comparing with figures 32 and 33 respectively. The nonlinear equalizer used was described in section XVII.

The MATLAB program which collected the data, as generated by the phase estimation methods, and created the graphs shown in figures 32 through 37 for the 10Khz Monte Carlo analysis is located in Appendix III. The programs for the other cases are slightly modified versions of this program.

As an indication of the total computation time needed for the Monte Carlo analysis the times are listed below which represents the number of minutes to process 14 frequency points on a 33Mhz 386 microcomputer running MATLAB 386. The computational effort of the various phase estimation methods can be ordered, from greatest to least, as follows: AM/PM system identification = 80 min. for 250 taps, PM Hilbert = 56 min., AM Hilbert = 16 min., AM correlation receiver = 6.5 min., PM correlation receiver = 6 min.

Figure 32; Relative Group Delay Error, Noiseless Environment, 10Khz AM Modulation

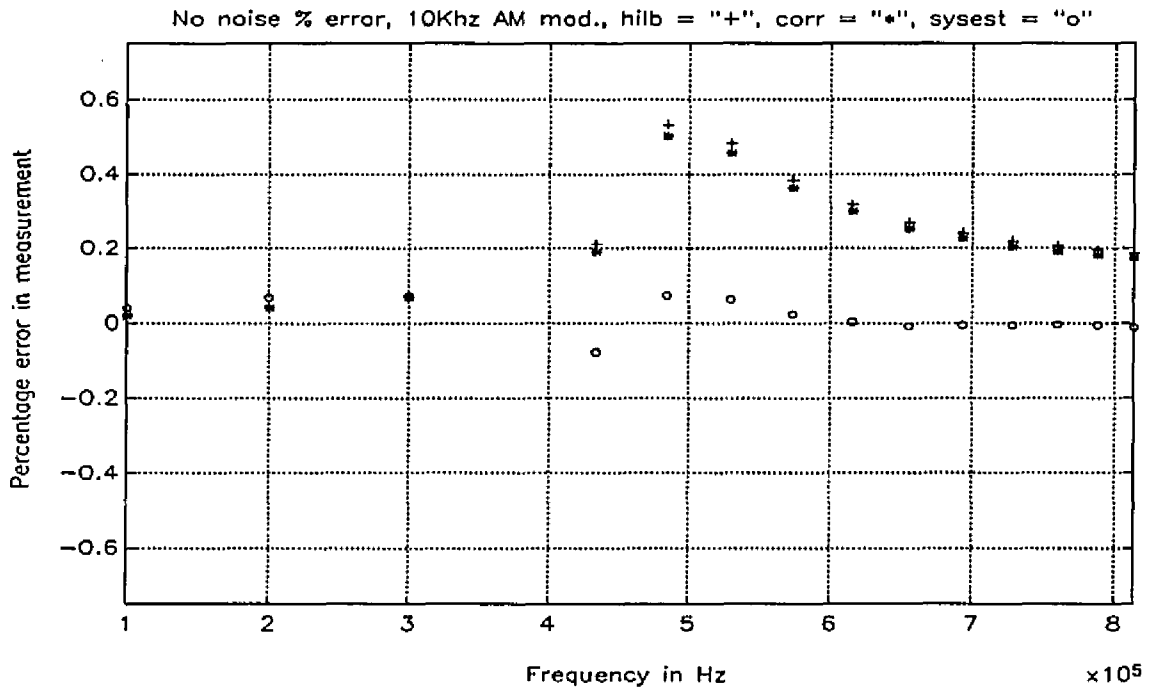


Figure 33; Relative Group Delay Error, Noiseless Environment, 10Khz PM Modulation

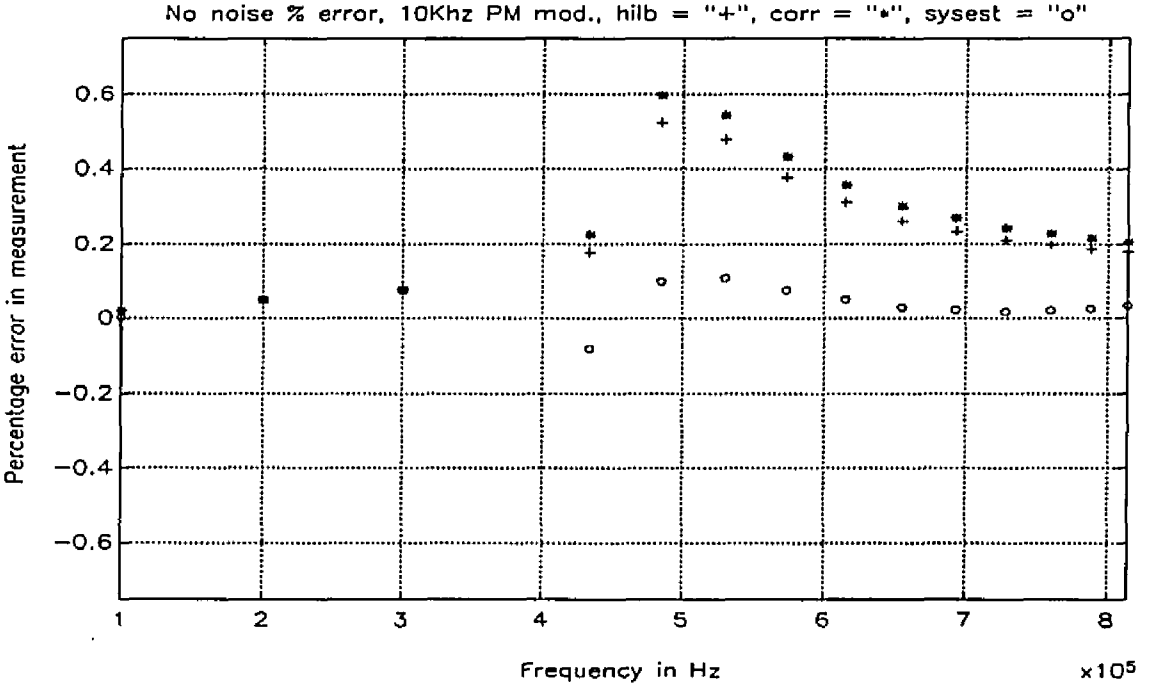


Figure 34; Relative Group Delay Error, Additive Noise, 10Khz AM Modulation

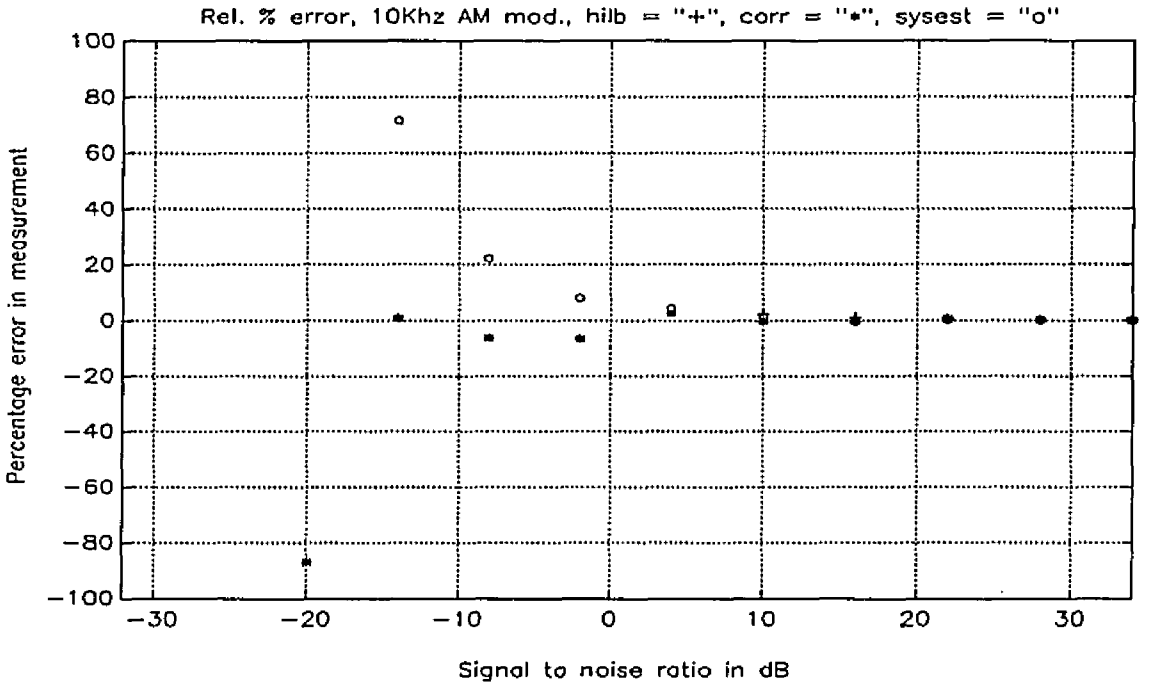


Figure 35; Relative Group Delay Error, Additive Noise, 10Khz PM Modulation

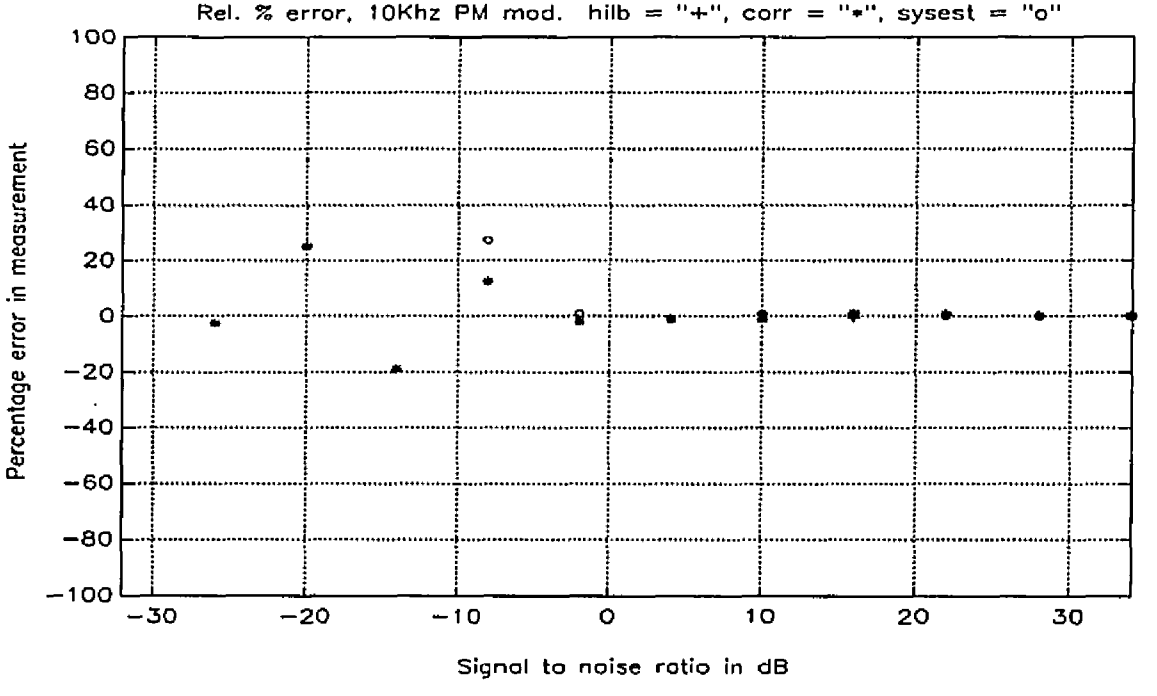


Figure 36; Coefficient of Variation, Additive Noise, 10Khz AM Modulation

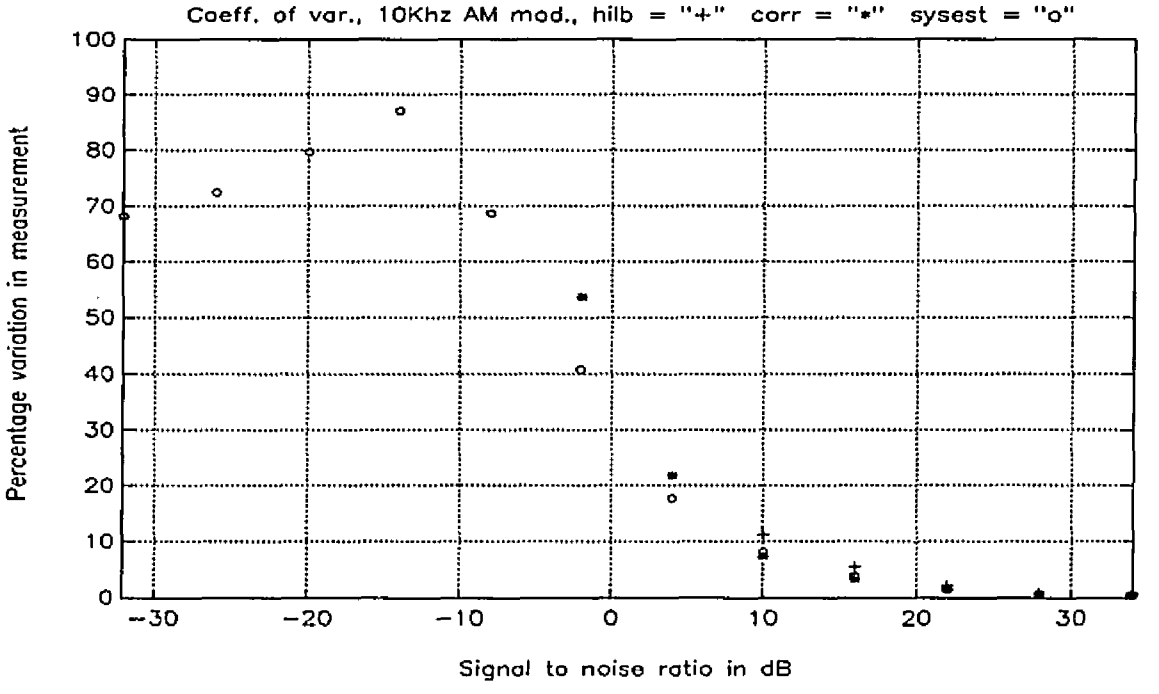


Figure 37; Coefficient of Variation, Additive Noise, 10Khz PM Modulation

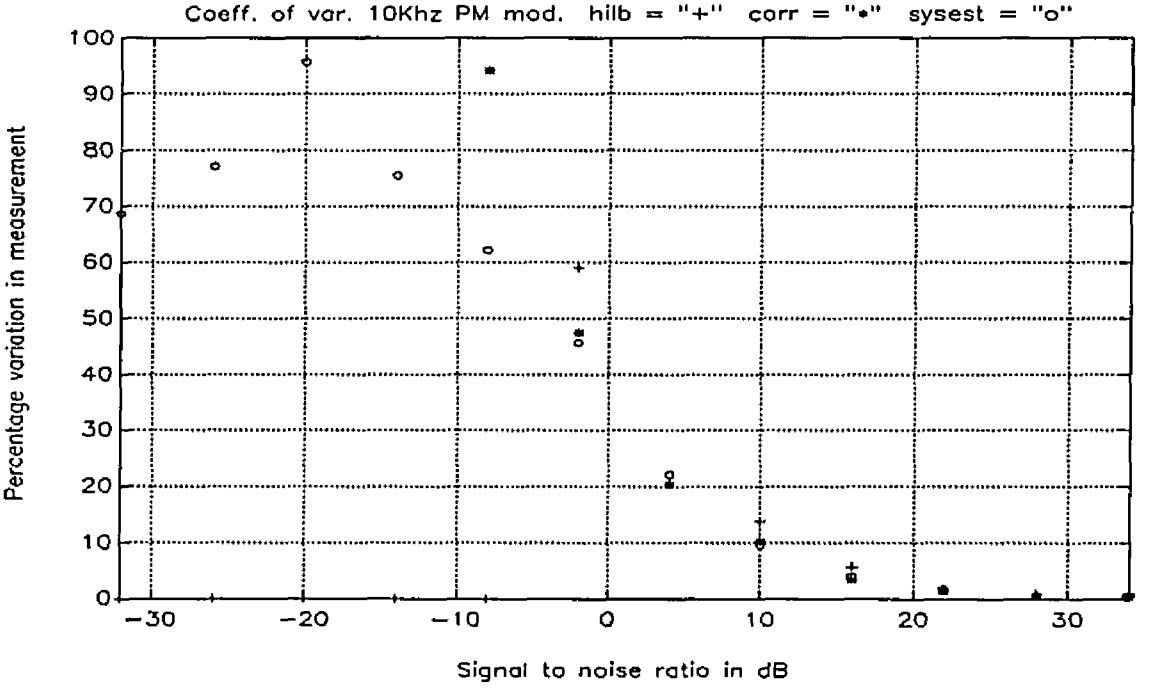


Figure 38; Relative Group Delay Error, Noiseless Environment, 1Khz AM Modulation

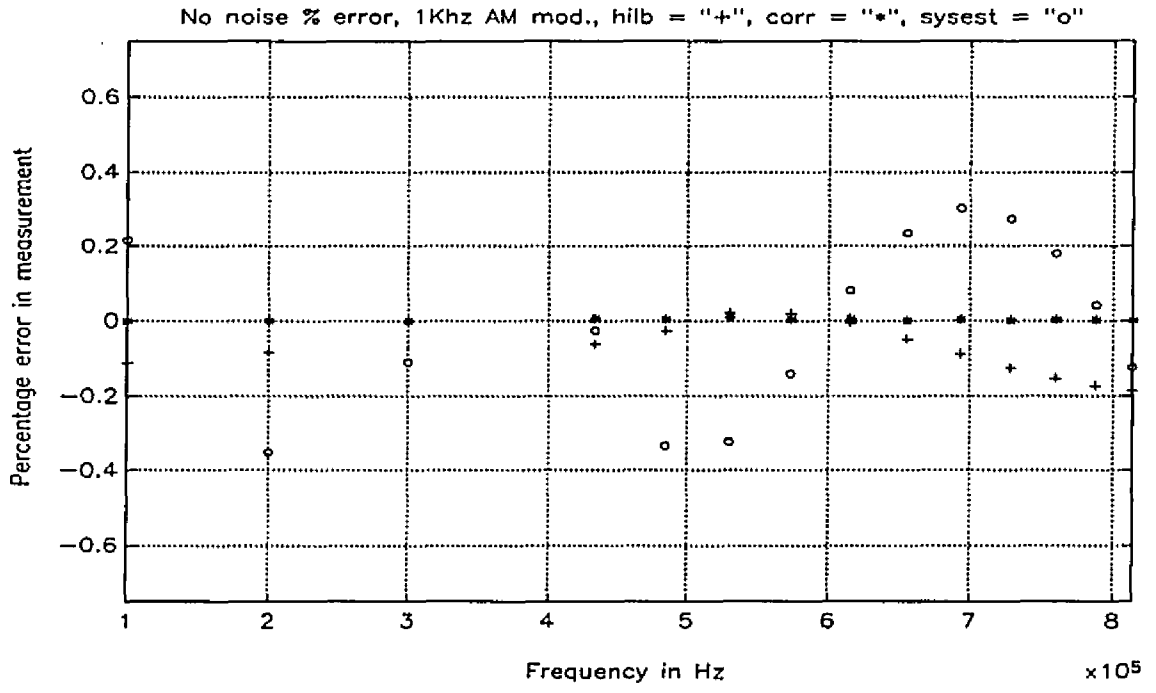


Figure 39; Relative Group Delay Error, Noiseless Environment, 1Khz PM Modulation

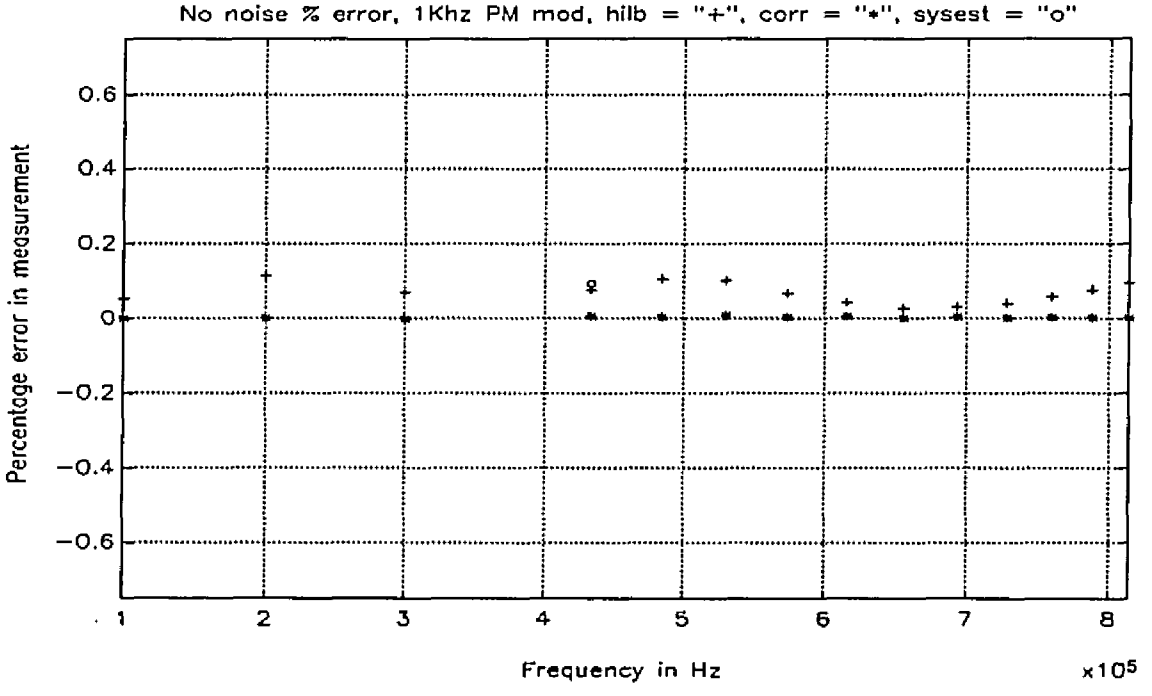


Figure 40; Relative Group Delay Error, Additive Noise, 1Khz AM Modulation

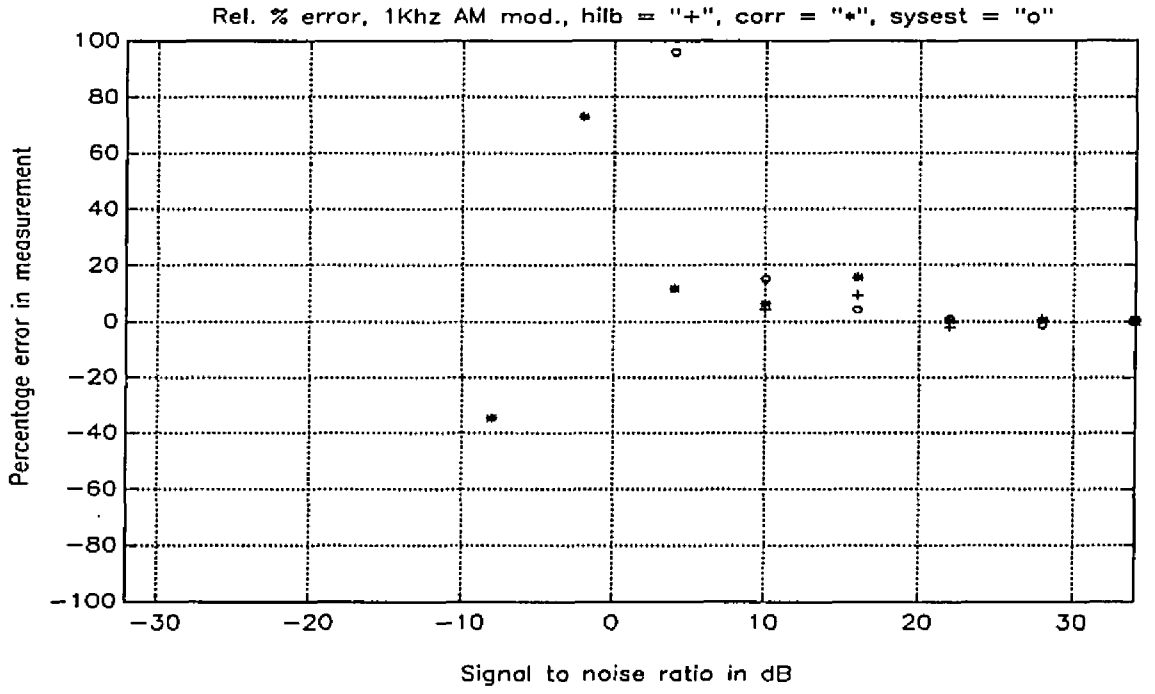


Figure 41; Relative Group Delay Error, Additive Noise, 1Khz PM Modulation

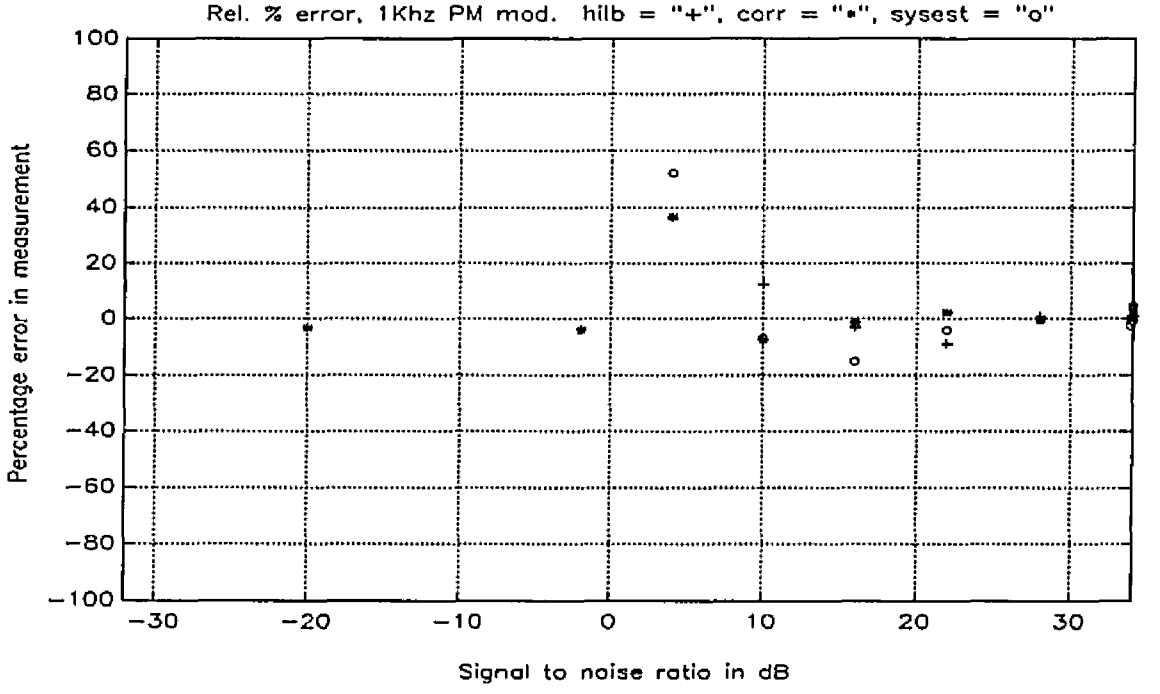


Figure 42; Coefficient of Variation, Additive Noise, 1Khz AM Modulation

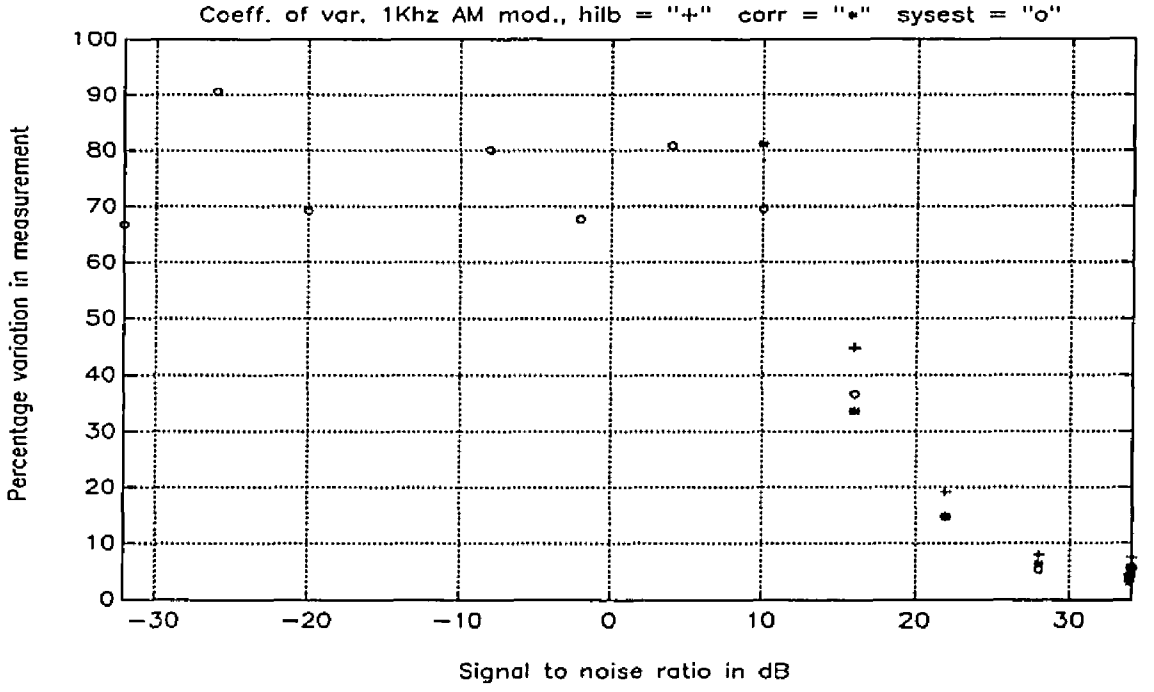


Figure 43; Coefficient of Variation, Additive Noise, 1Khz PM Modulation

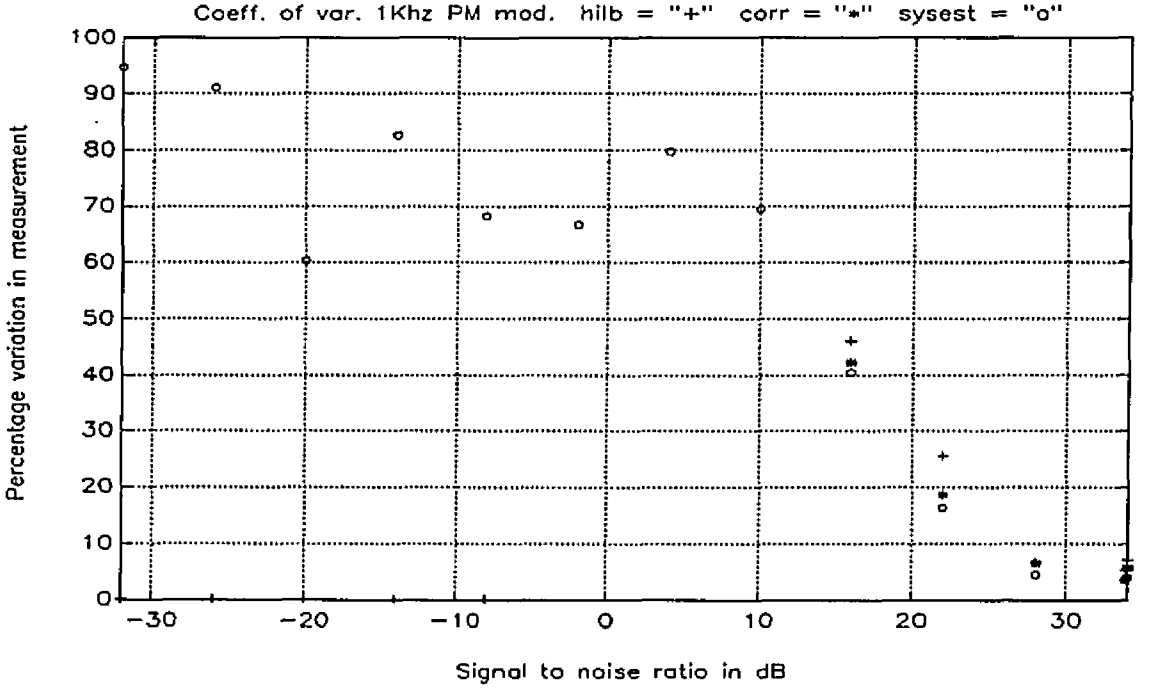


Figure 44; Relative Group Delay Error, 25Khz Sinusoidal Inter., 10Khz AM Modulation

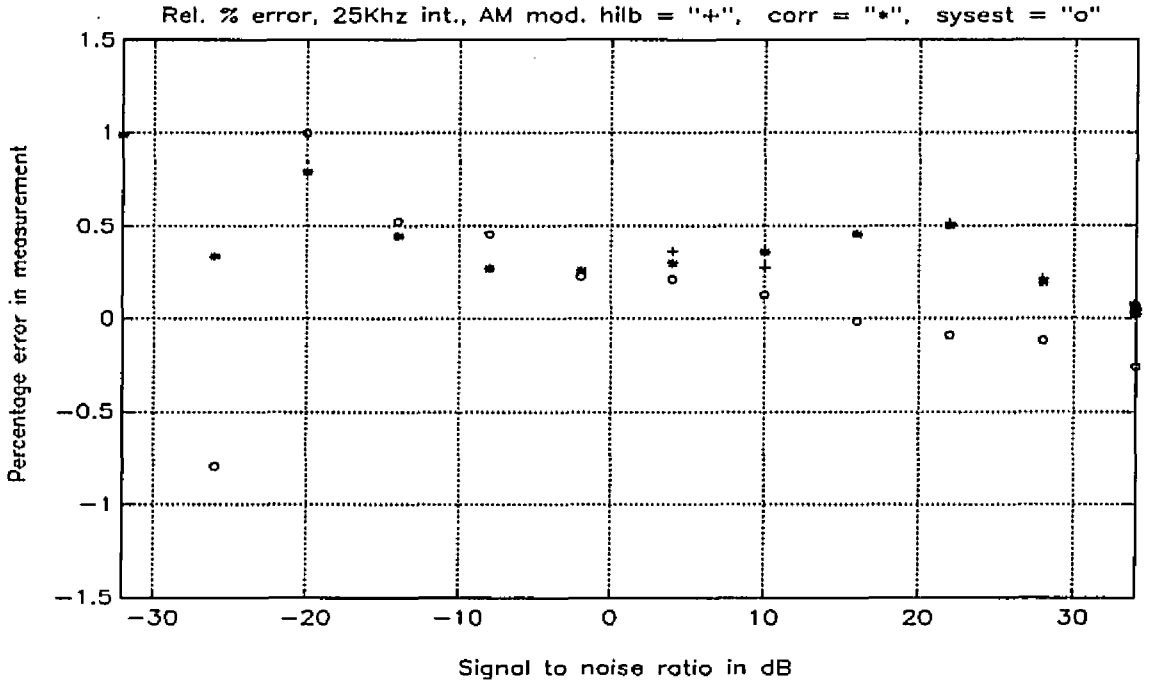


Figure 45; Relative Group Delay Error, 25Khz Sinusoidal Inter., 10KHz PM Modulation

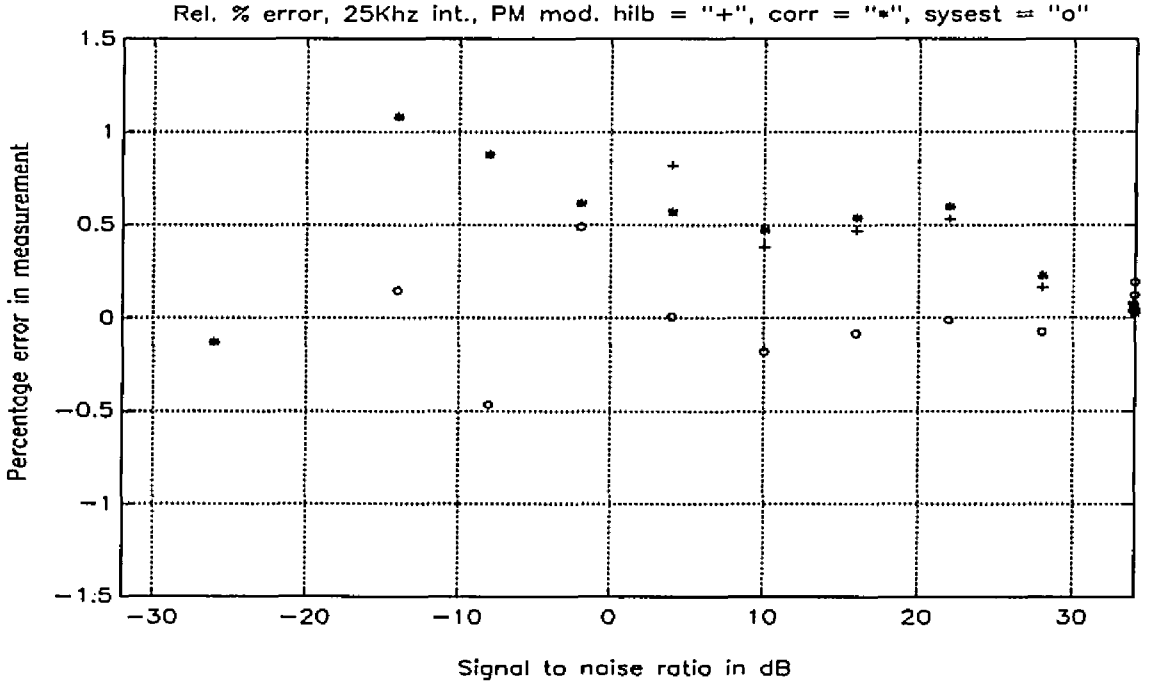


Figure 46; Coefficient of Variation, 25Khz Sinusoidal Inter., 10Khz AM Modulation

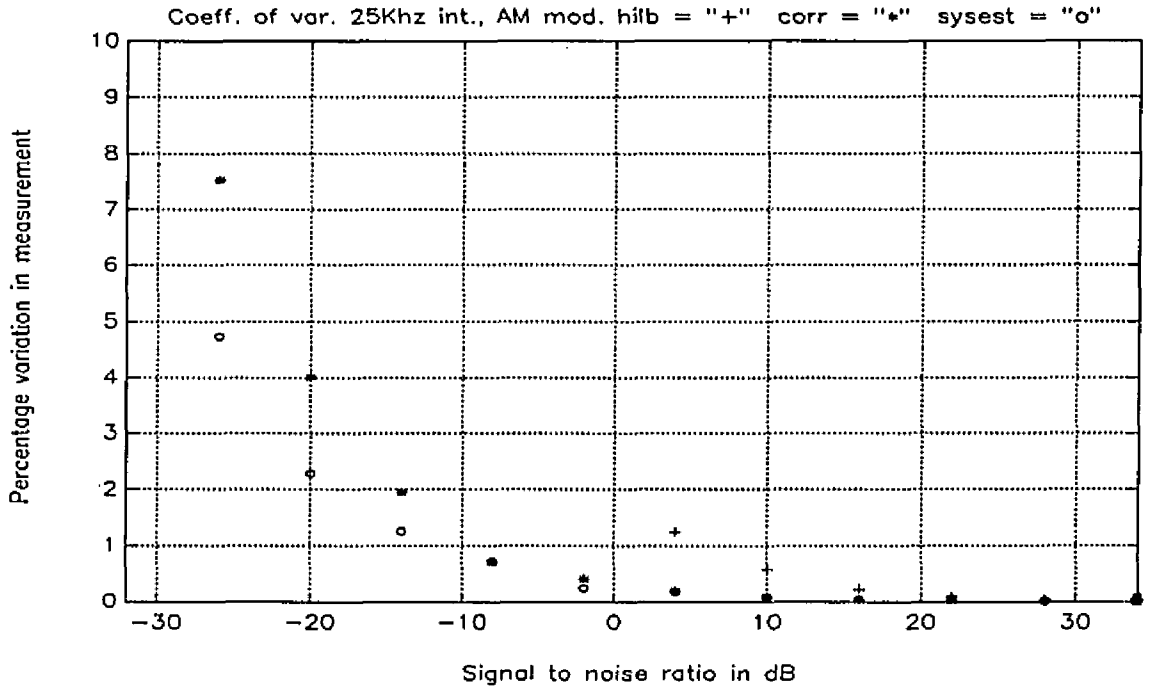


Figure 47; Coefficient of Variation, 25Khz Sinusoidal Inter., 10Khz PM Modulation

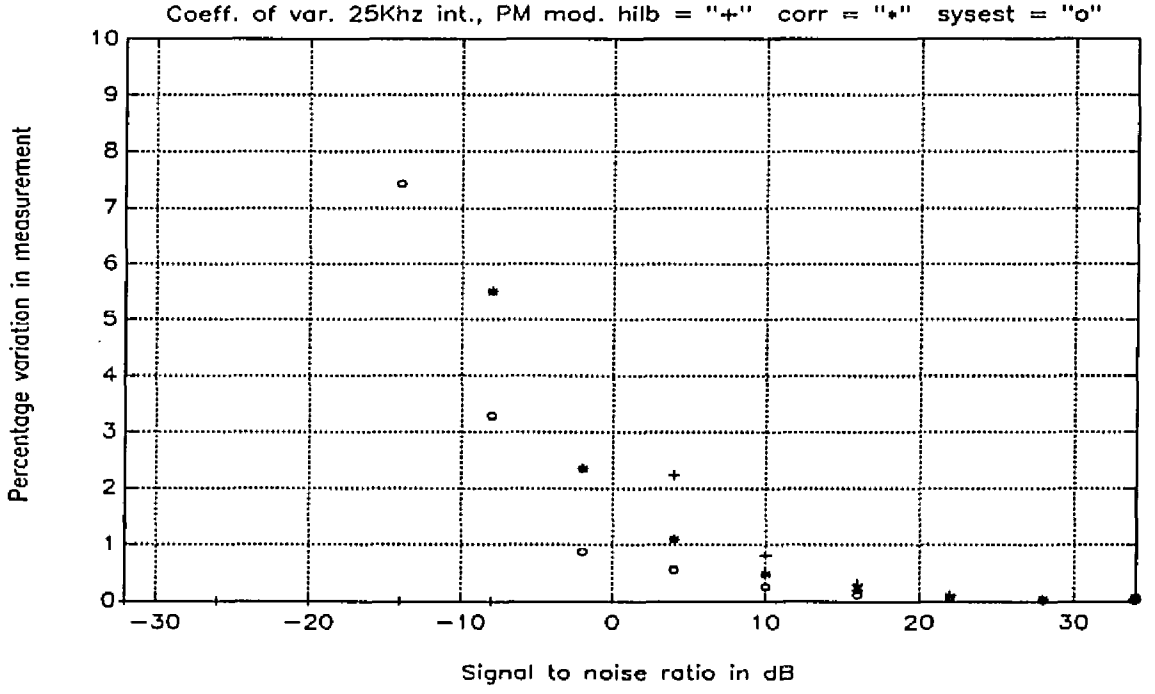


Figure 48; Relative Group Delay Error, Second Order Distortion, 10Khz AM Modulation

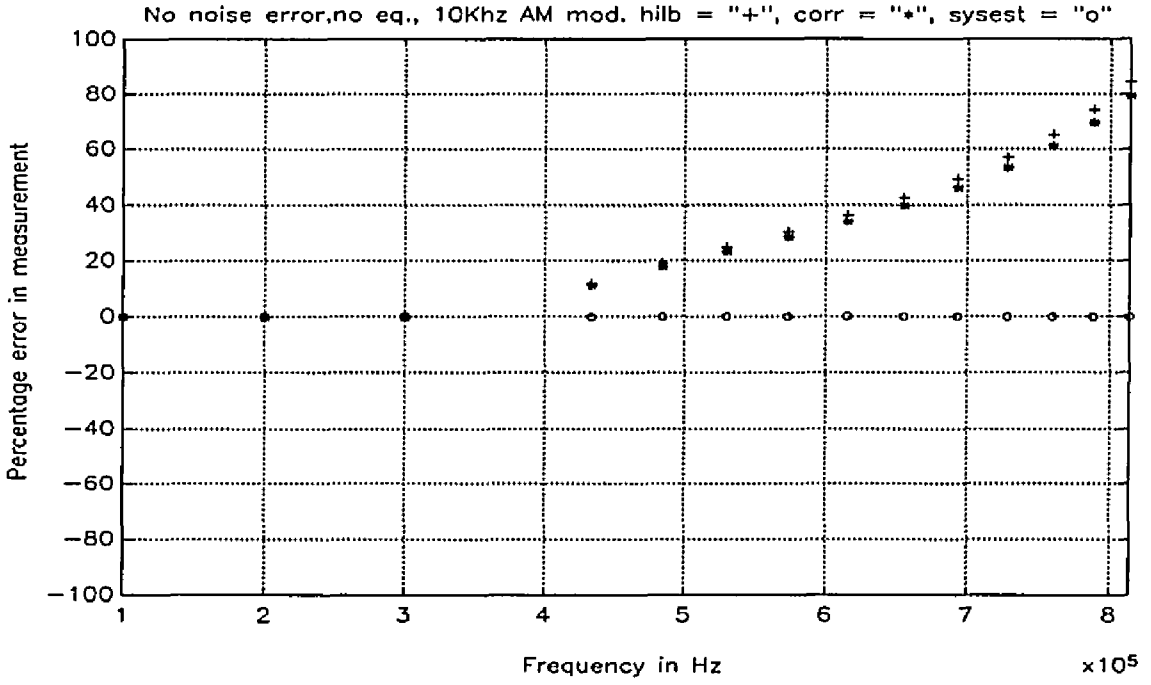


Figure 49; Relative Group Delay Error, Second Order Distortion, 10Khz PM Modulation

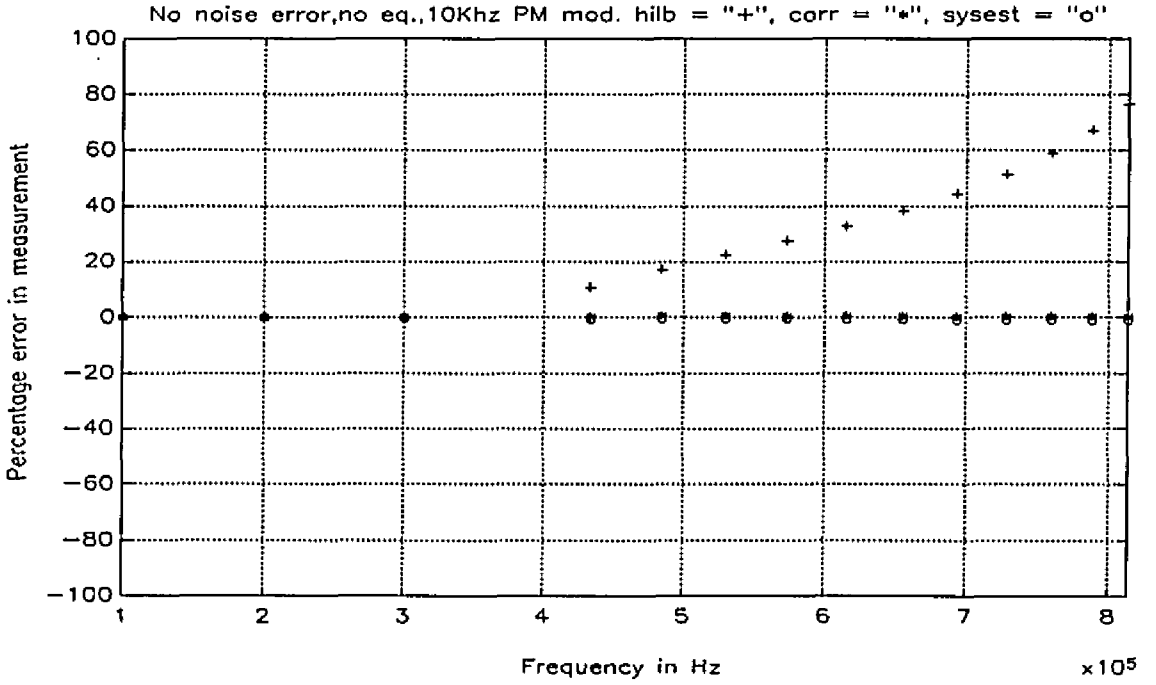


Figure 50; Relative Group Delay Error, Distortion with Equalizer, 10Khz AM Modulation

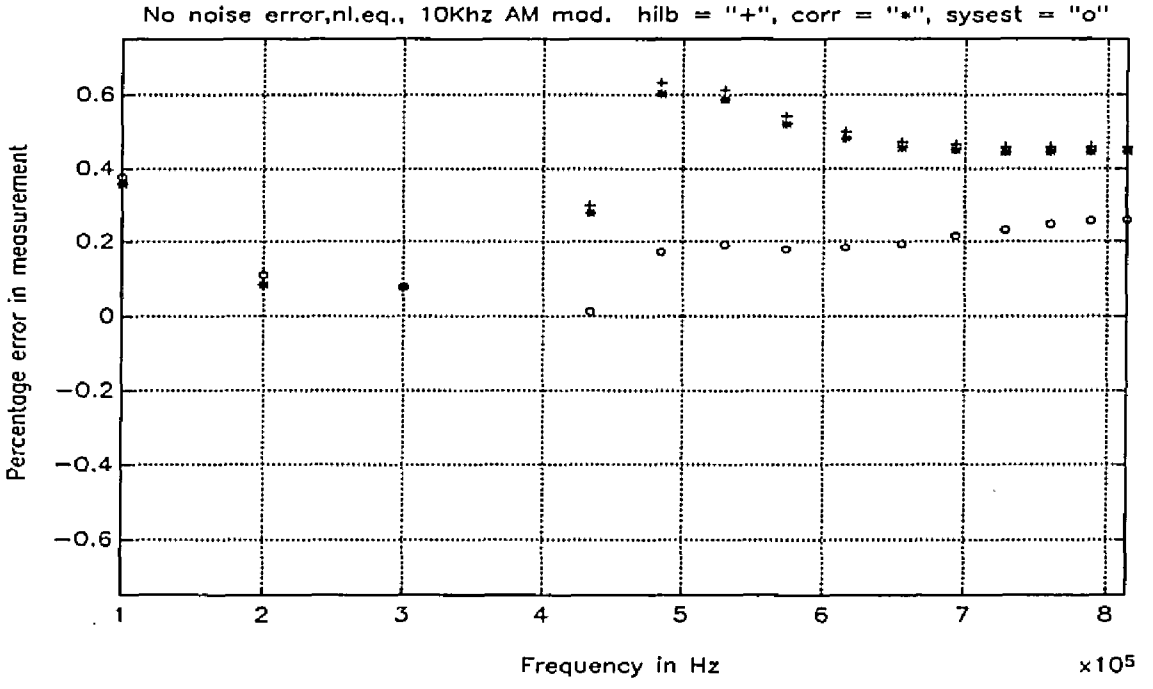
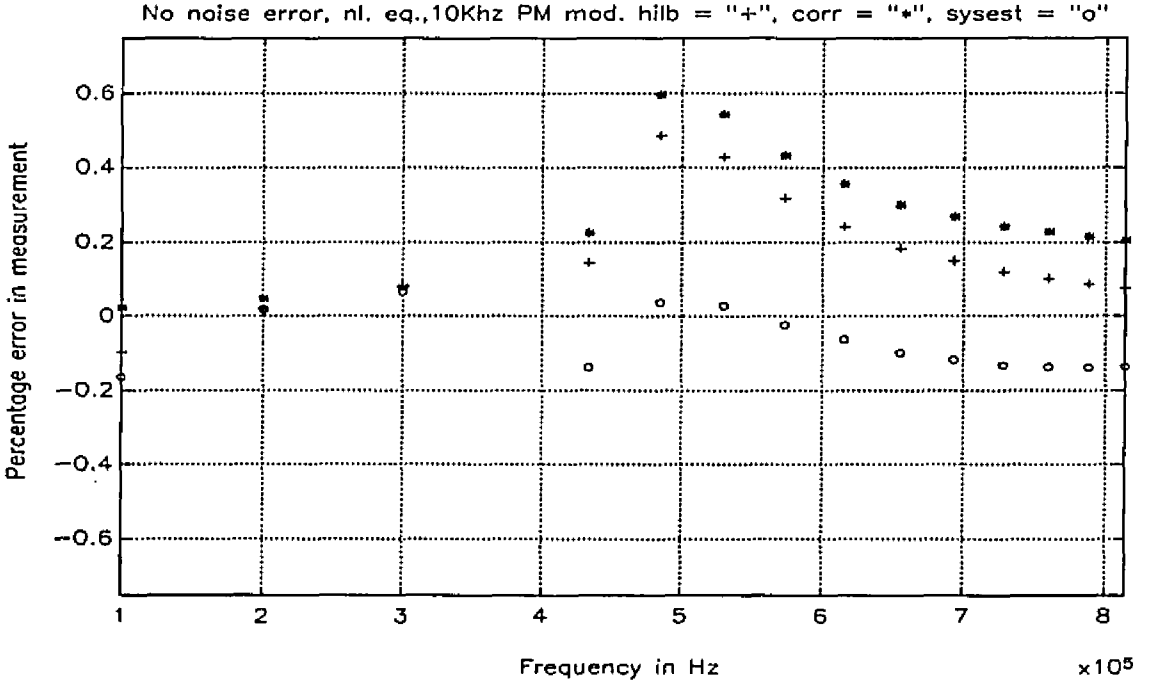


Figure 51; Relative Group Delay Error, Distortion with Equalizer, 10Kkz PM Modulation



XIX. CONCLUSIONS

The Monte Carlo analysis performed in section XVII for the different phase detection methods revealed their strengths and weaknesses under different conditions. These conditions included modulation rate, additive noise, sinusoidal interference and second order distortion.

As shown in section XVII, the choice of the modulation rate for the highest accuracy group delay estimate depends on the linear distortion produced by the DUT, the aperture error and the SNR present. The lowest modulation rate will induce the least linear distortion and aperture error but will have the smallest modulation envelope phase shift through the DUT, thereby being more susceptible to noise. The system identification method requires more filter taps to resolve the sidebands at the lower modulation rate thus increasing this method's computational load. This is demonstrated by comparing the system's identification performance for the 10Khz and 1Khz cases where the number of filter taps remained constant at 250.

In a noiseless environment without distortion effects the correlation receiver method gave the best phase estimate. The system identification method requires a large number of filter taps to achieve good phase resolution. The Hilbert transform method suffers from the truncation effects, i.e. ripples, of a finite data buffer size being used with the quadrature filter.

Of the three phase estimation methods the correlation receiver method and the system identification method show more additive noise immunity than the Hilbert transform method. The accuracy advantage that the correlation method has in the noiseless and distortionless environment disappears in the nonideal environment. The correlation receiver method and system identification method show quite similar SNR performance with additive, broadband noise.

The correlation receiver method and the system identification method show significantly

better 25Khz sinusoidal interference immunity than the Hilbert transform method. This is explained by the fact that the correlation receiver method integrates the interfering sinusoid over the modulation period. The system identification method's filter adaptively creates a transmission notch at the interfering frequency of 25Khz giving good results even below a SNR of -20 dB.

The AM and PM system identification methods along with the PM correlation receiver method were shown to be the least susceptible to second order distortion created by the network of figure 21. The remaining methods required an equalizer to bring the group delay measurement accuracy back to pre-distortion levels.

Since the DUT used in the simulation of the different phase estimation methods is linear it does not have an AM compression characteristic that would have shown the advantage of PM modulation over AM modulation for this type of (nonlinear) DUT.

Any conclusions as to the "best" phase detection method need to be considered in light of the fact that the simulation of this thesis covered only a small, albeit important, number of the possible deleterious conditions existing in the actual measurement system. Conditions that were beyond the scope of this thesis include: noise added to the reference channel, phase noise, quantization effects and higher order nonlinearities, to name several.

Also a consideration is the computational requirements of a given method. The computation time for the phase estimation methods was listed in section XVIII. The corrective equalization that may or may not be needed also requires consideration. Unless the nonlinear system to be canceled has only a short "memory" the requirement to process the data within seconds, and the multidimensional discrete convolutions needed, will far exceed the capabilities of today's minicomputers.

APPENDIX I.A. AM AND PM MODULATION SIGNAL GENERATION

AMGEN.M

```
clear;
%
am_mod_index = input('Enter am modulation index, =? ');
am_mod_freq = input('Enter am modulation frequency, =? ');
%
n_max = 14;              % set number of frequency points
%
load c:\matlab\data\freqlist
%
if_freq = 50E3;
%
buffer_length = 32768 + 1600;   % add extra points to eliminate filter transients
sample_period = 0.5E-6; % sample at 2 MHZ
end_time = (buffer_length - 1) * sample_period;
t = 0:sample_period:end_time;
%
load buttfilt;                  % load DUT filter
%
%*************************************************************************
for mod_nomod_loop =1:2         % carrier only loop / carrier and modulation loop
   for n = 1:n_max
   %
      carrier = freq_list(n)          % load carrier from frequency list
      lo_freq = carrier - if_freq;
      %
      if mod_nomod_loop == 1,
         ref_signal = cos(2*pi*carrier*t);
      else
         ref_signal =(1 + am_mod_index*cos(2*pi*am_mod_freq*t)).* cos(2*pi*carrier*t);
      end
      down_conv_ref_sig = ref_signal .* cos(2*pi*lo_freq*t);
      ref_sig_ext = decimate(down_conv_ref_sig,8);
      ref_sig = ref_sig_ext(101:4096 + 100);
      %
      dut_signal = filter(b2,a2,ref_signal);
      down_conv_dut_sig = dut_signal .* cos(2*pi*lo_freq*t);
      dut_sig_ext = decimate(down_conv_dut_sig,8);
      dut_sig = dut_sig_ext(101:4096 + 100);
      %
      if mod_nomod_loop == 1,
         eval(['save ','c:\matlab\data\10khzlin\amnom',int2str(n),' ref_sig dut_sig']);
      else
```

```
        eval(['save ','c:\matlab\data\10khzlin\amm',int2str(n),' ref_sig dut_sig']);
      end
   end                    % next "n" loop
end                       % next mod,no mod loop
*
clear;
```

PMGEN.M

```
clear;
%
pm_mod_index = input('Enter pm modulation index, =? ');
pm_mod_freq = input('Enter pm modulation frequency, =? ');
%
n_max = 14;                          % set number of frequency points
%
load c:\matlab\data\freqlist;
%
if_freq = 50E3;
%
buffer_length = 32768 + 1600;        % add points to eliminate filter transients
sample_period = 0.5E-6;              % sample at 2 MHZ
end_time = (buffer_length - 1) * sample_period;
t = 0:sample_period:end_time;
%
load buttfilt;                       % load filter DUT
%
%*************************************************************************
for mod_nomod_loop =1:2      % carrier only loop / carrier plus modulation loop
   for n = 1:n_max
      %
      carrier = freq_list(n)
      %
      lo_freq = carrier - if_freq;
      %
      if mod_nomod_loop == 1,
         ref_signal = cos(2*pi*carrier*t);
      else
         ref_signal =  cos(2*pi*carrier*t + pm_mod_index*sin(2*pi*pm_mod_freq*t));
      end
      down_conv_ref_sig = ref_signal .* cos(2*pi*lo_freq*t);
      ref_sig_ext = decimate(down_conv_ref_sig,8);
      ref_sig = ref_sig_ext(101:4096 + 100);
      %
      dut_signal = filter(b2,a2,ref_signal);
      down_conv_dut_sig = dut_signal .* cos(2*pi*lo_freq*t);
      dut_sig_ext = decimate(down_conv_dut_sig,8);
      dut_sig = dut_sig_ext(101:4096 + 100);
      %
      if mod_nomod_loop == 1,
         eval(['save ','c:\matlab\data\10khzlin\pmnom',int2str(n),' ref_sig dut_sig']);
      else
         eval(['save ','c:\matlab\data\10khzlin\pmm',int2str(n),' ref_sig dut_sig']);
```

```
      end
   end                  % next "n" loop
end                     % next mod,no mod loop
*
clear;
```

APPENDIX I.B. AM HILBERT TRANSFORM METHOD

AMHILB.M

```
clear;
rand('normal');          % set for normal distribution
%
file_offset = input(' Enter data storage file offset =? ')
am_mod_freq = 10E3;
noise_scale = 0.003026;
n_max = 14;              % set for 14 frequency points
end_collect = 50;        % collect fifty of each
%
if_freq = 50E3;
%
new_buffer_length = 4096;
new_sample_period = 4E-6;          % sample at 250 KHz
new_end_time = (new_buffer_length - 1) * new_sample_period;
new_t = 0:new_sample_period:new_end_time;
mod_samples = fix(1/(am_mod_freq * new_sample_period));
%
%
% find number of modulation cycles for averaging
cycles = fix((new_buffer_length - 2000)/mod_samples);
l_indx = fix(new_buffer_length/2 - (mod_samples/2)*cycles)  % upper usable buffer limit
h_indx = round(mod_samples*cycles + l_indx - 1)             % lower limit
%
load nsfilt;                       % load noise bandwidth limiting filter
load c:\matlab\data\freqlist;
%
%*****************************************************************************
for data_iter = 1:end_collect      % collect data until 50 points gathered
   %
   for n = 1:n_max
   %
      carrier = freq_list(n)
      %
      freq_array(1,n) = carrier;
      %
      %
      time_vec = clock
      new_seed = time_vec(6) + 60*time_vec(5) + 3600*time_vec(4) + 86400*time_vec(3)

      rand('seed',new_seed);
      eval(['load ','c:\matlab\data\10khzlin\amm',int2str(n)])
      bl_noise = filter(b3,a3,rand(1,new_buffer_length));
```

```
        dut_sig = dut_sig +  noise_scale*bl_noise;
        %
        ref_analytic = hilbert(ref_sig);        % see flowchart in writeup
        ref_mag = abs(ref_analytic);
        ref_mag_phase = angle(hilbert(ref_mag - mean(ref_mag(l_indx:h_indx))));
        ref_mag_phase_dev = unwrap(ref_mag_phase) - 2*pi*am_mod_freq*new_t;
        average_ref_phase = mean(ref_mag_phase_dev(l_indx:h_indx))
        %
        dut_analytic = hilbert(dut_sig);
        dut_mag = abs(dut_analytic);
        dut_mag_phase = angle(hilbert(dut_mag - mean(dut_mag(l_indx:h_indx))));
        dut_mag_phase_dev = unwrap(dut_mag_phase) - 2*pi*am_mod_freq*new_t;
        average_dut_phase = mean(dut_mag_phase_dev(l_indx:h_indx))
        %
        envelope_phase_diff = average_ref_phase - average_dut_phase;
        group_delay(1,n) = envelope_phase_diff / (2*pi*am_mod_freq)
    end         % last n loop
    %
    eval(['save ','c:\matlab\data\10khzlin\amhil',int2str(file_offset + data_iter),'
    group_delay']);
end             % end data collection
%
clear;
```

APPENDIX I.C. PM HILBERT TRANSFORM METHOD

PMHILB.M

```
clear;
rand('normal');
%
%
file_offset = input(' Enter data storage file offset =? ')
pm_mod_freq = 1E3;
noise_scale = 0.003026;
n_max = 14;            % set number of frequency points
end_collect = 50;      % collect fifty of each
if_freq = 50E3;
%
new_buffer_length = 4096;
new_sample_period = 4E-6;       % sample at 250 KHz
new_end_time = (new_buffer_length - 1) * new_sample_period;
new_t = 0:new_sample_period:new_end_time;
%
% find number of modulation cycles for averaging purposes
mod_samples = fix(1/(pm_mod_freq * new_sample_period));
cycles = fix((new_buffer_length - 2000)/mod_samples);
l_indx = fix(new_buffer_length/2 - (mod_samples/2)*cycles);
h_indx = round(mod_samples*cycles + l_indx - 1);
%
load nsfilt;
load c:\matlab\data\freqlist;
%
for data_iter = 1:end_collect
   for n = 1:n_max
      carrier = freq_list(n);
      freq_array(1,n) = carrier;
      %
      time_vec = clock;
      new_seed = time_vec(6) + 60*time_vec(5) + 3600*time_vec(4) +
86400*time_vec(3);
      rand('seed',new_seed);
      eval(['load ','c:\matlab\data\1khzlin\pmm',int2str(n)])
      bl_noise = filter(b3,a3,rand(1,new_buffer_length));
      dut_sig = dut_sig + noise_scale*bl_noise;
      % see flowchart in writeup
      ref_analytic = hilbert(ref_sig);
      ref_phase = angle(ref_analytic);
      ref_phase_dev = unwrap(ref_phase) - 2*pi*if_freq*new_t;
      ref_phase_phase = angle(hilbert(ref_phase_dev...
```

```
            - mean(ref_phase_dev(l_indx:h_indx))));
        ref_phase_phase_dev = unwrap(ref_phase_phase) - 2*pi*pm_mod_freq*new_t;
        average_ref_phase = mean(ref_phase_phase_dev(l_indx:h_indx));
        if average_ref_phase < 0,
           average_ref_phase = average_ref_phase + 2*pi;
        end
        %
        dut_analytic = hilbert(dut_sig);
        dut_phase = angle(dut_analytic);
        dut_phase_dev = unwrap(dut_phase) - 2*pi*if_freq*new_t;
        dut_phase_phase = angle(hilbert(dut_phase_dev...
              - mean(dut_phase_dev(l_indx:h_indx))));
        dut_phase_phase_dev = unwrap(dut_phase_phase) - 2*pi*pm_mod_freq*new_t;
        average_dut_phase = mean(dut_phase_phase_dev(l_indx:h_indx));
        if average_dut_phase < 0,
           average_dut_phase = average_dut_phase + 2*pi;
        end
        %
        envelope_phase_diff = average_ref_phase - average_dut_phase
        group_delay(1,n) = envelope_phase_diff / (2*pi*pm_mod_freq)
    end                 % next n
    %
    eval(['save ','c:\matlab\data\1khzlin\pmhil',int2str(file_offset + data_iter),'
group_delay']);

%*************************************************************************
end                     % end data collection
clear;
```

APPENDIX I.D. AM CORRELATION RECEIVER METHOD

AMCORR.M

```
clear;
rand('normal');            % set for normal distribution
%
file_offset = input(' Enter data storage file offset =? ')
am_mod_freq = 1E3;
noise_scale = 0.003026;
n_max = 14;                % set number of frequency points
end_collect = 50;
%
if_freq = 50E3;
%
new_buffer_length = 4096;
new_sample_period = 4E-6; % sample at 250 KHz
new_end_time = (new_buffer_length - 1) * new_sample_period;
new_t = 0:new_sample_period:new_end_time;
%
% find number of modulation cycles for integration purposes
mod_samples = fix(1/(am_mod_freq * new_sample_period));
cycles = fix((new_buffer_length - 200)/mod_samples)
l_indx = fix(new_buffer_length/2 - (mod_samples/2)*cycles)
h_indx = round(mod_samples*cycles + l_indx)
last_point = h_indx - l_indx + 1;
%
% find number of carrier cycles for integration purposes
if_samples = fix(1/(if_freq * new_sample_period))
if_cycles = fix((new_buffer_length - 200)/if_samples)
if_l_indx = fix(new_buffer_length/2 - (if_samples/2)*if_cycles)
if_h_indx = round(if_samples*if_cycles + if_l_indx)
if_last_point = if_h_indx - if_l_indx + 1
%
load nsfilt;       % load noise bandlimiting filter
load linfilt4;     % load extraneous upper product filter
load c:\matlab\data\freqlist;
%
sin_if = sin(2*pi*if_freq*new_t);
cos_if = cos(2*pi*if_freq*new_t);
%
for data_iter = 1:end_collect
   for carrier_est_loop = 1:2
      for n = 1:n_max
         %
         carrier = freq_list(n)
```

```
freq_array(1,n) = carrier;
%
if carrier_est_loop == 1,
   time_vec = clock
   new_seed = time_vec(6) + 60*time_vec(5) + 3600*time_vec(4) +
     86400*time_vec(3)
   rand('seed',new_seed);
   eval(['load ','c:\matlab\data\1khzlin\amnom',int2str(n)])
   bl_noise = filter(b3,a3,rand(1,new_buffer_length));
   dut_sig = dut_sig + noise_scale*bl_noise;
   %
else
   time_vec = clock
   new_seed = time_vec(6) + 60*time_vec(5) + 3600*time_vec(4) +
     86400*time_vec(3)
   rand('seed',new_seed);
   eval(['load ','c:\matlab\data\1khzlin\amm',int2str(n)])
   bl_noise = filter(b3,a3,rand(1,new_buffer_length));
   dut_sig = dut_sig +  noise_scale*bl_noise;
end
%
if carrier_est_loop == 1,
%
   ref_if_trunc = ref_sig(if_l_indx:if_h_indx);
   ref_if_sin_prod = ref_if_trunc .* sin_if(if_l_indx:if_h_indx);
   ref_if_cos_prod = ref_if_trunc .* cos_if(if_l_indx:if_h_indx);
   ref_if_sin_int = sum(ref_if_sin_prod) - 0.5*(ref_if_sin_prod(1) +
     ref_if_sin_prod(if_last_point));
   ref_if_cos_int = sum(ref_if_cos_prod) - 0.5*(ref_if_cos_prod(1) +
    ref_if_cos_prod(if_last_point));
    ref_if_phase(n) = - atan2(-ref_if_sin_int, ref_if_cos_int);
   %
   dut_if_trunc = dut_sig(if_l_indx:if_h_indx);
   dut_if_sin_prod = dut_if_trunc .* sin_if(if_l_indx:if_h_indx);
   dut_if_cos_prod = dut_if_trunc .* cos_if(if_l_indx:if_h_indx);
   dut_if_sin_int = sum(dut_if_sin_prod) - 0.5*(dut_if_sin_prod(1) +
     dut_if_sin_prod(if_last_point));
   dut_if_cos_int = sum(dut_if_cos_prod) - 0.5*(dut_if_cos_prod(1) +
     dut_if_cos_prod(if_last_point));
   dut_if_phase(n) = - atan2(-dut_if_sin_int , dut_if_cos_int);
   %
end
%
if carrier_est_loop == 2,
   ref_dwn_cnvt = ref_sig .* cos(2*pi*if_freq*new_t - ref_if_phase(n));
   ref_dwn_cnvt = filtfilt(b4,a4,ref_dwn_cnvt);
```

```
            ref_trunc = ref_dwn_cnvt(l_indx:h_indx);
            ref_trunc = ref_trunc - mean(ref_trunc);
            ref_sin_prod = ref_trunc .* sin(2*pi*am_mod_freq*new_t(l_indx:h_indx));
            ref_cos_prod = ref_trunc .* cos(2*pi*am_mod_freq*new_t(l_indx:h_indx));
            ref_sin_prod_int = sum(ref_sin_prod) - 0.5*(ref_sin_prod(1) +
              ref_sin_prod(last_point));
            ref_cos_prod_int = sum(ref_cos_prod) - 0.5*(ref_cos_prod(1) +
              ref_cos_prod(last_point));
            ref_phase = atan2(-ref_sin_prod_int ,ref_cos_prod_int)
            %
            dut_dwn_cnvt = dut_sig .* cos(2*pi*if_freq*new_t  - dut_if_phase(n));
            dut_dwn_cnvt = filtfilt(b4,a4,dut_dwn_cnvt);
            dut_trunc = dut_dwn_cnvt(l_indx:h_indx);
            dut_trunc = dut_trunc - mean(dut_trunc);
            dut_sin_prod = dut_trunc .* sin(2*pi*am_mod_freq*new_t(l_indx:h_indx));
            dut_cos_prod = dut_trunc .* cos(2*pi*am_mod_freq*new_t(l_indx:h_indx));
            dut_sin_prod_int = sum(dut_sin_prod) - 0.5*(dut_sin_prod(1) +
              dut_sin_prod(last_point));
            dut_cos_prod_int = sum(dut_cos_prod) - 0.5*(dut_cos_prod(1) +
              dut_cos_prod(last_point));
            dut_phase = atan2(-dut_sin_prod_int ,dut_cos_prod_int)
            %
            envelope_phase_diff = -(dut_phase - ref_phase);
            group_delay(1,n) = envelope_phase_diff / (2*pi*am_mod_freq)
          end
        end             % next n
      end               % next carrier_est_loop
      %
      eval(['save ','c:\matlab\data\1khzlin\amcor',int2str(file_offset + data_iter),'
        group_delay']);
end                 % end data collection
%
clear;
```

APPENDIX I.E. PM CORRELATION RECEIVER METHOD

PMCORR.M

```
clear;
rand('normal');                % set to normal distribution
global pm_mod_index ref_cs1 ref_cc2 ref_ss1 ref_sc2 dut_cs1 dut_cc2 dut_ss1 dut_sc2
%
file_offset = input(' Enter data storage file offset =? ')
pm_mod_index = 0.5;
pm_mod_freq = 10E3;
noise_scale = 0.003026;
n_max = 14;                    % set number of frequency points
end_collect = 50;              % collect 50 of each point
%
new_buffer_length = 8192;
new_sample_period = 2E-6;      % sample at 500 KHz
new_end_time = (new_buffer_length - 1) * new_sample_period;
new_t = 0:new_sample_period:new_end_time;
%
% find the number of modulation cycles for integration purposes
mod_samples = fix(1/(pm_mod_freq * new_sample_period));
cycles = fix((new_buffer_length - 200)/mod_samples);
l_indx = fix(new_buffer_length/2 - (mod_samples/2)*cycles)
h_indx = round(mod_samples*cycles + l_indx)
last_point = h_indx - l_indx + 1;
%
% find the number of IF cycles for integration purposes
if_freq = 50E3;
if_samples = fix(1/(if_freq * new_sample_period))
if_cycles = fix((new_buffer_length - 200)/if_samples)
if_l_indx = fix(new_buffer_length/2 - (if_samples/2)*if_cycles)
if_h_indx = round(if_samples*if_cycles + if_l_indx)
if_last_point = if_h_indx - if_l_indx + 1
%
sin_if = sin(2*pi*if_freq*new_t);
cos_if = cos(2*pi*if_freq*new_t);
%
load nsfilt;                   % load bandlimiting noise filter
load c:\matlab\data\freqlist
%
sin_1wm = sin(2*pi*pm_mod_freq*new_t(l_indx:h_indx));
sin_2wm = sin(4*pi*pm_mod_freq*new_t(l_indx:h_indx));
cos_1wm = cos(2*pi*pm_mod_freq*new_t(l_indx:h_indx));
cos_2wm = cos(4*pi*pm_mod_freq*new_t(l_indx:h_indx));
%*************************************************************************
```

```
for data_iter = 1:end_collect
   for carrier_est_loop = 1:2
      for n = 1:n_max
         carrier = freq_list(n);
         freq_array(1,n) = carrier;
         %
         if carrier_est_loop == 1,
            time_vec = clock;
            new_seed = time_vec(6) + 60*time_vec(5) + 3600*time_vec(4) +
              86400*time_vec(3);
            rand('seed',new_seed);
            eval(['load ','c:\matlab\data\10khzlin\pmnom',int2str(n)])
            ref_sig_interp = interp(ref_sig,2); % interpolate by two
            bl_noise = noise_scale*filter(b3,a3,rand(1,new_buffer_length/2));
            dut_sig = dut_sig + bl_noise;
            dut_sig_interp = interp(dut_sig,2); % interpolate by two
         else
            time_vec = clock;
            new_seed = time_vec(6) + 60*time_vec(5) + 3600*time_vec(4) +
              86400*time_vec(3);
            rand('seed',new_seed);
            eval(['load ','c:\matlab\data\10khzlin\pmm',int2str(n)])
            ref_sig_interp = interp(ref_sig,2); % interpolate by two
            bl_noise = noise_scale*filter(b3,a3,rand(1,new_buffer_length/2));
            dut_sig = dut_sig + bl_noise;
            dut_sig_interp = interp(dut_sig,2); % interpolate by two
         end
         if carrier_est_loop == 1,  % estimate IF phase
            %
            disp('carrier phase calc')
            ref_if_trunc = ref_sig_interp(if_l_indx:if_h_indx);
            ref_if_sin_prod = ref_if_trunc .* sin_if(if_l_indx:if_h_indx);
            ref_if_cos_prod = ref_if_trunc .* cos_if(if_l_indx:if_h_indx);
            ref_if_sin_int = sum(ref_if_sin_prod) - 0.5*(ref_if_sin_prod(1) +
              ref_if_sin_prod(if_last_point));
            ref_if_cos_int = sum(ref_if_cos_prod) - 0.5*(ref_if_cos_prod(1) +
              ref_if_cos_prod(if_last_point));
            ref_if_phase(n) = - atan2(-ref_if_sin_int, ref_if_cos_int);
            %
            dut_if_trunc = dut_sig_interp(if_l_indx:if_h_indx);
            dut_if_sin_prod = dut_if_trunc .* sin_if(if_l_indx:if_h_indx);
            dut_if_cos_prod = dut_if_trunc .* cos_if(if_l_indx:if_h_indx);
            dut_if_sin_int = sum(dut_if_sin_prod) - 0.5*(dut_if_sin_prod(1) +
              dut_if_sin_prod(if_last_point));
            dut_if_cos_int = sum(dut_if_cos_prod) - 0.5*(dut_if_cos_prod(1) +
              dut_if_cos_prod(if_last_point));
```

```
            dut_if_phase(n) = - atan2(-dut_if_sin_int , dut_if_cos_int);
          end
          %
          if carrier_est_loop == 2,   % estimate modulation envelope phase
            %
            disp('mod envelope phase calc')
            ref_sin_if = sin(2*pi*if_freq*new_t(l_indx:h_indx) - ref_if_phase(n));
            ref_cos_if = cos(2*pi*if_freq*new_t(l_indx:h_indx) - ref_if_phase(n));
            ref_sig_trunc = ref_sig_interp(l_indx:h_indx);
            r1 = ref_sig_trunc .* ref_sin_if .* cos_1wm;
            r2 = ref_sig_trunc .* ref_cos_if .* cos_2wm;
            r3 = ref_sig_trunc .* ref_sin_if .* sin_1wm;
            r4 = ref_sig_trunc .* ref_cos_if .* sin_2wm;
            %
            ref_cs1 = sum(r1) - 0.5*(r1(1) + r1(last_point)); % trapezoidal integration
            ref_cc2 = sum(r2) - 0.5*(r2(1) + r2(last_point));
            ref_ss1 = sum(r3) - 0.5*(r3(1) + r3(last_point));
            ref_sc2 = sum(r4) - 0.5*(r4(1) + r4(last_point));
            %
            ref_phase = fzero('ref_root',0);  % find solution to phase estimate
            %
            dut_sin_if = sin(2*pi*if_freq*new_t(l_indx:h_indx) - dut_if_phase(n));
            dut_cos_if = cos(2*pi*if_freq*new_t(l_indx:h_indx) - dut_if_phase(n));
            dut_sig_trunc = dut_sig_interp(l_indx:h_indx);
            d1=dut_sig_trunc .* dut_sin_if .* cos_1wm;
            d2=dut_sig_trunc .* dut_cos_if .* cos_2wm;
            d3=dut_sig_trunc .* dut_sin_if .* sin_1wm;
            d4=dut_sig_trunc .* dut_cos_if .* sin_2wm;
            %
            dut_cs1 = sum(d1) - 0.5*(d1(1) + d1(last_point));  % trapezoidal integration
            dut_cc2 = sum(d2) - 0.5*(d2(1) + d2(last_point));
            dut_ss1 = sum(d3) - 0.5*(d3(1) + d3(last_point));
            dut_sc2 = sum(d4) - 0.5*(d4(1) + d4(last_point));
            %
            dut_phase = fzero('dut_root',0); % find solution to phase estimate
            %
            envelope_phase_diff = -(dut_phase - ref_phase);
            group_delay(1,n) = envelope_phase_diff / (2*pi*pm_mod_freq)
          end
        end                 % next n
      end                   % next carrier_est_loop
      eval(['save ','c:\matlab\data\nl_no_eq\pmcor',int2str(file_offset + data_iter),'
group_delay']);
end                         % end data collection
clear;
```

Functions for use with PMCORR.M

REF_ROOT.M

```
function y = f(ref_phase_est)
   ref_odd_terms = bessel(1,pm_mod_index)*(ref_cs1*cos(ref_phase_est)...
                                - ref_ss1*sin(ref_phase_est));
   ref_even_terms = 2*bessel(2,pm_mod_index)*(ref_cc2*sin(2*ref_phase_est)...
                                  + ref_sc2*cos(2*ref_phase_est));
   y = ref_odd_terms + ref_even_terms;
```

DUT_ROOT.M

```
function y = f(dut_phase_est)
   dut_odd_terms = bessel(1,pm_mod_index)*(dut_cs1*cos(dut_phase_est)...
                                - dut_ss1*sin(dut_phase_est));
   dut_even_terms = 2*bessel(2,pm_mod_index)*(dut_cc2*sin(2*dut_phase_est)...
                                  + dut_sc2*cos(2*dut_phase_est));
   y = dut_odd_terms + dut_even_terms;
```

APPENDIX I.F. LMS ALGORITHM

LMSADAP.M

```
clear;
axis([1 2 3 4]);axis;                    % auto scale plots
%
am_mod_index = input('Enter am modulation index, =? ');
am_mod_freq = input('Enter am modulation frequency, =? ');
mu = input('Enter mu, =? ');
no_taps = input('Enter number of taps, =? ') - 1;
noise_var = input('Enter the noise variance, =? ');
buffer_exp = input('Enter the buffer size exponent, =? ');
carrier = 50E3;
%
%
buffer_length = 2^buffer_exp;
sample_period = 4E-6;                    % sample at 250 KHZ
end_time = (buffer_length + 32 - 1) * sample_period;
t_ = 0:sample_period:end_time;
%
t = t_(1:buffer_length);
%
%*************************************************************************
disp(' Creating data arrays ')
%
ref_sig_1 = (1 + am_mod_index*cos(2*pi*am_mod_freq*t_)) .* cos(2*pi*carrier*t_);
load buttfilt;                           % load DUT filter
rand('uniform');
dut_sig_1 = filter(b2,a2,ref_sig_1);
ref_sig = ref_sig_1(33:buffer_length + 32);
dut_sig = dut_sig_1(33:buffer_length + 32) + noise_var*rand(1,buffer_length);
%
plot(t,ref_sig,':',t,dut_sig,'-')
title(' ref_sig = ..... , dut_sig = ____ ')
pause
%*************************************************************************
disp(' Adaptive filtering ')
%
n_ = buffer_length;
m = no_taps;
mp1 = m + 1;
h = zeros(1,mp1);
w = zeros(1,mp1);
xhat_vec = zeros(1,n_);
h(1) = 0;
```

```
                                        % End initialization
%
for loop=1:n_;
   w(1) = ref_sig(loop);
   xhat = sum(h .* w);                  % find x_est by dotting vectors
   xhat_vec(loop) = xhat;
   e_ = dut_sig(loop) - xhat;
   h = h + 2*mu*e_ * w;                 % update coefficient vector
   w(2:mp1) = w(1:m);                   % reassign to old vector
end
%*************************************************************************
weight_vec = h;
last = buffer_length - 500;
if last <=0 ,
   last = 1;
end
plot(t(last:n_),xhat_vec(last:n_),':',t(last:n_),dut_sig(last:n_),'-')
title('y_out_vec = .... , dut_sig = ----- ')
pause
error_sig_vec = dut_sig - xhat_vec;
plot(error_sig_vec(1:n_))
title([' AM mod , error_sig_vec, mu = ',num2str(mu),'  taps = ',num2str(mp1)])
ylabel('Error')
xlabel('Buffer entry number')
pause
plot((4096 - 500):4096,error_sig_vec(last:n_))
title([' AM mod , error_sig_vec, mu = ',num2str(mu),'  taps = ',num2str(mp1)])
ylabel('Error')
xlabel('Buffer entry number')
pause
bar(weight_vec)
grid
pause
%*************************************************************************
b3 = weight_vec;
a3 = zeros(b3);
a3(1) = 1;
[h3,w3] = freqz(b3,a3,1024);  % find the frequency response of filter
weight_phase = angle(h3);
[h2,w2] = freqz(b2,a2,1024);
right_phase = angle(h2);
axis([1 2 3 4]);axis;
plot(w3*125E3/pi,weight_phase,':',w2*125E3/pi,right_phase,'-')
title(['adaptive filter phase = ..., filter phase = ____, mu = ',num2str(mu)])
xlabel('Frequency in Hertz')
ylabel('Phase in Radians')
```

```
grid
pause
%*************************************************************************
test_sig = filter(b3,a3,ref_sig);
difference = dut_sig_1(33:buffer_length + 32) - test_sig;
plot(t(50:buffer_length),difference(50:buffer_length))
title(['Plot of dut_sig - test_sig,  mu = ',num2str(mu)])
grid
pause
%*************************************************************************
%
clear;
```

APPENDIX I.G. RLS ALGORITHM

RLSADAP4.M

```
clear;
axis([1 2 3 4]);axis;                    % set auto scaling
%
am_mod_index = input('Enter am modulation index, =? ');
am_mod_freq = input('Enter am modulation frequency, =? ');
lambda = input('Enter lambda, =? ');
no_taps = input('Enter number of taps, =? ');
noise_var = input('Enter the noise variance, =? ');
buffer_exp = input('Enter the buffer exponent, =? ');
carrier = 50E3;
%
%
buffer_length = 2^buffer_exp;
sample_period = 4E-6;                    % sample at 250 KHZ
end_time = (buffer_length + 32 - 1) * sample_period;
t_ = 0:sample_period:end_time;
%
t = t_(1:buffer_length);
%
%*************************************************************************
disp(' Creating data arrays ')
%
ref_sig_1 = (1 + am_mod_index*cos(2*pi*am_mod_freq*t_)) .* cos(2*pi*carrier*t_);
%
load buttfilt;                           % load DUT filter
rand('uniform');
dut_sig_1 = filter(b2,a2,ref_sig_1);
ref_sig = ref_sig_1(33:buffer_length + 32);
dut_sig = dut_sig_1(33:buffer_length + 32) + noise_var*rand(1,buffer_length);
%
save c:\tc\user\dut_sig dut_sig;         % store data vectors
save c:\tc\user\ref_sig ref_sig;
%
%*************************************************************************
buff_exp_ = buffer_exp;
length___ = no_taps;
lambda___ = lambda;
save c:\tc\user\param buff_exp_ length___ lambda___; % store parameters
%*************************************************************************
disp(' Adaptive filtering ')
%
!cd c:\tc\user
```

```
!rls                              % call C subroutine
load x_est                          % load resulting estimate vector
load w_avg                           % load coefficient vector
!cd c:\matlab\user
%
%*************************************************************************
weight_vec = w_n_averg
xhat_vec = x_est_vec;
n_ = buffer_length;
mp1 = no_taps;
last = buffer_length - 500;
if last <=0,
   last = 1;
end
plot(t(last:n_),xhat_vec(last:n_),':',t(last:n_),dut_sig(last:n_),'-')
title('y_out_vec = .... , dut_sig = ----- ')
pause
error_sig_vec = dut_sig - xhat_vec;
plot(error_sig_vec(1:n_))
title([' AM mod , error_sig_vec, lambda = ',num2str(lambda),'  taps = ',num2str(mp1)])
xlabel('Buffer entry number')
ylabel('Error')
pause
plot((4096-500):4096,error_sig_vec(last:n_))
title([' AM mod , error_sig_vec, lambda = ',num2str(lambda),'  taps = ',num2str(mp1)])
xlabel('Buffer entry number')
ylabel('Error')
pause
bar(weight_vec)
grid
pause
%*************************************************************************
b3 = weight_vec;
a3 = zeros(b3);
a3(1) = 1;
[h3,w3] = freqz(b3,a3,1024);    % find frequency response of filter
weight_phase = angle(h3);
[h2,w2] = freqz(b2,a2,1024);
right_phase = angle(h2);
h4 = freqz(b3,a3,[pi*40/125 pi*50/125 pi*60/125])
test_phase = angle(h4)
pause
axis([1 2 3 4]);axis;
plot(w3*125E3/pi,weight_phase,':',w2*125E3/pi,right_phase,'-')
title(['adaptive filter phase = ..., filter phase = ____, lambda = ',num2str(lambda)])
grid
```

```
xlabel('Frequency in Hertz')
ylabel('Phase in Radians')
pause
%*************************************************************************
test_sig = filter(b3,a3,ref_sig);
difference = dut_sig_1(33:buffer_length + 32) - test_sig;
plot(t(50:buffer_length),difference(50:buffer_length))
title(['Plot of dut_sig_1 - test_sig,  lambda= ',num2str(lambda)])
grid
%*************************************************************************
%
clear;
```

APPENDIX I.H. LINEAR SYSTEM IDENTIFICATION METHOD

SYSEST2.M

```
clear;
rand('normal');            % set to normal distribution
%
mod_type = input('Enter modulation type, am or pm ? ','s');
no_taps = input('Enter number of taps, =? ');
file_offset = input(' Enter data storage file offset =? ')
mod_freq = 10E3;
noise_scale = 0.003026;
n_max = 14;                % set number of frequency points
end_collect = 50;          % set number of these points collected
%
if_freq = 50E3;
%
new_buffer_length = 4096;
new_sample_rate = 250E3        % sample rate is 250kHz
new_sample_period = 1/new_sample_rate;
new_t = 0:new_sample_period:new_sample_period*(new_buffer_length - 1);
%
load nsfilt;               % load bandlimiting noise filter
load c:\matlab\data\freqlist;
%
buff_exp_ = 12;
length___ = no_taps;
lambda___ = 1;
save c:\tc\user\param buff_exp_ length___ lambda___;% store parameter vector
%
%*****************************************************************************
for data_iter = 1:end_collect
   for n = 1:n_max
      %
      carrier = freq_list(n)
      %
      freq_array(1,n) = carrier;

%*****************************************************************************
    time_vec = clock
      new_seed = time_vec(6) + 60*time_vec(5) + 3600*time_vec(4) + 86400*time_vec(3)

      rand('seed',new_seed);
      if strcmp(mod_type,'am') == 1,
         eval(['load ','c:\matlab\data\10khzlin\amm',int2str(n)])
         disp('am file loaded')
```

```
      else
         eval(['load ','c:\matlab\data\10khzlin\pmm',int2str(n)])
         disp('pm file loaded')
      end
      bl_noise = filter(b3,a3,rand(1,new_buffer_length));
      dut_sig = dut_sig +  noise_scale*bl_noise;
      %
      save c:\tc\user\dut_sig dut_sig; % save data vectors
      save c:\tc\user\ref_sig ref_sig;
%*****************************************************************************
      disp(' Adaptive filtering ')
      !cd c:\tc\user
      !rls                % call C subroutine
      load x_est            % load estimate vector
      load w_avg             % load adaptive filter coefficient vector
      !cd c:\matlab\user
%*****************************************************************************
      b1 = w_n_averg;
      a1 = zeros(b1);
      a1(1) = 1;
      low_sideband = 2*pi*(if_freq - mod_freq)/new_sample_rate;
      up_sideband = 2*pi*(if_freq + mod_freq)/new_sample_rate;
      h1 = freqz(b1,a1,[low_sideband up_sideband]); % find frequency response of filter
      dut_phase = angle(h1);
%*****************************************************************************
      %
      envelope_phase_diff = abs(dut_phase(2) - dut_phase(1))
      if envelope_phase_diff > pi,
         envelope_phase_diff = abs(envelope_phase_diff - 2*pi);
      end
      group_delay(1,n) = envelope_phase_diff / (4*pi*mod_freq)
      %
   end                  % next n
%*****************************************************************************
   if strcmp(mod_type,'am'),
      eval(['save ','c:\matlab\data\nl_no_eq\amlin',int2str(file_offset + data_iter),'
group_delay']);
   else
      eval(['save ','c:\matlab\data\nl_no_eq\pmlin',int2str(file_offset + data_iter),'
group_delay']);
   end
end                     % end data collection
%
clear;
```

RLS.C C Subroutine for RLS Algorithm

```
/* ************************************************************************** */
# include <stdlib.h>
# include <stdio.h>
# include <conio.h>
# include <dos.h>
# define  MAX_TAPS 750
# define  BUF_SIZE 512
# define  NUM_CNT  512
# define  delta  0.001
/* ************************************************************************** */
main()
{
   FILE *file_in1 ,*file_in2, *file_in3, *file_out1, *file_out2, *file_out3;

   double buffer_size_exp,length_flt, w_n_avg[MAX_TAPS+1];
   double ref_sig_buf[BUF_SIZE], dut_sig_buf[BUF_SIZE];
   double averages ,x_est_buf[BUF_SIZE];

   double w[MAX_TAPS+1],a[MAX_TAPS+1],b[MAX_TAPS+1],ktilde[MAX_TAPS];
   double D1a,D1b,nutilde;
   double k[MAX_TAPS+1],kbar[MAX_TAPS],nu,nubar;
   double e0a,e0b,e1a,e1b,D0a,D0b,e0,xhat0;
   double h[MAX_TAPS +1], xhat, e_, lambda;

   unsigned long type, mrows, ncols1, imagef, namelen, ncols2;
   unsigned int data_loops, taps, i, j, l, n, x_index;
   unsigned int near_end, shift_val, M;

   char var_name1[10], var_name2[10];
   char file_name1[14], file_name2[14], file_name3[14], file_name4[14];
   char file_name5[14], junk[30];

/* ************************************************************************** */
   strcpy(var_name1,"x_est_vec");
   strcpy(var_name2,"w_n_averg");
   strcpy(file_name1,"param.mat");
   strcpy(file_name2,"dut_sig.mat");
   strcpy(file_name3,"ref_sig.mat");
   strcpy(file_name4,"x_est.mat");
   strcpy(file_name5,"w_avg.mat");
/* ************************************************************************** */
   file_in1 = fopen(file_name1, "rb");
   file_in2 = fopen(file_name2, "rb");
   file_in3 = fopen(file_name3, "rb");
```

```
   file_out1 = fopen(file_name4, "wb");
   file_out2 = fopen(file_name5, "wb");
/* ************************************************************************ */
   fread(junk, 30, 1, file_in1);          /* read parameter file  */
   fread(&buffer_size_exp , 8, 1, file_in1);
   fread(junk, 30, 1, file_in1);
   fread(&length_flt, 8, 1, file_in1);
   fread(junk, 30, 1, file_in1);
   fread(&lambda, 8, 1, file_in1);
   fclose(file_in1);
/* ************************************************************************ */
   taps = length_flt;
   M = taps - 1;
   shift_val = buffer_size_exp;
   near_end = BUF_SIZE - 50;
   data_loops = 1 << (shift_val - 9);
   type =  0;              /* initialize matlab file header parameters */
   mrows = 1;
   ncols1 = 1;
   for(i = 1;i < shift_val + 1; i++) {
      ncols1 = 2*ncols1;
   }
   ncols2 = taps;
   imagef = 0;
   namelen = 10;
/* ************************************************************************ */
/* Fast RLS algorithm taken from Orfanidis, "Optimum Signal Processing" p576 */

   for(i=1;i<=M;i++){
      h[i] = 0;
      a[i] = 0;
      b[i-1] = 0;
      w[i] = 0;
      ktilde[i-1] = 0;
   }
   D1a = delta;
   D1b = delta;
   nutilde = 0;
   h[0] = 0;
   a[0] = 1;
   b[M] = 1;
/* ************************************************************************ */
   fwrite(&type, 4, 1, file_out1);     /* write out file header for estimate vector */
   fwrite(&mrows, 4, 1, file_out1);
   fwrite(&ncols1, 4, 1, file_out1);
   fwrite(&imagef, 4, 1, file_out1);
```

```
   fwrite(&namelen, 4, 1, file_out1);
   fwrite(var_name1, 1, 10, file_out1);
/* ************************************************************************** */
   fwrite(&type, 4, 1, file_out2);     /* write out file header for filter coefficient vector */
   fwrite(&mrows, 4, 1, file_out2);
   fwrite(&ncols2, 4, 1, file_out2);
   fwrite(&imagef, 4, 1, file_out2);
   fwrite(&namelen, 4, 1, file_out2);
   fwrite(var_name2, 1, 10, file_out2);
/* ************************************************************************** */
   fread(junk, 28, 1, file_in2);       /* read in and throw away unwanted file headers */
   fread(junk, 28, 1, file_in3);       /* from ref and DUT data vectors  */
/* ************************************************************************** */
   for(i = 0; i <= M; i++) {
      w_n_avg[i] = 0.0;
   }
   averages = 0.0;
   for(l = 1; l < data_loops + 1; l++) {
      fread(&ref_sig_buf[0], 8, NUM_CNT, file_in3); /* read in 512 numbers from data
        vectors */
      fread(&dut_sig_buf[0], 8, NUM_CNT, file_in2);
      for(n = 0; n < BUF_SIZE; n++) {
         w[0] = ref_sig_buf[n];
         for(e0a=0,i=0;i<=M;i++)
            e0a += a[i]*w[i];
         e1a = e0a/(1+nutilde);
         D0a = lambda*D1a;
         k[0] = e0a/D0a;
         D1a = D0a+e1a*e0a;
         for (i=1;i<=M;i++)
            k[i] = ktilde[i-1] +k[0]*a[i];
         D0b = lambda * D1b;
         e0b = k[M] * D0b;
         for (i=0;i<M;i++)
            kbar[i] = k[i] - k[M]*b[i];
         nu = nutilde + e0a*k[0];
         nubar = nu-e0b*k[M];
         e1b = e0b / (1+nubar);
         D1b = D0b + e1b*e0b;
         for(i=1;i<=M;i++)
            a[i] -= e1a*ktilde[i-1];
         for(i=0;i<M;i++)
            b[i] -= e1b*kbar[i];
         for (xhat0=0,i=0;i<=M;i++)
            xhat0 += h[i]*w[i];
         e0 = dut_sig_buf[n] - xhat0;
```

```
        .
        e_ = e0/(1+nu);
/*      if ((l == 1)||(l == data_loops)) {
           printf("  e_ is  %e  n is  %d   l is  %i \n",e_,n,l);
        }                                                    */
        xhat = dut_sig_buf[n] - e_;
        x_est_buf[n] = xhat;
        for(i=0;i<=M;i++)
           h[i] += e_*k[i];
        if ((l == data_loops)&&(n > near_end)) {
           for(i = 0; i <= M; i++) {
              w_n_avg[i] = w_n_avg[i] + h[i]; /* average last few coefficient vectors */
           }
           averages = averages + 1;
        }
        for(i=1;i<=M;i++)
           ktilde[i-1] = kbar[i-1];
        nutilde = nubar;
        for(i=M;i>=1;i--)
           w[i] = w[i-1];
     }          /* next n  , next point in the buffer */
     fwrite(&x_est_buf[0],8,NUM_CNT,file_out1); /* write out 512 estimate vector points
      */
  }             /* next l , read in another 512 points */
  for(i = 0; i <= M; i++) {
     w_n_avg[i] = w_n_avg[i]/averages;
  }
  fwrite(&w_n_avg[0],8,taps,file_out2); /* write out the adaptive filter coefficient vector */
  fclose(file_in2);
  fclose(file_in3);
  fclose(file_out1);
  fclose(file_out2);
}                   /* end of main  */

/* ************************************************************************** */
```

APPENDIX II.A. PROGRAM FOR SECOND ORDER NONLINEAR SYSTEM IDENTIFICATION

FASTNS.M

```
clear;
rand('normal');            % set to normal distribution
axis([1 2 3 4]);axis;      % set to auto scaling
%
mu = input('Enter mu, =? ');
square_coeff = input('Enter square term coefficient, =? ');
N = input('Enter number of taps, =? ');
```

```
noise_scale = input('Enter the noise scale, =? ');
buffer_exp = input('Enter the buffer exponent, =? ');
carrier = 50E3;
%
buffer_length = 2^buffer_exp;
sample_period = 4E-6;                    % sample at 250 KHZ
end_time = (buffer_length + 32 - 1) * sample_period;
t_ = 0:sample_period:end_time;
%
t = t_(1:buffer_length);
%
%*************************************************************************
disp(' Creating data arrays ')
%
ref_sig_1 = noise_scale*rand(1,buffer_length + 32);
%ref_sig_2 = ref_sig_1 + square_coeff*ref_sig_1 .* ref_sig_1;
%
b2 = [0.3 0.4 0.3 0.2 0.1];              % linear FIR filter coefficients
a2 = [1];
dut_sig_1 = filter(b2,a2,ref_sig_1);
dut_sig_2 = dut_sig_1 + square_coeff*dut_sig_1.^2;  % second order nonlinearity
ref_sig = ref_sig_1(33:buffer_length + 32);
dut_sig = dut_sig_2(33:buffer_length + 32);
%
save c:\tc\user\dut_sig dut_sig;
save c:\tc\user\ref_sig ref_sig;          % save distorted data vectors
buff_exp_ = buffer_exp;
length___ = N;
mu_______ = mu;
save c:\tc\user\param buff_exp_ length___ mu_______;    % save parameter vector
plot(t,ref_sig,':',t,dut_sig,'-')
title(' ref_sig = ..... , dut_sig = ____ ')
pause
%*************************************************************************
disp(' Adaptive filtering ')
%
! cd c:\tc\user
!filt2d                    % call adaptive filter C subroutine
load x_est                  % load estimate vector
load w_avg                  % load adaptive filter coefficients
load lin_indx               % load linear part of adaptive filter
! cd c:\matlab\user
%*************************************************************************
weight_vec = w_n_averg;
linear_weight_vec = weight_vec(lin_index);
last = buffer_length - 500;
```

```
if last <=0 ,
   last = 1;
end
n_ = buffer_length;
plot(t(last:n_),x_est_vec(last:n_),':',t(last:n_),dut_sig(last:n_),'-')
title('x_est_vec = .... , dut_sig = ----- ')
pause
error_sig_vec = dut_sig - x_est_vec;
plot(t(1:n_),error_sig_vec(1:n_))
title([' AM mod , error_sig_vec, mu = ',num2str(mu),'  taps = ',num2str(N)])
pause
plot(t(last:n_),error_sig_vec(last:n_))
title([' AM mod , error_sig_vec, mu = ',num2str(mu),'  taps = ',num2str(N)])
pause
bar(weight_vec)
xlabel('Adaptive filter vector coefficient number')
ylabel('Filter coefficient value')
grid
pause
%*************************************************************************
b1 = linear_weight_vec;
a1 = zeros(b1);
a1(1) = 1;
[h1,w1] = freqz(b1,a1,1024);        % calculate frequency response of linear part of filter
weight_phase = angle(h1);
[h2,w2] = freqz(b2,a2,1024);
right_phase = angle(h2);
axis([1 2 3 4]);axis;
plot(w1*125E3/pi,weight_phase,':',w2*125E3/pi,right_phase,'-')
title(['adaptive filter phase = ..., filter phase = ---, mu = ',num2str(mu)])
grid
pause
%*************************************************************************
clear;
```

FILT2D.C C SUBROUTINE FOR SECOND ORDER LMS FILTER

```
/* ************************************************************************ */
# include <stdlib.h>
# include <stdio.h>
# include <conio.h>
# include <dos.h>
# define  MAX_TAPS 1325
# define  BUF_SIZE 512
# define  NUM_CNT  512

/* ************************************************************************ */
main()
{
   FILE *file_in1 ,*file_in2, *file_in3, *file_out1, *file_out2, *file_out3;

   double buffer_size_exp,length_flt, mu, w_n_avg[MAX_TAPS+1];
   double x_n[MAX_TAPS + 1], ref_sig_buf[BUF_SIZE + 129];
   double dut_sig_buf[BUF_SIZE + 1],x_est_buf[BUF_SIZE + 1];
   double w_n[MAX_TAPS + 1], w_n_1[MAX_TAPS + 1];
   double linear_indx[129], e_n, averages;

   unsigned long type, mrows, ncols1, imagef, namelen, ncols2, ncols3;
   unsigned int data_loops, taps, i, j, k , l, n, x_index, length;
   unsigned int near_end, shift_val;

   char var_name1[10], var_name2[10], var_name3[10];
   char file_name1[14], file_name2[14], file_name3[14], file_name4[14];
   char file_name5[14], file_name6[14], junk[30];

/* ************************************************************************ */
   strcpy(var_name1,"x_est_vec");
   strcpy(var_name2,"w_n_averg");
   strcpy(var_name3,"lin_index");
   strcpy(file_name1,"param.mat");
   strcpy(file_name2,"dut_sig.mat");
   strcpy(file_name3,"ref_sig.mat");
   strcpy(file_name4,"x_est.mat");
   strcpy(file_name5,"w_avg.mat");
   strcpy(file_name6,"lin_indx.mat");
/* ************************************************************************ */
   file_in1 = fopen(file_name1, "rb");
   file_in2 = fopen(file_name2, "rb");
   file_in3 = fopen(file_name3, "rb");
   file_out1 = fopen(file_name4, "wb");
   file_out2 = fopen(file_name5, "wb");
```

```
   file_out3 = fopen(file_name6, "wb");
/* ************************************************************************ */
   fread(junk, 30, 1, file_in1);              /*  read in parameters   */
   fread(&buffer_size_exp , 8, 1, file_in1);
   fread(junk, 30, 1, file_in1);
   fread(&length_flt, 8, 1, file_in1);
   fread(junk, 30, 1, file_in1);
   fread(&mu, 8, 1, file_in1);
   fclose(file_in1);
/* ************************************************************************ */
   length = length_flt;
   taps = (length_flt*(length_flt + 3)/2);
   shift_val = buffer_size_exp;
   near_end = BUF_SIZE - 50;
   data_loops = 1 << (shift_val - 9);
   type =  0;                          /* initialize matlab file header parameters */
   mrows = 1;
   ncols1 = 1;
   for(k = 1;k < shift_val + 1; k++) {
      ncols1 = 2*ncols1;
   }
   ncols2 = taps;
   ncols3 = length;
   imagef = 0;
   namelen = 10;
   /* ********************************************************************** */
   fwrite(&type, 4, 1, file_out1);        /* write out estimate vector file header */
   fwrite(&mrows, 4, 1, file_out1);
   fwrite(&ncols1, 4, 1, file_out1);
   fwrite(&imagef, 4, 1, file_out1);
   fwrite(&namelen, 4, 1, file_out1);
   fwrite(var_name1, 1, 10, file_out1);
/* ************************************************************************ */
   fwrite(&type, 4, 1, file_out2);  /* write out adaptive filter coefficient file header */
   fwrite(&mrows, 4, 1, file_out2);
   fwrite(&ncols2, 4, 1, file_out2);
   fwrite(&imagef, 4, 1, file_out2);
   fwrite(&namelen, 4, 1, file_out2);
   fwrite(var_name2, 1, 10, file_out2);
/* ************************************************************************ */
   fwrite(&type, 4, 1, file_out3); /* write out linear filter coefficient file header */
   fwrite(&mrows, 4, 1, file_out3);
   fwrite(&ncols3, 4, 1, file_out3);
   fwrite(&imagef, 4, 1, file_out3);
   fwrite(&namelen, 4, 1, file_out3);
   fwrite(var_name3, 1, 10, file_out3);
```

```
/* ************************************************************************** */
   fread(junk, 28, 1, file_in2);  /* read in and throw away ref and DUT data file headers */
   fread(junk, 28, 1, file_in3);
/* ************************************************************************** */
   for(k = 1; k < taps + 1; k++) { /* initialize coefficient vectors */
      w_n_avg[k] = 0.0;
      w_n_1[k] = 0.0;
   }
   for(k = BUF_SIZE + 1; k < BUF_SIZE + 129; k++) {
      ref_sig_buf[k] = 0.0;
   }
   averages = 0.0;
   for(l = 1; l < data_loops + 1; l++) {
      for(k = 1; k < 129 ; k++) {
         ref_sig_buf[k] = ref_sig_buf[k + BUF_SIZE]; /* relocate upper 128 points to
           beginning */
      }
      fread(&ref_sig_buf[129], 8, NUM_CNT, file_in3); /* read in 512 data points */
      fread(&dut_sig_buf[1], 8, NUM_CNT, file_in2);
      for(n = 1; n < BUF_SIZE + 1; n++) {
         x_index = 1;
         for(i = 1; i < length + 1; i++) {
            x_n[x_index] = ref_sig_buf[n-i+129];
            linear_indx[i] = x_index++; /* keep track of the first order coefficients */
            for(j = i; j < length + 1; j++) {
               x_n[x_index]=ref_sig_buf[n-i+129]*ref_sig_buf[n-j+129]; /* prepare data
                 vector for dot product */
               ++x_index;
            }
         }
         x_est_buf[n] = 0.0;
         for(k = 1; k < taps + 1; k++) {
            x_est_buf[n] = x_est_buf[n] + x_n[k] * w_n_1[k]; /* filter input data */
         }
         e_n = dut_sig_buf[n] - x_est_buf[n]; /* calculate error */
         if ((l == 1)||(l == data_loops)) {
           printf("  e_n is  %e  n is  %d   l is  %i \n",e_n,n,l);
         }
         for(k = 1; k < taps + 1; k++) {
            w_n[k] = w_n_1[k] + 2*mu*e_n*x_n[k]; /* update adaptive filter coefficients */

            w_n_1[k] = w_n[k]; /* reassign to "old" coefficient vector */
         }
         if ((l == data_loops)&&(n > near_end)) {
            for(k = 1; k < taps + 1; k++) {
               w_n_avg[k] = w_n_avg[k] + w_n[k]; /* average last few filter coefficient
```

```
                    vectors */
                }
                averages = averages + 1;
            }
        } /* next n  */
        fwrite(&x_est_buf[1],8,NUM_CNT,file_out1);/* write out 512 estimated points */
    } /* next l  */
    for(k = 1; k < taps + 1; k++) {
        w_n_avg[k] = w_n_avg[k]/averages;
    }
    fwrite(&w_n_avg[1],8,taps,file_out2); /* write out averaged adaptive filter coefficient
      vector */
    fwrite(&linear_indx[1],8,taps,file_out3);/* write out linear part of adaptive filter */
    fclose(file_in2);
    fclose(file_in3);
    fclose(file_out1);
    fclose(file_out2);
    fclose(file_out3);
}               /* end of main  */

/* ************************************************************************** */
```

APPENDIX II.B. SUBROUTINE FOR SECOND ORDER NONLINEAR EQUALIZER

NL_EQUAL.M

```
%*************************************************************************
function y = nl_equal(x)
%
disp(' Equalizing ')
%
load c:\matlab\data\w_avg;      % load adaptive filter coefficients
load c:\matlab\data\lin_index   % load index to linear part of filter
%
linear_weight_vec = w_n_averg(lin_index);
%
second_order_coeff = w_n_averg(2)/(w_n_averg(1)^2);% calculate the second order
coefficient
%
nl_buf = .x - second_order_coeff*x.^2 + 2*(second_order_coeff^2)*x.^3; % and equalize the
nonlinearity
%
y = filter([1],linear_weight_vec,nl_buf); % equalize the linear part
%
%*************************************************************************
```

APPENDIX III. SAMPLE PROGRAM FOR THE MONTE CARLO ANALYSIS

MONTE10D.M

```
clear;
%
%
f_end = input('Enter last file suffix =? ')
%
load c:\matlab\data\freqlist;   % load the frequency list
load c:\matlab\data\idealgrp;   % load the true group delay
load c:\matlab\data\snrlist;    % load the SNR list
%
%
for file_loop = 1:f_end + 1
   %
   eval(['load ','d:\10khzlin\amhil',int2str(file_loop - 1)])
   amhil(file_loop,:) = group_delay;
   eval(['load ','d:\10khzlin\pmhil',int2str(file_loop - 1)])
   pmhil(file_loop,:) = group_delay;
   eval(['load ','d:\10khzlin\amcor',int2str(file_loop - 1)])
   amcor(file_loop,:) = group_delay;
   eval(['load ','d:\10khzlin\pmcor',int2str(file_loop - 1)])
   pmcor(file_loop,:) = group_delay;
   eval(['load ','d:\10khzlin\amlin',int2str(file_loop - 1)])
   amlin(file_loop,:) = group_delay;
   eval(['load ','d:\10khzlin\pmlin',int2str(file_loop - 1)])
   pmlin(file_loop,:) = group_delay;
   %
end
%
%
for freq = 1:14
%
%
   nn_amhil_bias_per(freq) = 100*(amhil(1,freq) - ideal(freq))/ideal(freq);
   nn_amcor_bias_per(freq) = 100*(amcor(1,freq) - ideal(freq))/ideal(freq);
   nn_amlin_bias_per(freq) = 100*(amlin(1,freq) - ideal(freq))/ideal(freq);
   %
   amhil_bias_per(freq) = 100*(mean(amhil(2:f_end,freq)) - ideal(freq))/ideal(freq);
   amcor_bias_per(freq) = 100*(mean(amcor(2:f_end,freq)) - ideal(freq))/ideal(freq);
   amlin_bias_per(freq) = 100*(mean(amlin(2:f_end,freq)) - ideal(freq))/ideal(freq);
   %
   %
   amhil_relvar(freq) = 100*std(amhil(2:f_end,freq))/mean(amhil(2:f_end,freq));
   amcor_relvar(freq) = 100*std(amcor(2:f_end,freq))/mean(amcor(2:f_end,freq));
```

```
   amlin_relvar(freq) = 100*std(amlin(2:f_end,freq))/mean(amlin(2:f_end,freq));
   %
end
%
axis([freq_list(1) freq_list(14) -.75 .75]);
plot(freq_list,nn_amhil_bias_per,'+',freq_list,nn_amcor_bias_per,'*',freq_list,nn_amlin_bias_
per,'o')
title('No noise % error, 10Khz AM mod., hilb = "+", corr = "*", sysest = "o"')
xlabel(' Frequency in Hz ')
ylabel(' Percentage error in measurement')
grid
pause
%
%
axis([snr_list(14) snr_list(1) -15.0 15.0]);
plot(snr_list,nn_amhil_bias_per,'+',snr_list,nn_amcor_bias_per,'*',snr_list,nn_amlin_bias_per
  ,'o')
title('Rel. (%) error, 10Khz AM mod. hilb = "+", corr = "*", sysest = "o"')
xlabel(' Signal to noise ratio in dB ')
ylabel(' Percentage error in measurement')
grid
pause
%
axis([freq_list(1) freq_list(14) -100 100]);
plot(freq_list,amhil_bias_per,'+',freq_list,amcor_bias_per,'*',freq_list,amlin_bias_per,'o')
title('Rel. % error, 10Khz AM mod., hilb = "+", corr = "*", sysest = "o"')
xlabel(' Frequency in Hz ')
ylabel(' Percentage error in measurement')
grid
pause
%
axis([snr_list(14) snr_list(1) -100 100]);
plot(snr_list,amhil_bias_per,'+',snr_list,amcor_bias_per,'*',snr_list,amlin_bias_per,'o')
title('Rel. % error, 10Khz AM mod., hilb = "+", corr = "*", sysest = "o"')
xlabel(' Signal to noise ratio in dB ')
ylabel(' Percentage error in measurement')
grid
pause
%
axis([snr_list(14) snr_list(1) -10 10]);
plot(snr_list,amhil_bias_per,'+',snr_list,amcor_bias_per,'*',snr_list,amlin_bias_per,'o')
title('Rel. (%) error, 10Khz AM mod.  hilb = "+",  corr = "*",  sysest = "o"')
xlabel(' Signal to noise ratio in dB ')
ylabel(' Percentage error in measurement')
grid
pause
```

```
%
axis([freq_list(1) freq_list(14) -100 100]);
plot(freq_list,amhil_relvar,'+',freq_list,amcor_relvar,'*',freq_list,amlin_relvar,'o')
title('Coeff. of var.  10Khz AM mod.  hilb = "+"  corr = "*"  sysest = "o"')
xlabel(' Frequency in Hz ')
ylabel(' Percentage variation in measurement')
grid
pause
%
axis([snr_list(14) snr_list(1) 0 100]);
plot(snr_list,amhil_relvar,'+',snr_list,amcor_relvar,'*',snr_list,amlin_relvar,'o')
title('Coeff. of var., 10Khz AM mod., hilb = "+"  corr = "*"  sysest = "o"')
xlabel(' Signal to noise ratio in dB ')
ylabel(' Percentage variation in measurement')
grid
pause
%
axis([snr_list(14) snr_list(1) -10 10]);
plot(snr_list,amhil_relvar,'+',snr_list,amcor_relvar,'*',snr_list,amlin_relvar,'o')
title('Coeff. of var.  10Khz AM mod.  hilb = "+"  corr = "*"  sysest = "o"')
xlabel(' Signal to noise ratio in dB ')
ylabel(' Percentage variation in measurement')
grid
pause
% ************************************************************************
%
for freq = 1:14
   %
   nn_pmhil_bias_per(freq) = 100*(pmhil(1,freq) - ideal(freq))/ideal(freq);
   nn_pmcor_bias_per(freq) = 100*(pmcor(1,freq) - ideal(freq))/ideal(freq);
   nn_pmlin_bias_per(freq) = 100*(pmlin(1,freq) - ideal(freq))/ideal(freq);
   %
   pmhil_bias_per(freq) = 100*(mean(pmhil(2:f_end,freq)) - ideal(freq))/ideal(freq);
   pmcor_bias_per(freq) = 100*(mean(pmcor(2:f_end,freq)) - ideal(freq))/ideal(freq);
   pmlin_bias_per(freq) = 100*(mean(pmlin(2:f_end,freq)) - ideal(freq))/ideal(freq);
   %
   %
   pmhil_relvar(freq) = 100*std(pmhil(2:f_end,freq))/mean(pmhil(2:f_end,freq));
   pmcor_relvar(freq) = 100*std(pmcor(2:f_end,freq))/mean(pmcor(2:f_end,freq));
   pmlin_relvar(freq) = 100*std(pmlin(2:f_end,freq))/mean(pmlin(2:f_end,freq));
   %
end
%
%
axis([freq_list(1) freq_list(14) -.75 .75]);
plot(freq_list,nn_pmhil_bias_per,'+',freq_list,nn_pmcor_bias_per,'*',freq_list,nn_pmlin_bias_
```

```
  per,'o')
title('No noise % error, 10Khz PM mod., hilb = "+", corr = "*", sysest = "o"')
xlabel(' Frequency in Hz ')
ylabel(' Percentage error in measurement')
grid
pause
%
%
axis([freq_list(1) freq_list(14) -100 100]);
plot(freq_list,pmhil_bias_per,'+',freq_list,pmcor_bias_per,'*',freq_list,pmlin_bias_per,'o')
title('Rel. (%) error, 10Khz PM mod.  hilb = "+", corr = "*", sysest = "o"')
xlabel(' Frequency in Hz ')
ylabel(' Percentage error in measurement')
grid
pause
%
axis([snr_list(14) snr_list(1) -100 100]);
plot(snr_list,pmhil_bias_per,'+',snr_list,pmcor_bias_per,'*',snr_list,pmlin_bias_per,'o')
title('Rel. % error, 10Khz PM mod.  hilb = "+", corr = "*", sysest = "o"')
xlabel(' Signal to noise ratio in dB ')
ylabel(' Percentage error in measurement')
grid
pause
%
axis([snr_list(14) snr_list(1) -10 10]);
plot(snr_list,pmhil_bias_per,'+',snr_list,pmcor_bias_per,'*',snr_list,pmlin_bias_per,'o')
title('Rel. (%) error, 10Khz PM mod. hilb = "+", corr = "*", sysest = "o"')
xlabel(' Signal to noise ratio in dB ')
ylabel(' Percentage error in measurement')
grid
pause
%
axis([freq_list(1) freq_list(14) -100 100]);
plot(freq_list,pmhil_relvar,'+',freq_list,pmcor_relvar,'*',freq_list,pmlin_relvar,'o')
title('Coeff. of var. 10Khz PM mod. hilb = "+"  corr = "*"  sysest = "o"')
xlabel(' Frequency in Hz ')
ylabel(' Percentage variation in measurement')
grid
pause
%
axis([snr_list(14) snr_list(1) 0 100]);
plot(snr_list,pmhil_relvar,'+',snr_list,pmcor_relvar,'*',snr_list,pmlin_relvar,'o')
title('Coeff. of var. 10Khz PM mod.  hilb = "+"  corr = "*"  sysest = "o"')
xlabel(' Signal to noise ratio in dB ')
ylabel(' Percentage variation in measurement')
grid
```

```
pause
%
axis([snr_list(14) snr_list(1) -10 10]);
plot(snr_list,pmhil_relvar,'+',snr_list,pmcor_relvar,'*',snr_list,pmlin_relvar,'o')
title('Coeff. of var. 10Khz PM mod.  hilb = "+"  corr = "*"  sysest = "o"')
xlabel(' Signal to noise ratio in dB ')
ylabel(' Percentage variation in measurement')
grid
pause
%
clear
```

REFERENCES

[1] Vifian, H., Internal Seminar Paper, Hewlett-Packard Co., Santa Rosa, CA., March 1982.

[2] Van Trees, H.L., Detection, Estimation and Modulation Theory Part II, John Wiley and Sons, New York, p32, 1971.

[3] Papoulis, A., Probability, Random Variables, and Stochastic Processes, McGraw-Hill, New York, p178, 1984.

[4] Meyer, R. et al, "Cross Modulation and Intermodulation in Amplifiers at High Frequencies", *IEEE Journal of Solid State Circuits*, Vol 7, pp16-23, February 1972.

[5] Oppenheim, A.V. and Schafer, R.W., Digital Signal Processing, Prentice-Hall, Englewood Cliffs, N.J., p29, 1975.

[6] Jackson, L.B., Digital Filters and Signal Processing, Kluwer Academic Publishers, Boston, MA, p116, 1986.

[7] Bellanger, M.G., Digital Signal Processing of Signals, John Wiley and Sons, New York, p247, 1984

[8] Stark, H. and Tuteur, F.B.,Modern Electrical Communication, Prentice-Hall, Englewood Cliffs, N.J., p71, 1979.

[9] Srinath, M.D. and Rajasekaran, P.K. An Introduction to Statistical Signal Processing with Applications, John Wiley and Sons, New York, 1979.

[10] Bahr, Randall, Internal Memo, University of Arizona, Tucson, AZ, 1990.

[11] Press, W.H. et al, Numerical Recipes in C, Cambridge University Press, Cambridge, p114, 1988.

[12] Fante, R.L., Signal Analysis and Estimation, John Wiley and Sons, New York, p158, 1988.

[13] Orfanidis, S.J., Optimum Signal Processing, McGraw-Hill, New York, 1988.

[14] Zeidler, J.R., " Adaptive Enhancement of Multiple Sinusoids in Uncorrelated Noise", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, V 26, pp240-254, June 1978.

[15] Mulgrew, B and Cowan ,C. Adaptive Filters and Equalisers, Kluwer Academic Publishers, Boston, MA, 1988.

[16] Bellanger, M.G., Adaptive Digital Filters and Signal Analysis, Marcel Dekker, Inc., New York, 1987.

[17] Billings, S.A., " Identification of Nonlinear Systems- A Survey", *IEE Proceedings*, Vol. 127, Part D, No. 6, November 1980.

[18] Coker, M.J. and Simkins, D.N., " A Nonlinear Adaptive Noise Canceler", *Proceedings of 1980 International Conference on Acoustics, Speech and Signal Processing*, pp470-473, 1980.

[19] Schetzen, M., The Volterra and Wiener Theories of Nonlinear Systems, John Wiley and Sons, New York, 1980.

[20] Rugh, W.J., Nonlinear System Theory, Johns Hopkins University Press, Baltimore, Maryland, 1981.

[21] Spiegel, M.R., Mathematical Handbook of Formulas and Tables, McGraw-Hill, New York, 1968