EFFECTIVE BITS OF AN ANALOG-TO-DIGITAL CONVERTER:
Analysis of Sinewave Curve Fitting Algorithms

KAMINI SHENOI

B.E., Bangalore University, India, 1985

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

Approved by:

Major Professor

TABLE OF CONTENTS

TABLE OF CONTENTS (contd.)

TABLE OF CONTENTS (contd.)

LIST OF FIGURES

i

LIST OF TABLES

ii

ACKNOWLEDGEMENT

CHAPTER 1

INTRODUCTION

The present work deals with the evaluation of several algorithms for sinewave parameter estimation, and the subsequent effective bit computations for Analog-to-Digital Converters (ADCs). Three well-known algorithms are investigated; they include the Closed Form Approximation algorithm, the Three Parameter Fit algorithm (with known frequency), and the Four Parameter Least Squared Fit algorithm (with unknown frequency). These three algorithms form part of the larger sinewave curve fit technique employed in dynamic testing of ADCs. The sinewave curve fit test gives an overall indication of the quality of an ADC under test in terms of a figure of merit – the effective number of bits of the ADC.

In this work, the ADC under consideration as well as the reference data used in testing are simulated in software. The reference data includes pure sinewaves and sinewaves corrupted by the addition of calculated amounts of noise or higher order harmonics. The degradation in the number of effective bits for the simulated ADC is observed for each test input.

1

Following these introductory remarks, we introduce in Chapter 2, the basic concepts involved in the dynamic testing of an ADC. The different dynamic test methods available and the requirements of the input test signal are presented.

Chapter 3 focuses on the sinewave curve fit technique. This technique is described in detail along with the errors it measures and its advantages and disadvantages. The notion of effective bits as a figure of merit for an ADC is also defined.

The prime focus of this report is to investigate the available sinewave curve fit algorithms. In Chapter 4, the Closed Form Approximation algorithm and the Three Parameter Fit algorithm are presented. Their performance is evaluated and their relative merits and demerits are listed.

Chapter 5 deals entirely with the Four Parameter Least Squared Fit algorithm. The standard algorithm is first described, methods for obtaining the initial parameter estimates are highlighted, and the algorithm is subsequently analyzed. Problems encountered with the algorithm are discussed and modifications made to the standard algorithm to enhance its performance are detailed.

The generation of simulated reference data for use in testing is presented in Chapter 6. This is followed by an

analysis of the results obtained for the various simulated data sets.

The main computer program and significant functions, written in the C programming language, implementing the major algorithms are described in Chapter 7.

Chapter 8 lists our conclusions and recommendations for future work.

.

CHAPTER 2

DYNAMIC TESTING OF ANALOG TO DIGITAL CONVERTERS

Dynamic testing refers to the performance evaluation of an ADC when it is subjected to time-varying inputs. This type of testing is essential because there often exists a wide discrepancy between the static and dynamic performance of an ADC. It may be caused by a variety of reasons, prime among which are the response time, settling time and slew-rate limitations of the active devices of the circuit, as well as the ADC aperture jitter. All these limitations are taken into account by dynamic testing, which is therefore far more stringent than static testing and consequently gives a better indication of the converter performance in its intended application. Dynamic performance characteristics may also vary with changes in the sampling frequency, signal frequency and amplitude. If the entire frequency range over which the ADC is operated is tested dynamically, then, an accurate picture of the quality of the ADC may be obtained.

2.1 TEST METHODS

Currently four methods of dynamic testing are widely used. They include the histogram test, the beat frequency test, the FFT test and the sinewave curve fit test. Each of these tests provides information on some of the performance specifications of the ADC. In this report, we thoroughly investigate the sinewave curve fit test and use it to determine the effective bits of the ADC under consideration.

2.2 INPUT TEST SIGNAL

The input test signal normally employed for most dynamic test measurements is a full-scale sinewave. Sinewaves have a number of distinct advantages not offered by other waveforms. They are easy to generate at any frequency of interest. They are also highly accurate, consistent and readily available. In addition, they are reproducible. Using any readily available, high quality sinewave generator, we can duplicate and verify test results as many times as desired. It should be noted however, that the above statement holds good only for testing ADCs with a small number of bits (<16). It is not

5

easy to get sinewaves pure enough to test ADCs with 16 or more bits.

Another major reason why sinewaves are preferred is because they are easy to analyze mathematically and their characteristics, especially the important slew rate characteristic, are well known. This greatly simplifies the algorithms used for data analysis [12]. Sinewaves are also bidirectional. This means that a sinewave test signal will accurately identify ADC problems that depend upon the slope of the input signal [5].

2.2.1   Amplitude and Frequency Requirements

For successful dynamic testing using the sinewave curve fitting method, certain guidelines should be followed with regard to the frequency and amplitude of the input test signal.

Amplitude

It is very important to use full scale sinewaves as input test signals. In some cases, due to additive noise, the sinewave amplitude may have to be slightly below the full scale value in order to avoid clipping at peak values which causes harmonic distortion.

6

Signals which are less than full scale are liable to produce misleading results due to the following reasons [5]:

If a half amplitude sinewave is used, then only half the possible codes are tested. Specifying performance relative to full scale implies that the behavior of the half of the codes not tested is identical to that of the tested half, an assumption that is obviously not true. If the input test signal is full scale, the complete dynamic range and all of the codes of the ADC are exercised.

Full scale testing is far more stringent than half scale testing because errors due to aperture jitter, dynamic non-linearity, noise, slewing and settling effects all increase as the input signal level is increased. In fact, a full scale sinewave has twice the slew rate of a half scale sinewave at the same frequency.

Non linear effects also increase out of proportion with the signal amplitude, allowing some failures to be observed only with a full scale input.

Frequency

The frequency should be chosen such that it ensures testing of as many of the digitizer codes as possible. It should also be similar to the intended application bandwidth.

2.2.2 Purity of Source

Successful dynamic testing relies heavily on the purity of the sinewave source [11]. Commonly available synthesized sources can be used to provide the short and long term stability needed for the dynamic range of the ADC. However, passive filtering may be required to eliminate the harmonic distortion from the source.

CHAPTER 3

SINEWAVE CURVE FITTING AND EFFECTIVE BITS

Sinewave curve fitting is a relatively easy method used in obtaining a global description of the dynamic performance of an ADC. By global, it means that all the errors measured by the test are averaged to give a general measurement of the ADC transfer function [12]. The total noise from all sources is evaluated giving rise to the popular figure of merit of an ADC, namely the *effective bits*.

The first section of this chapter discusses what is meant by the term *effective bits* of an ADC, and its significance in dynamic testing. The sinevave curve fit test is then described in detail. Errors measured and not measured by this test are discussed, followed by its advantages and disadvantages.

3.1 EFFECTIVE BITS

The number of effective bits is used as a figure of merit in evaluating the dynamic performance of an analog to

9

digital converter. The effective bit characterization is generally used to describe ADC errors for measurements of repetitive signals such as are used in the field of communication.

Definition

The effective bits of an ADC may be defined as the number of bits in a perfect ADC whose rms quantization error is equal to the total rms error of the practical converter under test. The following formula may be used in the calculation of the effective bits:

$$\text{Effective bits} = N - \log_2 \frac{\text{actual rms error}}{\text{ideal rms error}}$$

where N is the number of bits of the perfect ADC. The rms error (actual) is calculated from the sinewave curve fit test.

For an ideal ADC, the rms error is just the quantization error inherent in all converters due to their finite discrete resolution when converting a continuous analog signal into digital form. This however, is not the case for a practical ADC. In addition to the quantization noise, the rms error of a practical ADC includes other sources of noise such as amplifier distortion, differential nonlinearity, integral nonlinearity, missing codes, noise

and aperture uncertainty, which are often dependent on the amplitude and frequency of the input test signal.

Degradation in the number of effective bits due to the presence of noise comes from the noise sources listed below [2].

   1.   Analog system distortion.
   2.   Analog system white noise.
   3.   Quantization noise.
   4.   Sample clock jitter.
   5.   Analog system correlated noise.
   6.   Source noise floor.
   7.   Source phase noise.
   8.   Source distortion.

The best way to identify the different noise components is to analyze the digital data record in the frequency domain using the discrete Fourier transform (DFT).

The effective bits may also be defined in terms of the signal to noise ratio of the system.

$$\text{effective bits} \;=\; \log_2(\text{SNR}) \,-\, 0.5\log_2(1.5)$$

$$-\, \log_2(A \,/\, V)$$

where

$$\text{SNR} \;=\; \frac{\text{rms signal}}{\text{rms noise}}$$

is the signal to noise ratio of the system, A is the amplitude of the input sinewave and V is the full scale range of the converter input.

3.2 SINEWAVE CURVE FIT TEST

A sinewave generator is used to generate a spectrally pure sinewave which is fed to the ADC under test. The corresponding digitized data record is obtained. A high purity sinewave is then generated in software such that it is the best fit sinewave (with respect to the mean squared error) to the digitized data record. The difference between the actual ADC data and the best fit data is determined and the rms error (actual) is calculated. This is nothing but the standard deviation of the error.

Next, an idealized digital record of the input is generated by a perfect ADC (simulated in software), having the same number of bits as the actual ADC. The best fit sinewave to this idealized digital record is obtained. The rms error (ideal) is calculated from the difference between these two data records.

Since we know that the rms error of a perfect ADC is due only to its quantization error, we don't actually go through the process of finding the best fit sinewave to the

idealized digital record and thereby calculating the rms error. Instead, we assume the ideal rms error to be equal to the theoretical rms quantization error of a perfect N-bit ADC.

The theoretical quantization error as a function of input voltage is a triangular waveform with an amplitude equal to half an lsb. Using this definition, we calculate the ideal rms error as,

$$\text{rms ideal} = q / \sqrt{12}$$

$$q = V_{fs} / 2^N$$

where q is the width of an ideal code bin, V_{fs} is the full scale voltage and N is the number of bits of the ADC. Then, using the formula given previously, the effective bits of the ADC may be calculated.

3.3 ERRORS MEASURED

Errors such as random noise, quantization error, differential nonlinearity, missing codes, integral nonlinearity, harmonic distortion, aperture uncertainty and spurious response are measured by this test.

By varying the test conditions, some knowledge can be gained about the different error components present in the

13

system [12]. White noise produces the same degradation regardless of the input frequency or amplitude. Therefore varying the test conditions has no effect on this noise. Noise can also be identified by observing the difference between the best fit sinewave and the input data taken.

Aperture error generates an error that is a function of the input slew rate. It causes the number of effective bits to vary linearly with both the input frequency and amplitude of the waveform. If the input waveform is sampled only at points of constant slew rate, such as at zero crossings, then the aperture uncertainty corresponds to the amount that the effective bits decline as a function of slew rate.

Harmonic distortion which introduces harmonics in the error residuals and in the Fourier transform is the result of nonlinearities. By fitting the error residue with the best fit sinewaves, the harmonic amplitudes can be obtained.

A qualitative feel for the errors may be obtained by examining a plot of the residuals and of the Fourier transform of the digitized sinewave [14]. Errors concentrated near the zero crossings are probably due to aperture uncertainty, while errors concentrated near the peaks of the sinewave are probably due to nonlinearities.

14

3.4 ADVANTAGES AND DISADVANTAGES

The main advantage of the sinewave curve fit test is
that it is relatively easy to execute and does not require
any sophisticated equipment. It is a global test in that
it gives an overall description of the dynamic performance
of the ADC, and also gives insight into the distribution of
the digitization error which is essential for the design of
a digital filtering procedure [12]. The various errors
measured are described above.

However, the test does not measure amplitude flatness,
phase linearity or any long term variations. Gain, offset
and bandwidth errors are not measured either.

CHAPTER 4

ALGORITHMS FOR SINEWAVE PARAMETER ESTIMATION

Generation of the best fit sinusoid to a digital data record requires an estimation of the four sinewave parameters, namely the amplitude, offset, frequency and phase. Several fitting routines, both closed form and iterative exist for the calculation of these parameters.

In this report, we investigate the performance of three of these fitting routines. They include the closed form algorithm developed by Jenq and Crosby [7], the three parameter (known frequency) sinewave curve fit algorithm [14] and the four parameter (general case) sinewave curve fit algorithm developed by Peetz [12].

Of the three methods, only the three and four parameter methods are extensively tested, their shortcomings listed and modifications for improved operation suggested. The closed form approximation method is briefly discussed and evaluated in this chapter.

4.1 CLOSED FORM APPROXIMATION ALGORITHM

The algorithm developed by Jenq and Crosby [7] results in closed form approximations for the four parameters of the best fit sinusoid to a digital data record. A detailed description may be found in [7]. This section briefly examines the salient features of the algorithm in order to determine its usefulness.

4.1.1 Description of Algorithm

We start with a data record consisting of N samples assumed to be from a sinewave source of the form

$$S = \{s_k = s(k \mathbin{/} f_s) = A \sin(2\Pi f k / f_s + \theta) + DC\}$$

where $k = 0, 1, .., N-1$, and f_s is the sampling rate. From a knowledge of these samples we want to estimate the four parameters – amplitude, offset, frequency and phase of the sinewave.

The dc offset and the amplitude are determined first. For this the 4-term Blackman-Harris window is sampled to

17

obtain a window sequence of the form:

$$w_k = 0.35875 - 0.48829\cos(k2\pi/N) +$$

$$0.14128\cos(k4\pi/N) - 0.01168\cos(k6\pi/N),$$

where $k = 0, \ldots, N-1$.

Then the dc offset estimate is given by:

$$\hat{D} = \frac{\sum_{k=0}^{N-1} w_k s_k}{\sum_{k=0}^{N-1} w_k}$$

and the amplitude estimate is given by:

$$\hat{A} = \left[2\; \frac{\sum_{k=0}^{N-1} w_k (s_k - \hat{D})^2}{\sum_{k=0}^{N-1} w_k} \right]$$

Determination of the frequency and phase require some additional computations. We define

$$\hat{s}_k = (s_k - \hat{D}) / \hat{A}$$

$$u_k = 1 - (\hat{s}_k)^2$$

$$x_k = \sin^{-1}(\hat{s}_k), \text{ subject to monotonic condition,}$$

$$t_k = k / f_s$$

where $k = 0, \ldots, N-1$.

Using the above definitions, we set the following:

$$\text{SumU} = \Sigma u_k$$

$$\text{SumUT} = \Sigma u_k t_k$$

$$\text{SumUTT} = \Sigma u_k t_k t_k$$

18

$$\text{SumUX} = \Sigma u_k x_k$$

$$\text{SumUTX} = \Sigma u_k t_k x_k$$

$$\Delta = \text{SumU.SumUTT} - \text{SumUT.SumUT}$$

We now have the frequency and phase estimates as:

$$\hat{F} = \frac{1}{2\Pi} \frac{\text{SumU.SumUTX} - \text{SumUT.SumUX}}{\Delta}$$

$$\hat{\theta} = \frac{\text{SumUX.SumUTT} - \text{SumUt.SumUTX}}{\Delta}$$

Once the four parameters of the input sinewave are determined, the estimated sinewave is used as if it were the actual input sinewave to compute the effective bits of the digitizer using the formula:

$$\text{Effective Bits} = \log_2 \left(\frac{\text{Full Scale}}{\sqrt{12} \ \text{RMSE}} + 1 \right)$$

where Full Scale is the full scale of the digitizer used, and RMSE is the root mean square error of the digitized signal.

4.1.2 Evaluation of Algorithm

Although the method outlined above appears to be both simple and efficient, all is not as it seems. Calculation of the amplitude and offset are found to be fairly straightforward with fairly accurate results being obtained in all cases. For the frequency and phase estimates, some

19

additional terms need to be defined, one among which is the element

$$x_k = \sin^{-1}(\hat{s}_k), \quad k = 0,\ldots N-1$$

where x_k is in radians.

It is imperative that x_k increase monotonically if one wants accurate (or even reasonably accurate) estimates of frequency and phase. Herein lies the problem. Figure 4.1 shows a plot of the arcsine function. As can be seen from the figure, the arcsine is defined only between $\pm\ \pi/2$. To decide where exactly on the argument axis to put the result for x_k at each step in order to preserve the required monotonicity, we have to know not only the slope of the sinewave at the current data point but also the slopes at the previous and future data points, i.e., we need to know where exactly on the unknown sinewave the data point lies. Without this knowledge, there is no way of telling what the correct argument value is. For a well known sinewave with at least seven or eight data points per cycle, this is not a problem, but the addition of even a little noise would cause severe complications. Besides, we would like to use this algorithm for the estimation of the parameters of a totally unknown sinewave which may even have noise or some other distortions.

Figure 4.1. Plot of arcsin(k) function.

21
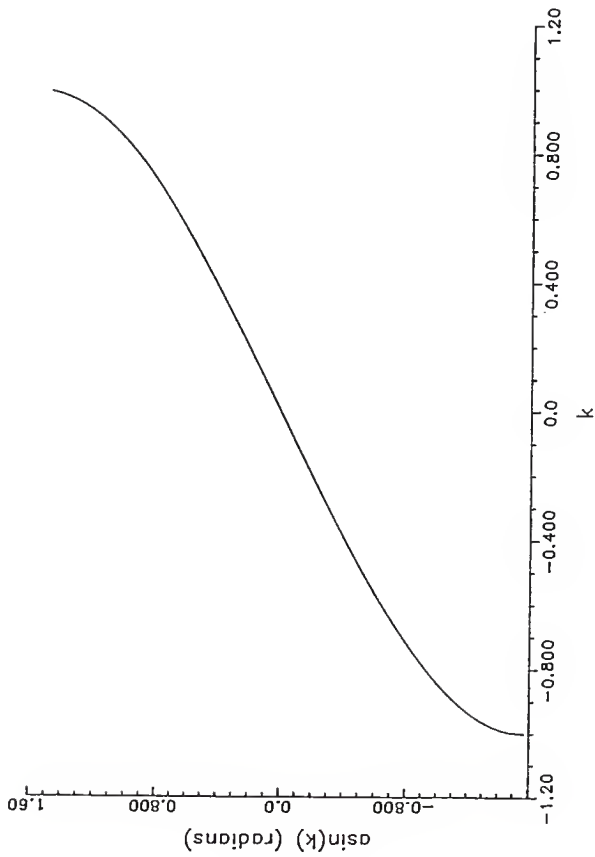
The problem detailed above was encountered after only preliminary testing of the algorithm. Since further work was not carried out, it is not known at this point whether a solution to this problem would be difficult to realize or not. Since then, a comprehensive analysis has been done and the results tabulated. Details of this study may be found in [4].

4.2 THREE PARAMETER ESTIMATION ALGORITHM

If the frequency of the input sinewave is accurately known, as is often the case, then the three parameter sinewave curve fit algorithm may be used. It is both convenient and quick and gives excellent results. The next section briefly highlights the salient features of the algorithm. Then, the method is evaluated and its advantages and disadvantages are discussed.

4.2.1 Description of Algorithm

A detailed description of the algorithm along with its derivation may be found in [14]. Basically, this is a closed form solution which can be used in the estimation of

22

three of the four parameters of the unknown sinewave. The
parameters estimated are the amplitude, offset and phase.
The frequency is assumed to be known.

The input to the algorithm consists of a data record
of N samples y_k taken at times t_k, where $k = 0,..,N-1$. The
samples are assumed to come from a sinusoidal source whose
output is sampled at a uniform rate.

We want to obtain a solution of the form,

$$Y_k' = A\cos(\omega t_k) + B\sin(\omega t_k) + C$$

where ω is the known angular input frequency and t_k are the
sample times. First the total residual errror, ϵ, of the
measured data relative to the fitted sinewave is
calculated. Once we have this, the rms error is readily
obtained from the following equation:

$$\epsilon_{rms} = [\epsilon / N]^{1/2}$$

Then using the formula for effective bits given in Section
3.2, the number of effective bits of the ADC under
consideration can be determined.

However, we are more interested in obtaining a
solution:

$$Y'_k = R\sin(\omega t_k + \theta) + C$$

since this is the form that we get using the other
algorithms studied. For this, we need to convert the

23

amplitude(R) and phase(θ) to the required form using,

$$R = [A^2 + B^2]^{1/2}$$

$$\theta = \tan^{-1}(-A/B)$$

Since the answer is obtained in closed form, convergence is always guaranteed.

4.2.2 Evaluation of Algorithm

The three parameter test algorithm was evaluated and the results analyzed. For testing we used a sinewave with an amplitude of 5 volts, an offset of 5 volts, and a sampling rate of 1.05E-4 secs. The frequency is 14.5 hz. The sampling rate is chosen such that it is non-harmonically related to the input frequency. These parameters have been chosen at random. This sinewave was digitized by a 16 bit ADC with a full scale range of 10 volts. The digital record obtained formed the input to the algorithm. The input phase was varied in steps of 30° from 0° upto 360°. At each step the values of the three parameters estimated, the rms error and the effective bits were tabulated (Tables 4.1 and 4.2).

Table 4.1. Summary of results for the three parameter algorithm.

INPUT PARAMETERS: Amplitude = 5.0 Volts   Offset = 5.0 Volts
                    Frequency = 14.5 Hz   Sampling Rate = 1.05E-4 secs

Input Phase (degrees)	Amplitude Estimate (Volts)	Offset Estimate (Volts)	Phase Estimate (degrees)
0	5.00003	4.99999	0
30	5.00000	5.00001	30
60	4.99999	4.99999	60
90	5.00003	5.00000	90
120	4.99999	4.99996	120
150	5.00000	5.00001	150
180	5.00003	4.99999	180
210	4.99999	4.99998	210
240	5.00001	4.99998	240
270	5.00000	4.99997	270
300	5.00001	4.99998	300
330	4.99999	4.99998	330
360	5.00003	4.99999	360

Table 4.2. Variation of effective bits with phase (Three parameter algorithm).

INPUT PARAMETERS: Amplitude = 5.0 Volts   Offset = 5.0 Volts
                    Frequency = 14.5 Hz   Sampling Rate = 1.05E-4 secs

Input Phase (degrees)	RMS Error	Effective Bits
0	0.000044	15.991459
30	0.000044	15.997597
60	0.000044	15.999408
90	0.000044	15.999249
120	0.000043	16.019889
150	0.000043	16.011923
180	0.000044	15.991487
210	0.000044	15.997627
240	0.000044	15.999382
270	0.000044	15.999296
300	0.000044	16.013802
330	0.000044	16.011917
360	0.000044	15.991604

26

For all values of the input phase, the effective bits obtained are found to be very accurate as may be seen from Figure 4.2. The amplitude and the offset of the sinewave are also estimated very accurately. We now turn our attention to the phase estimates obtained. It is seen that while the input phase varies from $0°$ upto $360°$, the estimated phase values do not exceed the range $\pm90°$. This is as expected because we are using the inverse tan function which yields results in the first and fourth quadrants only. To try to find some relation between the actual and the estimated values, and thereby devise some correction factor, we consider the equation for the phase. One interesting fact becomes evident. For all input phase values in the first quadrant, $(0° - 90°)$, both A and B have a positive sign. In the second quadrant, $(90° - 180°)$, A is positive while B is negative. In the third quadrant, $(180° - 270°)$, both A and B are negative, and in the fourth quadrant, $(270° - 360°)$, A is negative while B is positive.

Knowing the signs of A and B and the estimated phase value, the actual phase may be calculated as:

$$\Phi = \theta \qquad \rightarrow \quad \text{in the first quadrant}$$
$$\Phi = \Pi + \theta \quad \rightarrow \quad \text{in the second quadrant}$$
$$\Phi = \Pi + \theta \quad \rightarrow \quad \text{in the third quadrant}$$
$$\Phi = \theta \qquad \rightarrow \quad \text{in the fourth quadrant}$$

where Φ is the actual phase and θ is the phase estimate.

27
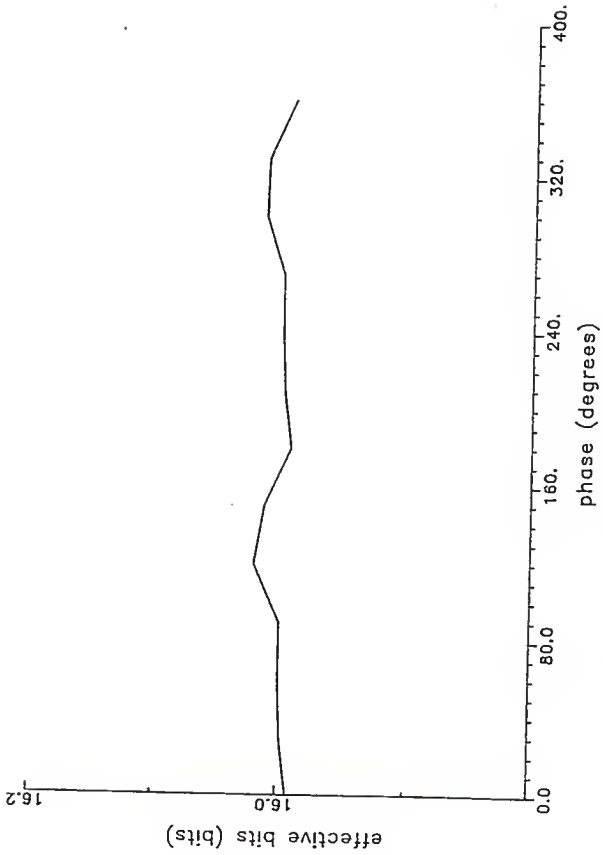
Figure 4.2. Variation of effective bits with phase.

<u>Example</u>

For an input phase value of 120°, the results obtained from the test are:

Estimated phase = − 60°,

A = +ve and B = −ve

This indicates that the angle lies in the second quadrant. Now, using the formula for the second quadrant given above, we get:

$$\Phi = \Pi + \Theta°$$
$$= \Pi - 60°$$
$$= 120°$$

which is what we want. A suitable modification is made to the algorithm to incorporate this phase correction factor.

It is seen therefore, that if the input frequency is accurately known, it would be best to use the three parameter method to determine the estimates of the sinewave parameters, and thereby the effective bits of the digitizer under consideration. The algorithm is both efficient and simple and very accurate results are obtained for all cases.

29

CHAPTER 5

FOUR PARAMETER LEAST SQUARED FIT ALGORITHM

The best-fit sinusoid to the digital record of a
sinewave using the standard four parameter least squared
fit technique for parameter estimation developed by Peetz
et al. [12] is presented. The problem statement is first
formulated, followed by a description of the method
employed. Different ways of selecting the initial
estimates of the parameters of the input data record are
then given. The method is evaluated and its shortcomings
are listed. Next, steps are taken to try to modify the the
standard fitting routine in order to remove some of the
ambiguities present. A technique for phase correction is
proposed. The requirements for proper operation of the
modified method are listed. Finally, the problems
encountered with the least squared fit technique in general
are discussed.

5.1 PROBLEM STATEMENT

Given a data record taken from a sinewave source, to generate a sinewave in software that is the best fit to the input data record.

INPUT

Data record containing N samples of amplitude y_k at times t_k, $k = 1, 2, \ldots, N$.

OUTPUT

Estimates of the amplitude, offset, frequency and phase of the best fit sinewave to the input data record. RMS error between the input sinewave and the best fit sinewave.

5.2 DESCRIPTION OF STANDARD ALGORITHM

The least squares curvefit principle for parameter estimation is employed. We start with a data record containing N samples of amplitude y_k at times t_k, ($k = 1$, $2, \ldots, N$). The data record is assumed to be taken from a sinewave source. This source may represent a high purity

31

sinewave or it may represent a sinewave contaminated by some form of noise or by the addition of harmonics to the fundamental.

The basic approach to the method is as follows [8].

1. An initial estimate of the four parameters is obtained from the input data. Several ways of making these initial estimates are listed in the next section.

2. Using a first order Taylor series expansion, the function is expanded about the initial estimates.

3. The least squares estimates of the required increments to the four parameters are calculated simultaneously.

4. The calculated increments are added to the initial estimates of the parameters.

5. Using the new parameter estimates, the least squares estimates of the new increments for the parameters are calculated.

6. These new increments are used to calculate the values of the new parameters.

7. Steps (5) and (6) are repeated until sufficiently accurate estimates of the true parameter values have been attained. A predetermined value of error magnitude serves as the stopping criterion.

Once the estimates of the four parameters have been calculated, the rms error between the input sinewave and the best fit sinewave is determined. This rms error is then used in the calculation of the effective bits of the ADC as per the formula given in Section 3.1.

5.3   INITIAL ESTIMATES

Initial estimates of the frequency and phase of the data record which are close to the true values are critical to the success of the algorithm. Some of the methods used in making these initial estimates are outlined below.

5.3.1 Frequency

The frequency is measured in radians per second.

1. Since the frequency of the sinewave generator used to obtain the data record is known, this frequency can be used as the initial estimate.

2. A DFT can be performed on either the full record or a portion of it to give the initial frequency estimate.

3. The initial estimate can also be calculated based on a knowledge of the times of the zero crossings of the data record.

5.3.2 Phase

The phase is measured in radians and is referred to the first point in the data record.

1. The linear least squares fit technique can be used to determine the phase estimate as the intercept of the relation between the zero-crossing times of the record and consecutive integer multiples of Π.

2. Given a frequency estimate, the phase estimate can be calculated in closed form using the three-parameter least squared fit technique [14].

3. Using the first two samples of the data record, a two point estimation of the phase can be made, given by,

$$\text{Phase} = (\text{sign}(y_2 - y_1)) \cos^{-1}((y_1 - C) / A)$$

where y_1 is the first data point of the record, y_2 is the second data point of the record, C is an estimate of the offset and A is an estimate of the amplitude of the data record [14].

$$\text{sign}(y_2 - y_1) = 1 \quad \text{for } y_2 > y_1$$
$$= -1 \quad \text{for } y_2 < y_1$$

5.3.3 Amplitude

1. If the digitizer is reasonably free from large random errors, the amplitude may be estimated as half the algebraic difference of the maximum and minimum values of the data record.

2. A histogram representing the probability density function of the digital codes may also be employed.

5.3.4 Offset

1. The offset may be obtained by taking one half of the algebraic sum of the maximum and minimum values of the data record.

2. Another method is to take the average value over an integer number of cycles.

5.4 EVALUATION OF ALGORITHM

The least squared fit method as proposed by Peetz et al. [12] was evaluated using both simulated as well as real-world data records. Sinewaves with different values of amplitude, offset, frequency and phase were used. In each case, three of the four parameters were held constant while the fourth was varied. The sampling rate was chosen

to be non-harmonically related to the input signal frequency.

Some interesting observations are brought to light. In practically all the trials, for the case of a pure sinewave, the amplitude, offset and frequency are estimated very accurately, with an error not exceeding about 1% of the actual value. However, although the phase estimate is found to have the correct magnitude, the signs are often reversed. This reversal of signs is quite arbitrary. It is seen that for data records with a phase value very close to 0 degrees or $n\Pi$, $(n = 0,...)$, the number of effective bits calculated by the algorithm is fairly close to the actual value. For an input phase other than those mentioned above, the number of effective bits computed by the algorithm bears no set relation to the expected number of bits. In some cases the correct number of bits is calculated. Tables 5.1 and 5.2 show the variation of the fitted parameters and effective bits, respectively, with phase.

For some of the cases in which the phase was incorrectly estimated by the algorithm, the rms error is found to be inordinately large, leading to a totally erroneous value of the effective bits of the ADC. However, no trend is discernible. A plot of the effective bits as a function of the input phase is presented in Figure 5.1.

Table 5.1. Summary of results for the standard four parameter algorithm.

INPUT PARAMETERS: Amplitude = 4.8 Volts   Offset = 4.8 Volts
                  Frequency = 14.5 Hz   Sampling Rate = 1.61E-4 secs

Input Phase (degrees)	Amplitude Est. (Volts)	Offset Est. (Volts)	Frequency Est. (Hz)	Phase Est. (degrees)
0	4.799965	4.799982	14.500001	0.0
5	4.809423	4.999941	14.500000	-4.99947
10	4.799940	4.999945	14.500000	-10.000026
15	4.799668	4.999943	14.500000	-15.000038
20	4.794892	4.999948	14.500000	-20.000002
25	4.799952	4.999945	14.499999	-25.000004
30	4.804231	4.999942	14.500000	-29.999997
35	4.799944	4.999942	14.500002	34.999943
40	4.799946	4.999945	14.500000	39.999989
45	4.809796	4.999949	14.499998	45.000001
50	4.799914	4.999946	14.500001	-50.000001
55	4.799973	4.999943	14.500000	54.999925
60	4.795913	4.999943	14.500003	59.999874
65	4.799960	4.799966	14.500000	65.000008
70	4.804895	4.999944	14.500000	70.000004
75	4.800239	4.799963	14.500000	74.999969
80	4.799951	4.999942	14.499999	80.000049
85	4.790391	4.799968	14.500000	85.000001
90	4.799947	4.999944	14.500000	90.000001

Table 5.2. Variation of effective bits with phase (Standard four parameter algorithm).

INPUT PARAMETERS: Amplitude = 4.8 Volts   Offset = 4.8 Volts
                   Frequency = 14.5Hz   Sampling Rate = 1.61E-4 secs

Input Phase (degrees)	RMS Error	Effective Bits
0	0.000094	16.001017
5	0.009064	8.315149
10	0.000045	15.973932
15	0.000271	13.378940
20	0.004823	9.225418
25	0.000044	15.999300
30	0.004032	9.483760
35	0.000044	16.001751
40	0.000044	16.011168
45	0.009524	8.243652
50	0.000053	15.721600
55	0.000051	15.788179
60	0.003837	9.555361
65	0.000044	15.990738
70	0.004710	9.259418
75	0.000264	13.417159
70	0.000045	15.976115
85	0.009107	8.308195
90	0.000044	16.003049
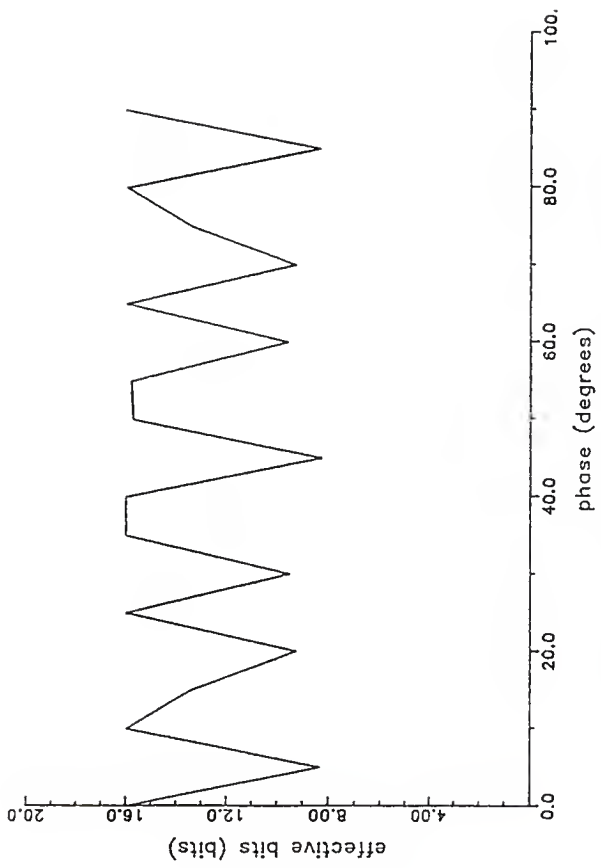
Figure 5.1. Variation of effective bits with phase.

39

From Figure 5.1, the arbitrary nature of the effective bit calculations is apparent.

The algorithm was tested several times with an input wave with some particular phase value. All tests gave the same results, and a clear trend was not discernible.

5.5 MODIFIED ALGORITHM

When using the standard least squared fit algorithm for parameter estimation, it has been observed that for input phase values very close to 0°, a fairly accurate phase estimate is obtained, whereas for input phase values further away from 0°, the phase values obtained may or may not be correct. This leads to unpleasant ambiguities in the effective bits calculations which need to be eliminated. In the next section, we develop a method to remove any phase present in the input data record thereby ensuring that the sinewave always starts with a phase of 0°. The basic least squared fit algorithm itself is not modified. The requirements for the correct operation of the modified fitting routine are also stated.

5.6   PHASE CORRECTION TECHNIQUE

     The technique employed for phase correction of the
input sinewave record is as follows. It is based on a
knowledge of the zero-crossing points of the unknown data
record and hence requires that the signal have no offset in
order to correctly determine these zero-crossings. A plot
of the input sinewave before phase correction is carried
out is shown in Figure 5.2. The first positive-going zero-
crossing point is determined. This forms the start of the
new data record. It is also useful in determining the
actual phase of the input signal before phase correction.
All successive points of the old data record now go to form
the new record, until the stopping criterion is met and the
program exits the loop. When a total of three positive-
going, zero-crossings have been detected, indicating that
two full periods of the input sinewave are present in the
new data record, then the stopping criterion is satisfied.

     We now have a new data record consisting of two full
periods of the same input sinewave as before, but with a
starting phase of 0° (Figure 5.3). Based on a knowledge of
the first point of the new data record and that of the
original record, we can calculate the actual phase of the
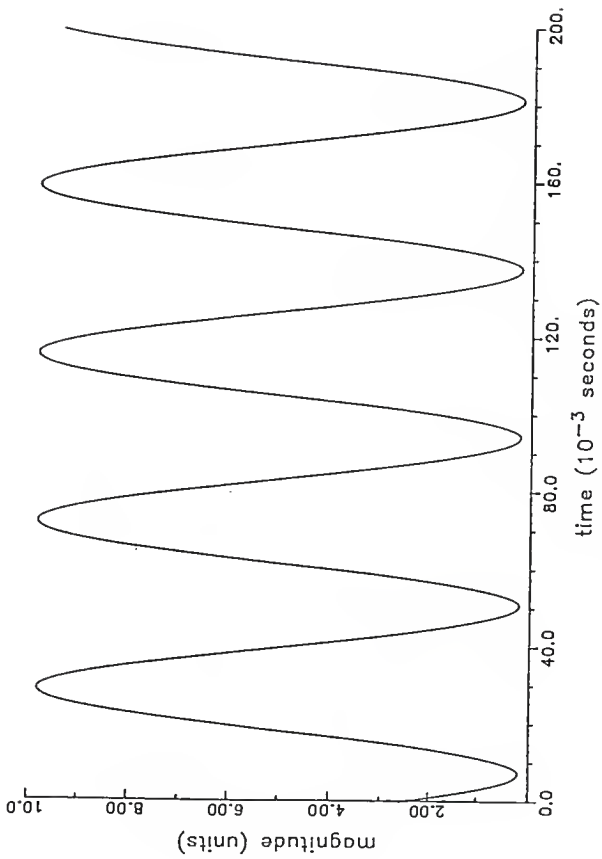
Figure 5.2. Input sinewave before phase correction.

42
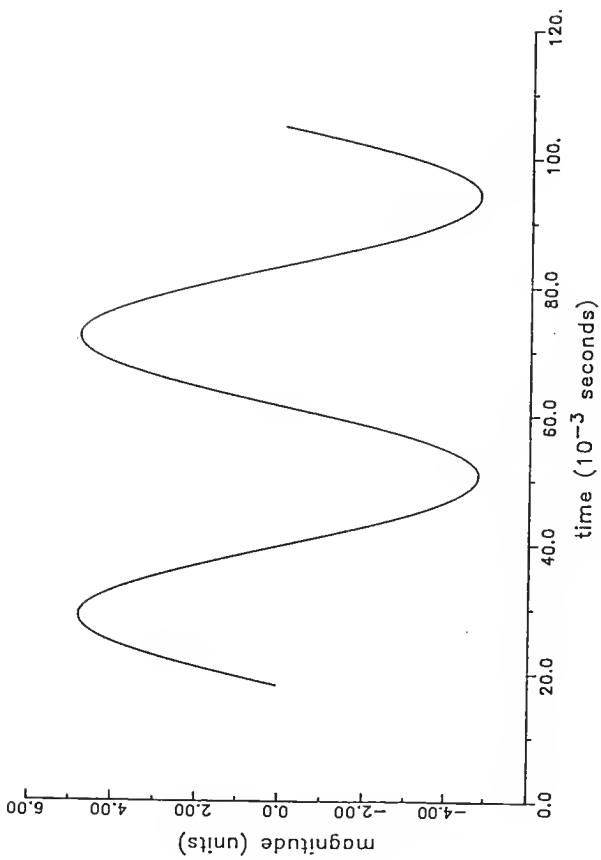
Figure 5.3. Sinewave after phase correction.

43

input sinewave as follows:

actual phase = 360 - (360 * crossover value /

points in one complete period)

This new data record has its starting point at some value of t, t > 0. We perform a time shift to bring this starting value to the origin at t = 0. Figure 5.4 shows the phase corrected sinewave after time shifting has been accomplished. At this point, the original dc offset which had been removed earlier, may be restored. The new data record now forms the input to the least squared fit algorithm for parameter estimation. Based on the rms error calculated in the algorithm, the effective bits of the digitizer may be determined.

5.7 REQUIREMENTS FOR PROPER OPERATION

To ensure correct operation of the modified fitting routine, regardless of whether simulated data or real-world data is used, certain requirements have to be met. Failure to adhere to these requirements may result in totally erroneous answers.

1. The data record should contain sufficient number of periods of the sinewave. The optimum number of periods is found to be equal to four or five.

44
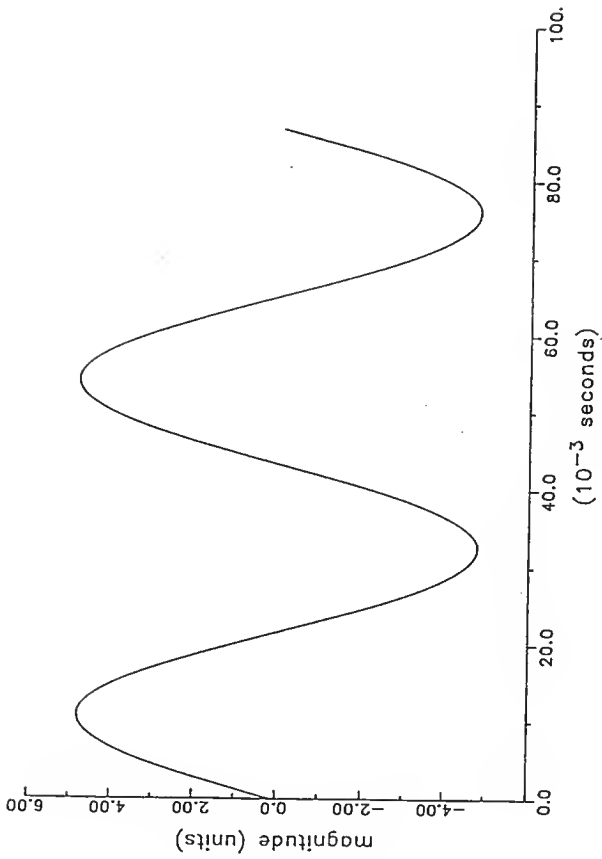
Figure 5.4. Phase corrected sinewave after time shifting.

45

2. The sampling rate should be chosen such that the interval between samples does not exceed roughly two degrees. This requirement is necessary to ensure proper phase correction of the recorded sinewave.

3. The amplitude of the input sinewave should be as close to the full-scale range of the digitizer as possible in order to exercise the complete dynamic range and all of the codes of the ADC. Full-scale testing is far more stringent than less than full-scale testing. This is because errors due to aperture jitter, dynamic nonlinearity, noise, slewing and settling effects can all increase as the input signal level is increased to full-scale [5]. Often, the amount of degradation in ADC performance is significantly more than the ratio in amplitudes. However, due to additive noise, the input sinewave may have to be slightly less than full-scale to avoid clipping at peak values which leads to harmonic distortion.

5.8 EVALUATION OF MODIFIED ALGORITHM

The modified method for parameter estimation and subsequent effective bits calculation has been tested under a variety of conditions. For our test purposes, we have considered an input sinewave having an amplitude of 4.8 volts, an offset of 5 volts, a frequency of 14.5 hz and a sampling rate of 2E-4 secs. These parameters may be varied as desired. A 16-bit ADC with a 10 volt full scale range was selected. The input phase was varied in steps of 5° from 0° to 90° and the output was observed. Table 5.3 shows the variation of the four estimated parameters and Table 5.4 shows the rms error and effective bits with changing values of input phase. Figure 5.5 is a plot of effective bits versus phase for the modified method.

From Table 5.3 it can be seen that for all values of input phase, the amplitude, offset and frequency are accurately estimated. This comes as no surprise because these three parameters were estimated correctly even in the standard method. Our interest lies in the values of the estimated phase.

Although the phase estimates are found to be not as accurate as those obtained using the three parameter test, they do lie significantly closer to the actual input phase values. The degree of accuracy of the estimation depends

47

Table 5.3. Summary of results for the modified four parameter algorithm.

INPUT PARAMETERS: Amplitude = 4.8 Volts Offset = 5.0 Volts
Frequency = 14.3 Hz Sampling Rate = 1.61E-4 secs

Input Phase (degrees)	Amplitude Est. (Volts)	Offset Est. (Volts)	Frequency Est. (Hz)	Phase Est. (degrees)
0	4.800100	5.000020	14.300010	0.0
5	4.800000	5.000024	14.300002	5.385501
10	4.799966	5.000021	14.299997	9.953917
15	4.799961	5.000021	14.300003	14.930876
20	4.799958	5.000018	14.299999	19.907834
25	4.799958	5.000022	14.300001	24.884793
30	4.799949	5.000019	14.299999	29.861751
35	4.799955	5.000022	14.299998	34.838710
40	4.799952	5.000020	14.299998	39.815668
45	4.799953	5.000023	14.300001	44.792627
50	4.799949	5.000024	14.299999	49.769585
55	4.800027	5.000022	14.299997	55.926352
60	4.800024	5.000021	14.299998	60.897583
65	4.800023	5.000020	14.299998	65.868815
70	4.800027	5.000019	14.300001	70.840046
75	4.800037	5.000022	14.300000	75.811277
80	4.800037	5.000021	14.300002	80.782509
85	4.800038	5.000020	14.300003	85.753740
90	4.800042	5.000024	14.299997	90.724971

Table 5.4. Variation of effective bits with phase (Modified four parameter algorithm).

INPUT PARAMETERS: Amplitude = 4.8 Volts   Offset = 5.0 Volts
                   Frequency = 14.3 Hz   Sampling Rate = 1.61E-4 secs

Input Phase (degrees)	RMS Error	Effective Bits
0	0.000065	15.452880
5	0.000072	15.299767
10	0.000071	15.585965
15	0.000061	15.535101
20	0.000062	15.514157
25	0.000062	15.501856
30	0.000063	15.473364
35	0.000066	15.426861
40	0.000065	15.430765
45	0.000067	15.403866
50	0.000068	15.382894
55	0.000043	16.022694
60	0.000044	16.011807
65	0.000044	15.995286
70	0.000045	15.971380
75	0.000046	15.939915
70	0.000046	15.947650
85	0.000046	15.947394
90	0.000047	15.918290
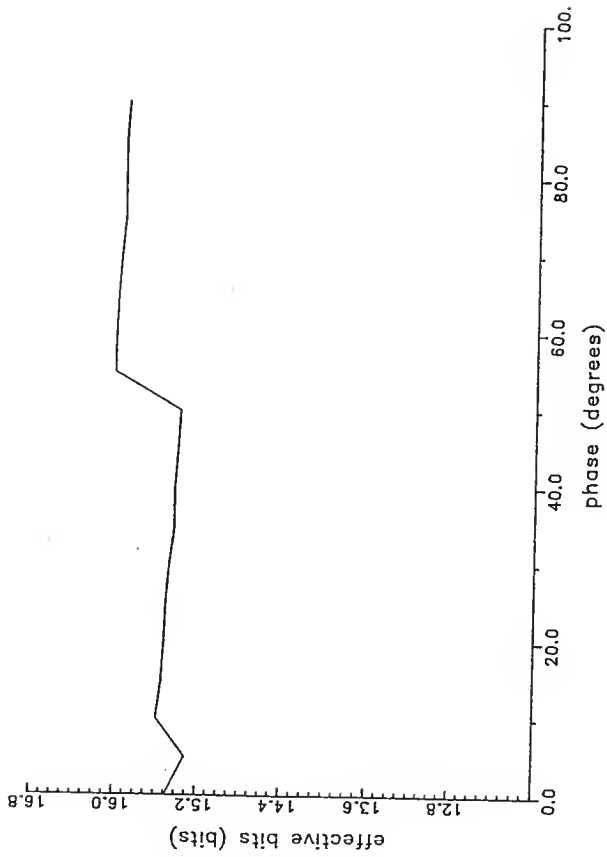
Figure 5.5. Variation of effective bits with phase.

50

to a great extent on the sampling rate chosen. For example, if sampling occurs at 0.7° intervals, then, the estimated phase value may lie anywhere within a 0.7° range of the actual value. This is illustrated in Figure 5.6.

We now turn our attention to Table 5.4 and Figure 5.5 which show the effective bits obtained as a function of the input phase. It is very interesting to note that now, for all input phase values, the number of effective bits calculated is very accurate, the difference between the maximum and minimum values calculated being not more than 0.1 bit. Thus the modified technique turns out to be a significant improvement over the standard method.

The modified fitting routine does have some major disadvantages, however. Chief among them is its extreme dependence on the sampling rate to get a good phase estimate. The requirement for a minimum data record size also turns out to be quite restrictive in many cases. While these are not unsurmountable problems, time and other constraints prevent us from doing anything about them for the moment.
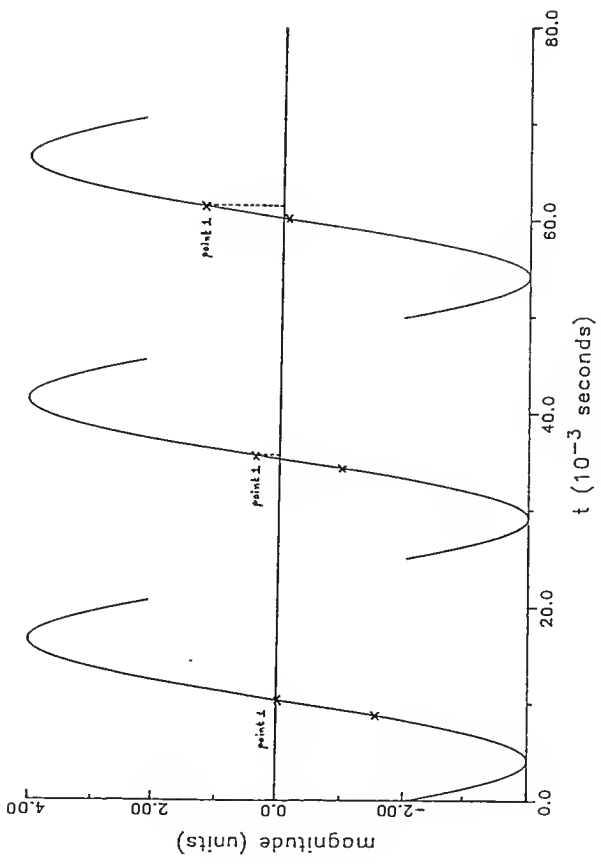
Figure 5.6. Error due to sampling interval.

5.9   PROBLEMS ENCOUNTERED WITH ALGORITHMS

    There are several problems common to both the original
as well as the modified method of parameter estimation
using the least squared fit technique.

    1. Being an iterative approach, convergence is not
        guaranteed. In fact the results from different
        runs   may not be consistent due to possible
        trapping at a local minimum [7]. This can become
        quite a problem when the data is poor or when
        there is inadequate computational resolution.

    2. If the initial frequency and phase estimates are
        not sufficiently close to the actual values, then,
        instead of converging, the program may diverge
        leading to totally erroneous results.

    3. If a very small step size is chosen in the
        iteration in the hope of always guaranteeing
        convergence, it   usually   takes   too long to
        converge.

    4. One of the greatest problems lies in the choice of
        the input frequency and the sampling frequency.
        Care should be taken to see that the sampling
        frequency is non-harmonically related to the input
        frequency.   If the sampling frequency is   an
        integral multiple of the input signal frequency,

53

then the same codes will be sampled over and over again [5]. If these happen to be good codes, then a falsely high value of the effective bits will be obtained, and vice versa for the case of bad codes being sampled. The worst situation occurs when the sampling frequency is exactly twice the signal frequency. When this happens, the quantization error is not measurable at all. Having non-harmonically related frequencies ensures that each record will have many different codes, thereby producing an rms error representative of the entire data record and not just of a few repeating codes. It also prevents certain harmonics from being aliased back onto the fundamental, thereby causing a higher signal-to-noise ratio and thus a higher effective number of bits.

CHAPTER 6

REFERENCE DATA USED IN TESTING

To evaluate the different algorithms developed for the
parameter estimation and subsequent effective bits
calculation using sinewave curve fitting, both simulated
data as well as real world data records are used. This
data may be pure or it may contain a known amount of
distortion. In this chapter we discuss the generation of
simulated data and then the results obtained from using
this data with the three parameter and the four parameter
estimation algorithms.

6.1 GENERATION OF SIMULATED DATA

The simulated data record may be of either a pure
sinewave or it may be of a sinewave contaminated by noise
or by the addition of higher order harmonics to the
fundamental. In this section, we discuss the generation of
all the three types of data records, followed by an
evaluation of the results obtained with the different data.

6.1.1 Pure Sinewave

A sinewave with the desired parameters is generated in software using the sine function of the computer. This sinewave is fed to an ideal N-bit ADC which has been simulated in software. Here it is quantized. The output of the ADC is then scaled to obtain the simulated sinewave data record. The error between the input sinewave and the digitized ADC output is plotted (Figure 6.1). A 16 bit ADC with a full scale range of 10 volts has been considered in this case. The error is seen to be within ± 1/2 lsb as is expected from an ideal 16 bit ADC.

6.1.2 Noisy Sinewave

Calculated amounts of rms noise may be added to a pure sinewave to produce a noisy sinewave. Standard Gaussian white noise is generated by the Box-Muller method using a random number generator. Scaling and shifting is then carried out to get noise having the desired mean and
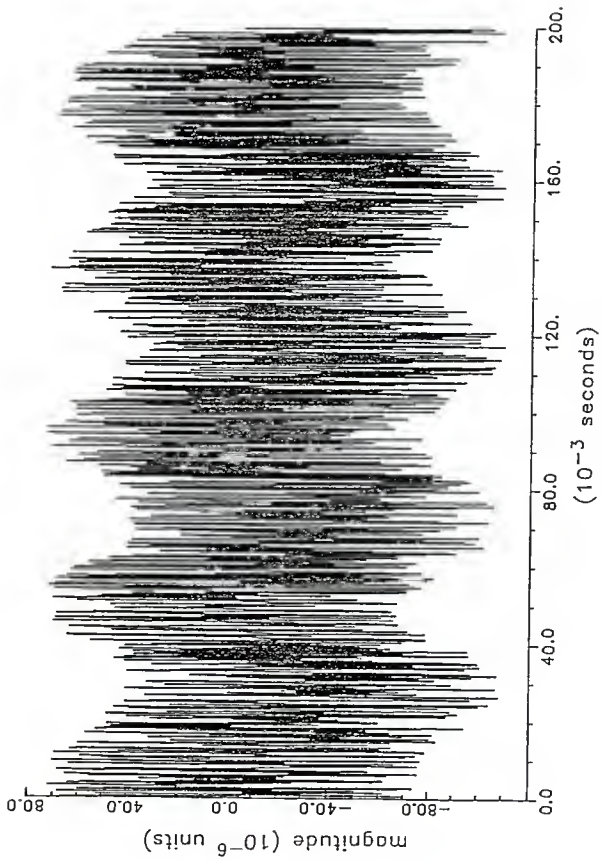
Figure 6.1. ADC error (16 – bit ADC).

57

variance. We get:

$$Y_i = x_i \sigma_y + \mu_y$$

where x_i is a standard Gaussian white noise sample, y_i is the scaled noise sample, μ_y and σ_y are the desired mean and variance respectively.

For a Gaussian distribution, the standard deviation of the distribution is equal to its rms value. Thus, we now have a noise file with a known rms value. This noise is added to the previously generated sinewave and the resultant file is fed to the ADC where it is digitized to obtain the required digital data record.

6.1.3 Addition of Higher Order Harmonics

Second and third order harmonics of desired amplitude with respect to the fundamental are generated. These are added in turn to the input sinewave (fundamental) and the resulting file is digitized by the ADC to produce the required digital data record.

6.2 ANALYSIS OF RESULTS

The different types of simulated data are used as the input for testing the three parameter fit and the four parameter least squared fit algorithms. An analysis of the results obtained for each case is given below.

6.2.1 Pure Sinewave

When a pure sinewave is used as the input to the algorithms, the results obtained are as expected. The parameters of the digitized sinewave are correctly estimated using both parameter estimation methods. The effective bits are also calculated without any problem.

6.2.2 Noisy Sinewave

The effect of pure sinewaves, contaminated by the addition of calculated amounts of Gaussian white noise, on the effective bits of an ADC have been studied. Files were calculated for rms noise amplitudes expressed as a percentage of the input sinewave amplitude, and these were added to the pure sinewave to produce the required noisy

sinewave. The rms noise amplitudes range from 20% down to 9.7×10^{-4}% of the input sinewave amplitude. In each case, the number of effective bits was observed as well as the accuracy of the fitted parameters (amplitude, offset, frequency and phase). Tables 6.1 and 6.2 show the variation of the rms error and the effective bits as a function of the rms noise amplitude for the four parameter method and the three parameter method, respectively.

Since the number of effective bits of an ADC is given by:

$$\text{Effective bits} = N - \log_2 \frac{\text{rms error (actual)}}{\text{rms error (ideal)}} ,$$

where N is the number of bits of a perfect ADC, and rms (ideal) is a constant for an ADC with fixed number of bits, we would expect the effective bits to be a function of the actual rms error.

The results are as expected. From Table 6.1 it is seen that an increase in the rms noise amplitude by a factor of 2 causes a reduction of one bit in the number of effective bits calculated for the ADC. A plot of the effective bits versus rms noise for the four parameter fit algorithm is presented in Figure 6.2. We see that for rms noise values from about 20% down to about 15.625×10^{-3}% of the input amplitude, a linear relationship is exhibited between the effective bits calculated and the rms noise

60

Table 6.1. Variation of effective bits with rms noise (Modified four parameter algorithm).

INPUT PARAMETERS: Amplitude = 3.0 Volts   Offset = 4.0 Volts   Phase = 0 degs
                    Frequency = 14.5 Hz   Sampling Rate = 2.00E-4 secs

RMS Noise (% Amplitude)	Effective Bits
8.000E00	4.174000
4.000E00	5.197674
2.000E00	6.197492
1.000E00	7.133416
5.000E-1	8.133458
2.500E-1	9.133884
1.250E-1	10.133936
6.250E-2	11.133418
3.125E-2	12.129516
1.563E-2	13.116469
7.813E-3	14.081300
3.907E-3	14.943130
1.953E-3	15.526022
9.765E-4	15.853354

Table 6.2. Variation of effective bits with rms noise (Three parameter algorithm).

INPUT PARAMETERS: Amplitude = 3.0 Volts   Offset = 4.0 Volts   Phase = 0 degs
                   Frequency = 14.5 Hz   Sampling Rate = 2.00E-4 secs

RMS Noise (% Amplitude)	Effective Bits
8.000E00	3.290571
4.000E00	4.342305
2.000E00	5.365775
1.000E00	6.376249
5.000E-1	7.379451
2.500E-1	8.375580
1.250E-1	9.374508
6.250E-2	10.372952
3.125E-2	11.375014
1.563E-2	·12.370763
7.813E-3	13.356964
3.907E-3	14.302838
1.953E-3	15.135251
9.765E-4	15.633709
4.883E-4	15.892986
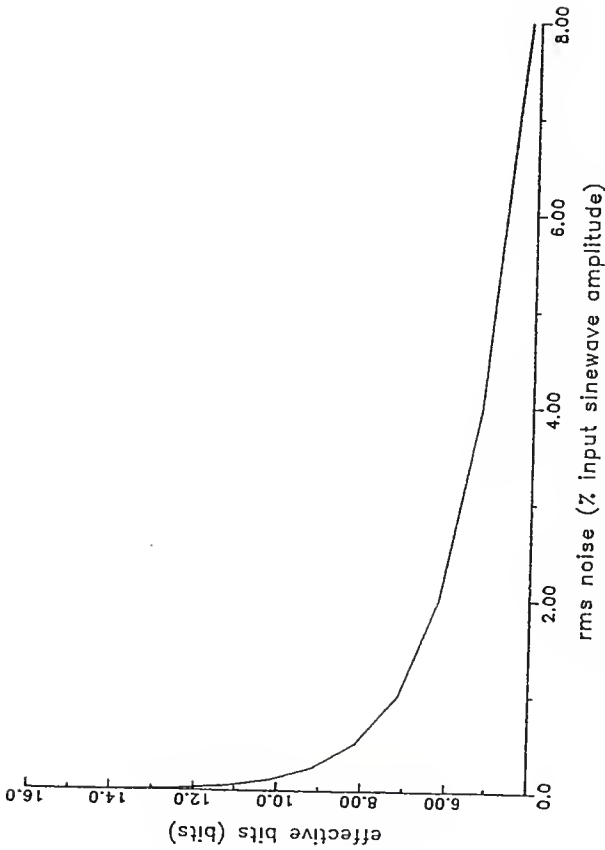
Figure 6.2.   Effective bits versus rms noise.

amplitude. For values below 15.625×10^{-3}% of the input amplitude, the fall-off in the number of effective bits is slower, until a point is reached where the noise amplitude is less than the resolution capability of the ADC, whereupon the noise ceases to affect the effective bit calculation. Essentially the same results are obtained with the three parameter fit algorithm.

The results obtained using the three parameter fit algorithm are similar to those obtained with the four parameter fit algorithm. Here too, the fall off in effective bits is directly proportional to the rms noise amplitude. A plot of the effective bits versus rms noise for the three parameter fit is presented in Figure 6.3.

The importance of this variation lies in the fact that, if a pure sinewave is corrupted by a calculated amount of noise, the corresponding number of bits of the ADC can be accurately determined. Any deviation from this value can only be due to the presence of non-linearities in the ADC under consideration. Hence the quality of the ADC can be tested.
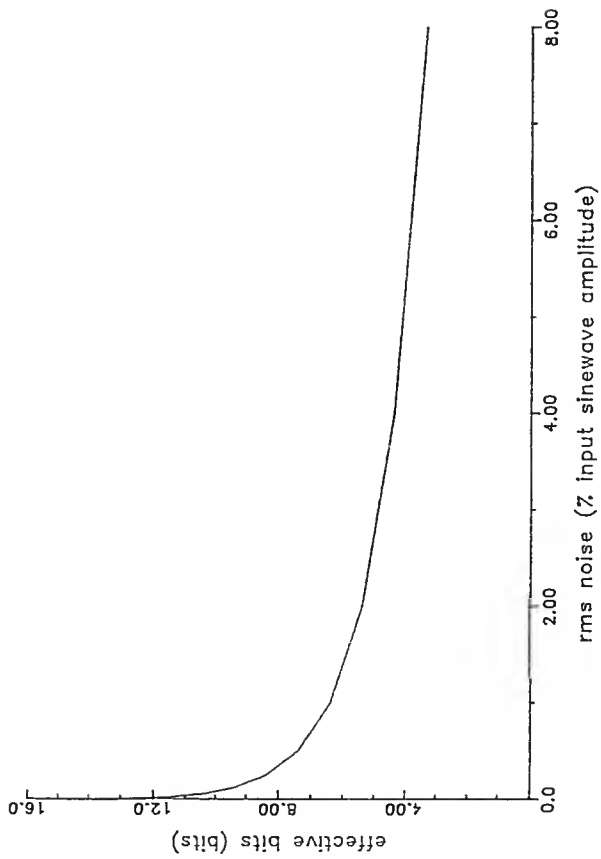
Figure 6.3. Effective bits versus rms noise.

65

6.2.3 Addition of Higher Order Harmonics

When a pure sinewave corrupted by the addition of
higher order harmonics forms the input to the parameter
estimation algorithms, a very significant reduction in the
number of effective bits takes place.   However at this
time, there does not seem to be any distinct relationship
between the reduction in effective bits and the amplitude
of the added harmonics.

CHAPTER 7

COMPUTER PROGRAMS

In this chapter, we describe the computer programs implementing the modified four parameter least squared fit algorithm developed in the present work. The main calling program for the modified algorithm is first described; this is followed by a description of each of the called functions.

7.1 MAIN PROGRAM

We start with an unknown data record consisting of N samples of amplitude y_k, k = 0,1,...,N-1, with known sampling rate. If simulated data is to be used, then a simulated data record has to be first generated using functions *sinewave()* and *digitizer()*. The sampling rate enables us to determine the x-axis time values through the

relation:

$$t_k = k * f_s$$

where f_s is the sampling rate in seconds, $k = 0,..N-1$, and t_k is the required x-axis value. Using the available data, the input unknown sinewave is plotted (Figure 7.1). It may be either a pure sinewave, a sinewave corrupted by a known amount of noise or a sinewave corrupted by the addition of higher-order harmonics.

Next an initial guess of the frequency is determined, which will be used as the initial frequency estimate of the least squared fit algorithm. Of the available options listed earlier for making an initial guess of the frequency, we have chosen to determine the zero-crossing points of the waveform and thereby calculate the frequency. This requires that any offset present be removed. Since this criterion is also required for the phase correction technique, it does not pose any additional labor.

To determine the offset, we have elected to use the method proposed by Jenq and Crosby [7]. This method requires a knowledge of the number of samples, the amplitudes y_k and the sampling rate, all of which we already have. Also, being a closed form solution, there's no question of a lack of convergence to the solution. By invoking the function *dc_offset( )* we are able to calculate
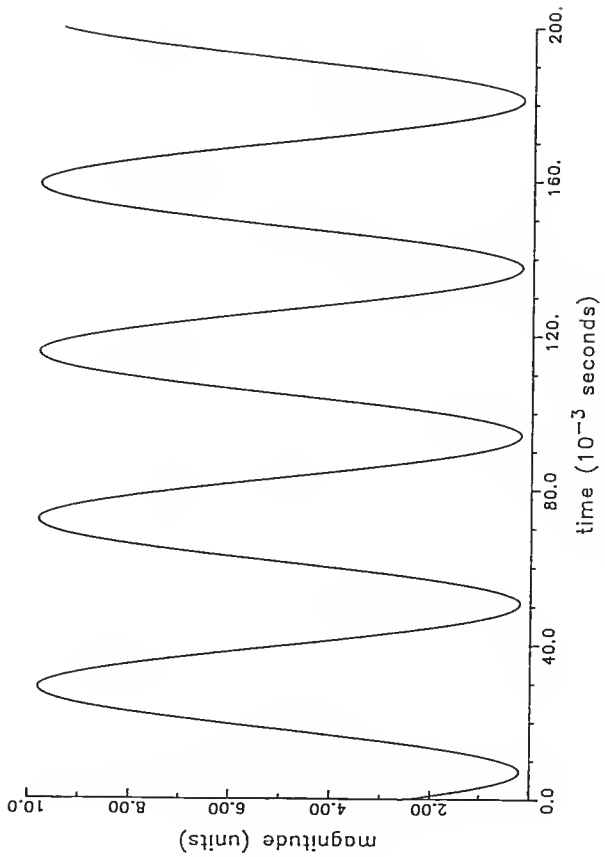
Figure 7.1. Input sinewave.

time (10^{-3} seconds)

magnitude (units)

69

the offset. Once the offset has been determined, it is removed from the input sinewave. The sinewave, now possessing no offset, is analyzed and two successive zero-crossing points are recorded. The interval between them gives the time of one complete period (T) of the sinewave, from which the frequency can be calculated, since $f = 1/T$.

The input now consists of a sinewave with no offset and with unknown starting phase. The next step is to remove this phase and get a wave with 0° starting phase. This is accomplished by invoking the phase correction function, *corphase()*. The input waveform after phase correction is plotted in Figure 7.2. We thus have a new data record consisting of two full periods of the original sinewave but with 0° starting phase. The dc offset removed earlier may now be restored if needed.

Time shifting is performed to enable the new data record to start from $t = 0$. This is shown in Figure 7.3. The phase corrected, time shifted data record now forms the input to the function *least_squared_fit()*, where its parameters are estimated. The output of this function consists of estimates of the four parameter of the best-fit sinewave to the unknown sinewave of the input data record. The four parameters are the amplitude, offset, frequency
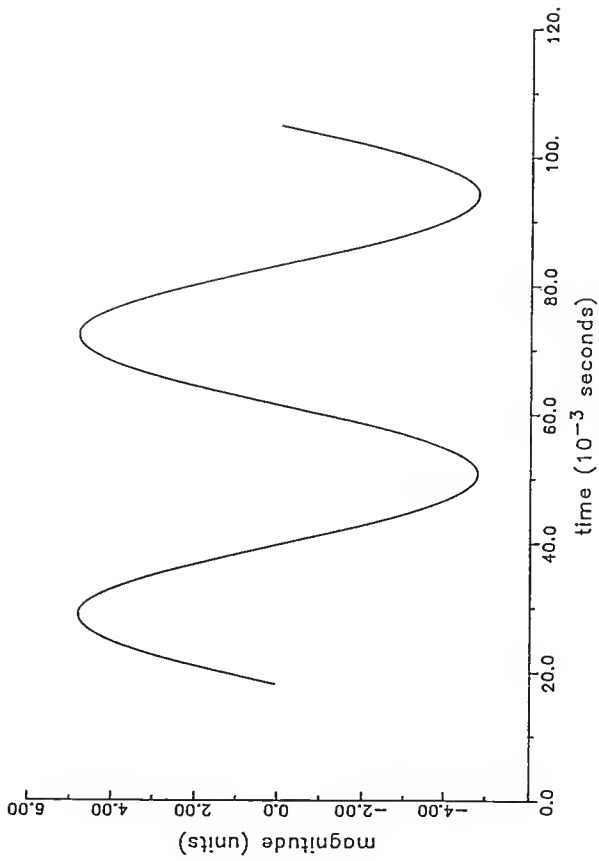
Figure 7.2. Sinewave after phase correction.

71
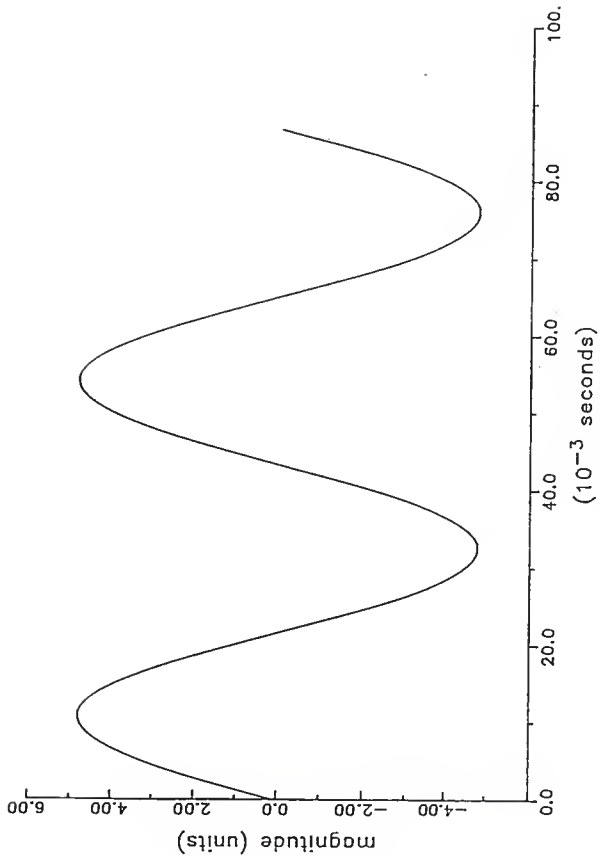
Figure 7.3. Phase corrected sinewave after time shifting.

72

and phase. The rms error between the input sinewave and the best-fit sinewave is also calculated in the function. The best-fit sinewave obtained is plotted in Figure 7.4. Finally the effective bits of the analog to digital converter are calculated using the appropriate formula.

7.2 FUNCTIONS CALLED

The first two functions, *sinewave()* and *digitizer()*, are used to generate the simulated sinewave data used in the parameter estimation and effective bits calculations.

sinewave()

This function generates a sinewave with specified parameters - amplitude, offset, frequency and phase. Inputs to the function are the four parameters, the sampling rate, and the number of samples required. The *sine()* function available in the math library is used. The output of this function is the simulated, software sinewave data record.

digitizer()

This function is the software model of an ideal n-bit analog to digital converter. Both unipolar as well as

73
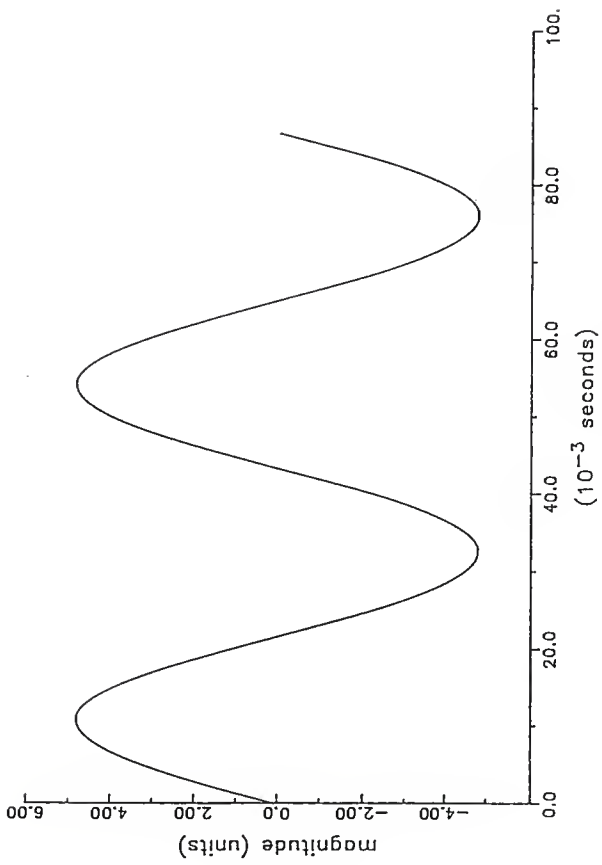
Figure 7.4. Best-fit sinewave.

74

bipolar versions of the ADC are simulated, the choice at any time being left to the user. It essentially scales and quantizes the input continuous waveform to produce the corresponding digital data record. It should be observed that the digital data record consists of only integer values. Inputs to the function are the ADC type, number of bits, full-scale range, and the input data record. The output consists of the digitized data record.

dc_estimator()

Estimation of the dc offset of the unknown sinewave is based on the algorithm developed by Jenq and Crosby [7]. The 4-term Blackman-Harris window is sampled to obtain a window sequence of the form,

$$w_k = 0.35875 - 0.48829Cos(k2\pi/N)$$

$$+ 0.14128Cos(k4\pi/N) - 0.01168Cos(k6\pi/N)$$

where

$$k = 0, 1, \ldots, N - 1.$$

The dc offset can then be estimated as

$$dc\ offset = \frac{\sum_{k=0}^{N-1} w_k s_k}{\sum_{k=0}^{N-1} w_k}$$

The inputs to the function consist of the digitized data record and the number of samples. The estimated value

75

of the dc offset forms the output. This function has been
found to estimate the dc offset quite accurately.

corphase( )

Phase correction of the unknown sinewave is
accomplished by means of the function *corphase( )*. Inputs
to the function are the number of samples of the input data
record, the x data and y data values corresponding to these
samples and the number of samples in *one* complete period.
The output consists of two cycles of the new phase-
corrected, data record of the input sinewave.

To begin with, the first zero-crossing, positive-going
point of the unknown sinewave is located. This forms the
first point of the new data record. Subsequent samples of
the old data record go to form the new data record until
the third zero-crossing, positive-going point is
encountered. This crossover indicates that two complete
cycles of the unknown sinewave have been included in the
new data record.

Next, based upon a knowledge of the first point of the
new data record, the phase of the original sinewave may be
determined by the relation,

phase = 360 - (fc * 360.0 / # points per cycle)

where fc is the first crossover point. After phase correction, the dc offset can be put back into the data record if required.

least_squared_fit()

This function finds the best-fit sinewave to the phase corrected new data record. The rms error between the input sinewave and the best fit sinewave is also calculated. It is based on the four parameter least squared fit technique developed by Peetz et al. [12] and is fairly straight-forward. The detailed derivation of this technique may be found in [14].

The inputs to the function are the number of samples, the x and y coordinates of the data record corresponding to each sample and an initial estimate of the input frequency. The outputs are the estimates of the four parameters of the best fit sinewave - amplitude, offset, frequency and phase, and the rms error.

The initial phase value is calculated within the function. The function is iterative in that it goes around in a loop calculating estimates of the frequency and phase until the final values lie within some prescribed interval, whereupon it exits the loop. In our case, the stopping criterion is when the frequency and phase estimates lie within 0.0001 of the actual value. Using these estimates,

the best fit amplitude and offset are determined. The rms
error between the input sinewave and the best fit sinewave
is calculated, from which the effective bits of the ADC may
be obtained.

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS

The aim of this work has been to thoroughly investigate several algorithms used in sinewave curve fitting and in the subsequent calculation of the effective bits of an ADC. Each algorithm is tested in order to discover its merits and demerits, and to devise techniques for enhancing its performance so that it can function as an integral part of a larger, more comprehensive fitting package.

In this report, three popular algorithms used in parameter estimation are analyzed. These include the closed form approximation technique [7], the three parameter algorithm [14], and the four parameter, general-purpose least squared fit algorithm [14]. Both pure sinewaves as well as "impure" sinewaves, contaminated by noise or by the addition of higher order harmonics to the fundamental, are digitized by the ADC under consideration; these sinewaves are subsequently employed as test inputs. Of the three algorithms listed above, only the last two are extensively tested with all types of inputs. The first algorithm is tested only with pure sinewave data.

79

The next section deals with the conclusions reached after analyzing each of the algorithms. These are followed by our recommendations for future work.

8.1 CONCLUSIONS

The closed form approximation algorithm is found to give fairly accurate estimates of the amplitude and offset of the digitized pure sinewave. Major difficulties arise in the estimation of the frequency and the phase. In order to get even reasonably accurate estimates for these parameters, phase unwrapping of the sinewave has to be very carefully performed to ensure the condition of strict monotonicity of the phase at all times. While this may not be too difficult to accomplish with a pure, known sinewave, it is almost impossible in the case of an unknown sinewave which has even a very small amount of added noise or distortion. Since it is necessary to devise an algorithm capable of handling all types of sinewave inputs, it is clear that this algorithm has only very limited use. However, it is both simple and efficient. Consequently, if a suitable modification could, in fact, be incorporated to facilitate the condition of phase monotonicity, the closed

form approximation algorithm should find extensive applications.

When the frequency of the input sinewave is known, the three parameter fit algorithm is found to be the best technique to employ in estimating the sinewave parameters and the effective bits of the ADC under consideration. It is both simple and efficient, and excellent results are obtained for all the test cases.

Our studies indicate that although the amplitude and offset are accurately estimated for all the phase inputs tested, errors exist in the phase estimates corresponding to input values in the second and third quadrants. We have observed that adding a correction factor of Π to these erroneous phase estimates yields satisfactory results. The three parameter fit algorithm provides excellent values for the number of effective bits in all the test cases; moreover, the maximum variation in the effective bits is no more than 0.1 bit.

The standard four parameter least squared fit algorithm gives fairly good estimates for the amplitude, offset and frequency of unknown sinewaves. Problem arise in the phase estimates and the effective bit calculations. For cases where the input phase is close to zero degrees, the phase estimates and the effective bits determined by the algorithm are relatively accurate. However, for

arbitrary input phase values, the phase estimates obtained bear no discernable resemblance to the actual values; the effective bits calculated are also totally erroneous. To account for these difficulties, we have incorporated a phase correction technique in the standard algorithm. The correction technique essentially shifts the unknown input sinewave to ensure that its starting phase is always zero degrees. The resulting modified four parameter algorithm gives good results for all the test cases considered.

The use of noisy sinewaves as input to the three parameter and four parameter algorithms has led to an interesting observation. An increase in the rms noise amplitude by a factor of 2 is found to cause a reduction of one bit in the corresponding number of effective bits for the ADC. The significance of this observation lies in the fact that the number of bits of an ADC can be determined fairly accurately even for a pure sinewave corrupted by noise. Therefore, any deviations from the expected value are attributable to the presence of non-linearities in the ADC itself; this provides a means for testing the quality of the ADC.

8.2 RECOMMENDATIONS

In this report, we have analyzed three popular algorithms for sinewave parameter estimation and the subsequent computation of effective bits for ADCs. We have presented the relative merits and demerits of the algorithms, and have suggested techniques for enhancing their performance. As follow-ups to this work we would like to make the following recommendations.

With regard to the closed form approximation algorithm, it is necessary to devise an effective technique for phase unwrapping. Specifically, the technique must achieve the desired phase unwrapping without excessive computation even in the case of noisy sinewaves. If this can, in fact, be accomplished, the resulting modified closed form approximation algorithm will prove to be simple, quick and effective.

The modified four parameter least squared fit algorithm also demands further investigation. Presently, the results obtained are sensitive to the sampling rate employed. It is especially important to investigate this effect, and to propose measures for reducing the dependence on the sampling rate. Such an enhancement promises to find widespread application for the algorithm.

The study of the effects of employing input sinewaves corrupted by the addition of higher order harmonics is a topic for future research. This will provide insights on the precise effect of the amplitudes of the added harmonics on the degradation of the effective number of bits of the ADC.

REFERENCES

[1] D. W. Doerfler, "Techniques for testing a 15-bit data acquisition system," M. S. Thesis, Department of Electrical and Computer Engineering, Kansas State University, Manhattan, Kansas, 1985.

[2] A. Gee, "Design and test aspects of a 4-MHz 12-bit analog-to-digital converter," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-35, no. 4, pp. 483-491, 1986.

[3] R. Gorton, "Noise and distortion in transient waveform recorders," *IEEE Proceedings of the Instrumentation and Measurement Conference-1988*, CH2569-2, pp. 208-211, 1988.

[4] P. J. Green, "The use of closed-form approximations for sinewave fitting parameters and the consequent accuracy of effective bit determinations," unpublished manuscript, 1988.

[5] Hewlett-Packard, *Dynamic Performance Testing of A to D Converters*, Product Note 5180A-2, Palo Alto, California, 1980.

[6] Y. C. Jenq, "High-precision sinusoidal frequency estimator based on weighted least square method," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-36, no. 1, pp. 124-127, 1987.

[7] Y. C. Jenq and P. Crosby, "Sinewave parameter estimation algorithm with application to waveform digitizer effective bits measurement," *IEEE Proceedings of the Instrumentation and Measurement Conference-1988*, CH2569-2, pp. 218-221, 1988.

[8] J. Kuffel, T. R. McComb and R. Malewski, "Comparative evaluation of computer methods for calculating the best-fit sinusoid to the digital record of a high-purity sinewave," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-36, no. 2, pp. 418-422, 1987.

[9] T. F. Linnenbrink, "Effective bits: Is that all there is?," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-33, no. 3, pp. 184-187, 1984.

[10] R. A. Malewski, T. R. McComb and M. M. C. Collins, "Measuring properties of fast digitizers employed for recording HV impulses," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-32, no. 1, pp. 12-17, 1983.

[11] B. E. Peetz, "Dynamic testing of waveform recorders," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-32, no. 1, pp. 17-22, 1983.

[12] B. E. Peetz, A. S. Muto and J. M. Neil, "Measuring waveform recorder performance," *Hewlett-Packard Journal*, pp. 21-29, Nov. 1982.

[13] J. Schoukens, R. Pintelon, E. van der Ouderaa and J. Renneboog, "Survey of excitation signals for FFT based signal analyzers," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-37, no. 3, pp. 342-352, 1988.

[14] Waveform Measurements and Analysis Committee of the IEEE Instrumentation and Measurement Society, "Trial use standard for digitizing waveform recorders," Working Draft, P1057/D11, May 1988.

SOURCE CODE LISTINGS

```
/*********************************************************************
 *                                                                   *
 *   SOURCE FILE:     cphase.c                                       *
 *                                                                   *
 *                                                                   *
 *   FUNCTION:        corphase(np, x_data, y_a, points, dc,          *
 *                             newx, newy, npts, realph)             *
 *                                                                   *
 *                                                                   *
 *   DESCRIPTION:     Performs phase correction and time             *
 *                    shifting of the input data record.             *
 *                                                                   *
 *                                                                   *
 *   FUNCTIONS                                                       *
 *   CALLED:          None.                                          *
 *                                                                   *
 *                                                                   *
 *   AUTHOR:          Kamini Shenni                                  *
 *                                                                   *
 *                                                                   *
 *   DATE CREATED:    OI September 1988    Version 1.00              *
 *                                                                   *
 *                                                                   *
 *********************************************************************/

#include <stdio.h>
#include <math.h>
#define MAXCHR   30        /* Character array size.              */
#define MAXPTS 5000        /* Plot array size.                   */
#define MAXTIT   50        /* Title array size.                  */
#define EOL    '\n'        /* New line symbol.                   */
#define ERROR   1          /* Error termination condition.       */
#define NORMAL  0          /* Normal termination condition.      */
#define PI 3.141592653     /* Constant value of PI.              */

int corphase(np, x_data, y_a, points, dc, newx, newy,
             npts, realph)

    int   np,              /* Number of points in input record.  */
          *npts;           /* Number of points in new record.    */

    double  dc,            /* Offset of input data record.       */
            points,        /* Number of points in one period.    */
            *x_data,       /* X-axis values of original record.  */
            *y_a,          /* Y-axis values of original record.  */
            *newx,         /* X-axis values of new data record.  */
            *newy,         /* Y-axis values of new data record.  */
            *realph;       /* Actual input phase value.          */
```

```
{

    int   i, j, k,        /* General purpose indices.           */
          crossover;      /* Value of first crossover point.    */

    double  degrees;      /* Calculated phase.                 */

    char   temp;          /* Dummy variable.                   */


/*-----------------------------------------------------------------*/
/* Find the first zero crossing point and generate new record. */
/*-----------------------------------------------------------------*/
    if (y_a[0] = 0.0)
        {
        if (y_a[1] > 0.0)
            *realph = 0.0;
        else
            *realph = 180.0;

        *npts = np;
        for (i = 0; i < np; i ++)
            {
            newx[i] = x_data[i];
            newy[i] = y_a[i];
            }
        }

    else
        {
        crossover = 0;

        i = 0;
        j = 0;

        do
          {
          if ((y_a[i + 1] > y_a[i]) && (y_a[i + 1] >= 0.0) &&
             (y_a[i] < 0.0))
             {
             if (crossover == 0)
                 {
                 newy[j] = y_a[i + 1];
                 newx[j] = x_data[i + 1];
                 j = j + 1;
                 k = i;  /* first pt after crossover. */
                 crossover = crossover + 1;
                 }
             else
                 {
                 newy[j] = y_a[i + 1];
                 newx[j] = x_data[i + 1];
                 j = j + 1;
                 crossover = crossover + 1;
```

89

```
                    }
                }
            else
                if (crossover > 0)
                    {
                    newy[j] = y_a[i + 1];
                    newx[j] = x_data[i + 1];
                    j = j + 1;
                    }

            i = i + 1;

            }
        while (crossover < 3);


/*--------------------------------------------------------------*/
/* Find phase of original data record.                        */
/*--------------------------------------------------------------*/
        degrees = (double)k * 360.0 / points;
        *realph = 360.0 - degrees;

        *npts = j - 1;
        }

        printf("\n\nCorphase function successful.");

    return(0);

}
```

90

```
/*********************************************************************
*                                                                   *
*  SOURCE FILE:     dc.c                                             *
*                                                                   *
*                                                                   *
*  FUNCTION:        dc_estimator(np, y_inp, dc)                      *
*                                                                   *
*                                                                   *
*  DESCRIPTION:     Code employing Jena's method for                *
*                   obtaining the initial dc offset estimate        *
*                   of the unknown data record.                     *
*                                                                   *
*                                                                   *
*  FUNCTIONS                                                         *
*  CALLED:          cos()              (user library)               *
*                                                                   *
*                                                                   *
*  AUTHOR:          Kamini Shenoi                                    *
*                                                                   *
*                                                                   *
*  DATE CREATED:    10 October 1988     Version 1.00                *
*                                                                   *
*                                                                   *
*********************************************************************/


#include <stdin.h>
#include <math.h>
#define EOL   '\n'              /* New line symbol.                   */
#define ERROR   1               /* Error termination condition.       */
#define MAXPTS  10000           /* Maximum number of points.          */
#define NORMAL  0               /* Normal termination condition.      */
#define PI 3.141592653          /* Constant value of PI.              */


int dc_estimator(np, y_inp, dc)

     int    np;                 /* Number of data points.             */

     double  *dc,               /* DC offset estimate.                */
             *y_inp;            /* Input data record.                 */

{
     int  k;                    /* General-purpose loop counter.      */

     double dnum,               /* Numerator of dc offset.            */
            ddenom,             /* Denominator of dc offset.          */
            window[MAXPTS];
                                /* Window sequence.                   */
```

91

```
/*---------------------------------------------------------------------*/
/* Obtain the window sequence (4 - term Blackman-Harris window.*/
/*---------------------------------------------------------------------*/
      for (k = 0; k < np; k ++)

          window[k] = 0.35875 - 0.48829 * cos((double)k * 2.0 *
                      PI / (double)np) + 0.14128 *
                      cos((double)k * 4.0 * PI / (double)np)
                      - 0.01168 * cos((double)k * 6.0 * PI /
                      (double)np);


/*---------------------------------------------------------------------*/
/* Calculate the DC - offset estimator.                                */
/*---------------------------------------------------------------------*/
      dnum = 0.0;
      ddenom = 0.0;

      for (k = 0; k < np; k ++)
          {
          dnum = dnum + window[k] * s_inp[k];
          ddenom = ddenom + window[k];
          }

      *dc = dnum / ddenom;


/*---------------------------------------------------------------------*/
/* Normal termination.                                                 */
/*---------------------------------------------------------------------*/
      return(NORMAL);
}
```

92

```
/*******************************************************************
 *                                                                 *
 *  SOURCE FILE:     digit.c                                       *
 *                                                                 *
 *                                                                 *
 *  FUNCTION:        digitizer(adctype, num_bits, vfs, num_pts,    *
 *                             y_a, y_d, maxinp)                    *
 *                                                                 *
 *                                                                 *
 *  DESCRIPTION:     Ideal N-bit ADC (bipolar or unipolar).        *
 *                   Digitizes an analog input voltage.            *
 *                                                                 *
 *  FUNCTIONS                                                       *
 *  CALLED:          None.                                         *
 *                                                                 *
 *                                                                 *
 *  AUTHOR:          Kamini Shenoi                                 *
 *                                                                 *
 *                                                                 *
 *  DATE CREATED:    28 June 1988    Version 1.00                  *
 *                                                                 *
 *                                                                 *
 *******************************************************************/


#include <stdio.h>
#include <math.h>
#define EOL   '\n'        /* New line symbol.                 */
#define NORMAL  0         /* Normal termination condition.    */


int digitizer(adctype, num_bits, vfs, num_pts, y_a, y_d, maxinp)

     int      num_pts,       /* Number of points to be digitized.  */
              num_bits,      /* Number of bits of the ADC.         */
              *y_d;          /* Digital code corres. to input val. */

     char     adctype;       /* Type of ADC.                       */

     double   vfs,           /* Full scale voltage of ADC.         */
              *maxinp,
              *y_a;
                             /* Array of input voltages.           */

{
     int  i;                 /* General-purpose loop counter.      */

     char   name;            /* Type of ADC.                       */
```

93

```
    double expn,            /* 2.0 ** n                          */
            precision)      /* Precision of the ADC.             */


/*-------------------------------------------------------------------*/
/* Calculate the precision of the ADC, i.e.the value of an lsb.*/
/*-------------------------------------------------------------------*/
    expn = pow(2.0, (double)num_bits);
    *maxinp = 0.0;

    if (adctype == 'b' || adctype == 'B')
        {
        precision = vfs / (expn / 2.0);
        for (i = 0; i < num_pts; i++)
            y_d[i] = (int)((((y_a[i] + vfs) * (expn - 1)) /
                            (2.0 * vfs)) + (precision / 2.0));
        }
    else
        {
        precision = vfs / expn;

        for (i = 0; i < num_pts; i++)
            {
            if (y_a[i] > *maxinp)
                *maxinp = y_a[i];
            y_d[i] = (int)(((y_a[i] * (expn - 1))
                / (vfs)) + 0.5);

            if (y_d[i] > (expn - 1))
                y_d[i] = (expn - 1);

            }
        }


/*-------------------------------------------------------------------*/
/* Normal termination.                                          */
/*-------------------------------------------------------------------*/
    return(NORMAL);
}
```

```
/*********************************************************************
*                                                                   *
*   SOURCE FILE:    lsf.c                                            *
*                                                                   *
*                                                                   *
*   FUNCTION:       least_squared_fit(n_points, time,               *
*                       sine_inp, f_estimate, offset,               *
*                       amplitude, freq, phase, rms_error)          *
*                                                                   *
*                                                                   *
*   DESCRIPTION:    Employs the four parameter least squared        *
*                   fit technique for parameter estimation of       *
*                   the input sinewave.                             *
*                                                                   *
*                                                                   *
*   FUNCTIONS                                                        *
*   CALLED:         None.                                           *
*                                                                   *
*                                                                   *
*   AUTHOR:         Kamini Shenoi                                    *
*                                                                   *
*                                                                   *
*   DATE CREATED:   21 August 1988     Version 1.00                 *
*                                                                   *
*                                                                   *
*********************************************************************/


#include <stdio.h>
#include <math.h>
#define EOL   '\n'           /* New line symbol.                 */
#define ERROR    1           /* Error termination condition.     */
#define MAXPTS   5000        /* Maximum number of points.        */
#define NORMAL   0           /* Normal termination condition.    */
#define PI 3.141592653

int least_squared_fit(n_points, time, sine_inp, f_estimate,
                      offset, amplitude, freq, phase,
                      rms_error)


    int    n_points;         /* Number of data points.           */

    double *amplitude,       /* Amplitude estimate of sinewave.  */
           *offset,          /* Offset estimate of sinewave.     */
           *freq,            /* Frequency estimate of sinewave.  */
           *phase,           /* Phase estimate of sinewave.      */
           *sine_inp,        /* Input sample values.             */
           *time,            /* Input sampling intervals.        */
           f_estimate;       /* Initial frequency estimate.      */
```

```
              frms_error;    /* Calculated rms error.                 */

<
    int   i;                      /* General-purpose loop counter.     */

    double   alpha, beta, sum_a, sum_b, sum_y, a_y, b_y, a_b,
             b_b, a_a, s_y, b_t_y, a_b_t, b_b_t, b_b_t_t, b_t,
             a_t, b_t_t, a11num, a12num, a21num, a22num, denom,
             a11, a12, a21, a22, amp_num,  amp_denom, value;
             phi, rs1_den, rs2_den, r, s, w, rms, aestimate,
             oestimate, max, min, sign, angle, f_error, p_error;

    char   temp;          /* Dummy variable.                        */


/*---------------------------------------------------------------------*/
/* Find the max and min value of input record.                        */
/*---------------------------------------------------------------------*/
    min = 100.0;
    max = 0.0;

    for (i = 0; i < n_points; i ++)
        {
        if (sine_inp[i] > max)
            max = sine_inp[i];
        if (sine_inp[i] < min)
            min = sine_inp[i];
        }

    aestimate = (max - min) / 2.0;
    oestimate = (max + min) / 2.0;

    if (sine_inp[1] >= sine_inp[0])
        sign = 1.0;
    else
        sign = - 1.0;

    angle = ((sine_inp[0] - oestimate) / aestimate);
    phi = sign * acos(angle);

    value = (double)n_points;

    *freq = f_estimate * 2.0 * PI;
    *phase = phi;


/*---------------------------------------------------------------------*/
/* Parameter estimation loop begins here.                             */
/*---------------------------------------------------------------------*/
    do
        {
        w = *freq;
        phi = *phase;
```

96

```
/*-----------------------------------------------------------------*/
/* Initialize the sums.                                            */
/*-----------------------------------------------------------------*/
        sum_a = 0.0;
        sum_b = 0.0;
        sum_y = 0.0;
        a_y = 0.0;
        b_y = 0.0;
        a_b = 0.0;
        b_b = 0.0;
        a_a = 0.0;
        y_y = 0.0;
        b_t_y = 0.0;
        a_b_t = 0.0;
        b_b_t = 0.0;
        b_b_t_t = 0.0;
        b_t = 0.0;
        a_t = 0.0;
        b_t_t = 0.0;



/*-----------------------------------------------------------------*/
/* Compute the sixteen sums required for the estimates.            */
/*-----------------------------------------------------------------*/
        for (i = 0; i < n_points; i ++)
            {
            alpha = cos(w * time[i] + phi);
            beta =  sin(w * time[i] + phi);

            sum_y = sum_y + sine_inp[i];
            sum_a = sum_a + alpha;
            sum_b = sum_b + beta;
            a_y = a_y + alpha * sine_inp[i];
            b_y = b_y + beta * sine_inp[i];
            a_b = a_b + alpha * beta;
            b_b = b_b + beta * beta;
            a_a = a_a + alpha * alpha;
            y_y = y_y + sine_inp[i] * sine_inp[i];
            b_t_y = b_t_y + beta * time[i] * sine_inp[i];
            a_b_t = a_b_t + alpha * beta * time[i];
            b_b_t = b_b_t + beta * beta * time[i];
            b_b_t_t = b_b_t_t + beta * beta * time[i] * time[i];
            b_t = b_t + beta * time[i];
            a_t = a_t + alpha * time[i];
            b_t_t = b_t_t + beta * time[i] * time[i];
            }



/*-----------------------------------------------------------------*/
/* Using the sums, calculate a11, a12, a21, a22, r and s.          */
/*-----------------------------------------------------------------*/
            a11num = (a_b_t - (b_t * (sum_a / value))) *
                     (a_b_t - (a_t * (sum_b / value))) -
```

```
                      (a_a - (sum_a * (sum_a / value))) *
                      (b_b_t_t - (b_t_t * (sum_b / value)));

         a12num = (a_b_t - (b_t * (sum_a / value))) *
                  (a_b - (sum_a * (sum_b / value))) -
                  (a_a - (sum_a * (sum_a / value))) *
                  (b_b_t - (b_t * (sum_b / value)));

         a21num = (a_b - (sum_b * (sum_a / value))) *
                  (a_b_t - (a_t *(sum_b / value))) -
                  (a_a - (sum_a * (sum_a / value))) *
                  (b_b_t - (b_t * (sum_b /value)));

         a22num = (a_b - (sum_b * (sum_a /value))) *
                  (a_b - (sum_a * (sum_b / value))) -
                  (a_a - (sum_a * (sum_a / value))) *
                  (b_b - (sum_b * (sum_b / value)));

         denom = (a_a - (sum_a * (sum_a / value))) *
                 (a_a - (sum_a * (sum_a / value)));


         a11 = a11num / denom;
   /     a12 = a12num / denom;
         a21 = a21num / denom;
         a22 = a22num / denom;


         rs1_den = (a_y - sum_a * (sum_y / value));
         rs2_den = (a_a - sum_a * (sum_a / value));

         r = ((b_t_y - b_t * (sum_y / value)) / rs1_den) -
             ((a_b_t - b_t * (sum_a / value)) / rs2_den);

         s = ((b_y - sum_b * (sum_y / value)) / rs1_den) -
             ((a_b - sum_b * (sum_a / value)) / rs2_den);


/*-------------------------------------------------------------*/
/* Using the estimates calculate the freq and phase.          */
/*-------------------------------------------------------------*/
         *freq = w + (((a22 * r) - (a12 * s)) /
                 ((a11 * a22) - (a12 * a21)));

         *phase = phi + (((a11 * s) - (a21 * r)) /
                  ((a11 * a22) - (a12 * a21)));


         f_error = *freq - w;
         p_error = *phase - phi;

      }
   while (((f_error * f_error) > 0.00001) ||
          ((p_error * p_error) > 0.00001));
```

98

```
/*------------------------------------------------------------------*/
/* Calculate the fitted amplitude and offset.                       */
/*------------------------------------------------------------------*/
     amp_num = 0.0;
     amp_denom = 0.0;

     amp_num = (a_y - (sum_a * (sum_y / value)) + b_y - (sum_b *
               (sum_y / value)) + b_t_y - (b_t * (sum_y / value)));

     amp_denom = (a_a - (sum_a * (sum_a / value)) + a_b - (sum_b
                  * (sum_a / value)) + a_b_t - (b_t * (sum_a / value

     *amplitude = amp_num / amp_denom;

     *offset = (sum_y / value) - (*amplitude * (sum_a / value));


/*------------------------------------------------------------------*/
/* Calculate the rms error.                                         */
/*------------------------------------------------------------------*/
     rms = (y_y / value) + (*amplitude * *amplitude * a_a /value) -
           (2.0 * *amplitude * a_y / value) + (*offset * *offset)
           - (2.0 * *offset * sum_y / value) +
           (2.0 * *amplitude * *offset * sum_a / value);

     *rms_error = sqrt(rms);


/*------------------------------------------------------------------*/
/* Normal termination.                                              */
/*------------------------------------------------------------------*/
     return(NORMAL);
}
```

```
/*****************************************************
*                                                   *
*   SOURCE FILE:    sinewave.c                       *
*                                                   *
*                                                   *
*   FUNCTION:       sinewave(np, amp, dc, freq, num_cycles, *
*                   theta, x_dat, y_dat)            *
*                                                   *
*                                                   *
*   DESCRIPTION:    Generates a sinewave with parameters *
*                   specified by the user. This sinewave *
*                   forms the input to the digitizer. *
*                                                   *
*                                                   *
*                                                   *
*   FUNCTIONS                                        *
*   CALLED:         sin()          (math library)   *
*                                                   *
*                                                   *
*   AUTHOR:         Kamini Shenoi                    *
*                                                   *
*                                                   *
*   DATE CREATED:   23 June 1988     Version 1.00   *
*                                                   *
*                                                   *
*****************************************************/

#include <stdio.h>
#include <math.h>
#define MAXPTS  4000     /* Plot array size.                */
#define EOL  '\n'        /* New line symbol.                */
#define NORMAL  0        /* Normal termination condition.   */
#define PI 3.141592653   /* Constant value Pi.              */


int sinewave(np, amp, dc, freq, num_cycles, theta, x_dat, y_dat)

    int     np;          /* Number of data points.          */
    double  amp,         /* Amplitude of sinewave.          */
            dc,          /* D.C. offset of sinewave.        */
            freq,        /* Input test frequency of sinewave. */
            num_cycles,  /* Number of cycles.               */
            theta,       /* Phase shift (degrees).          */
            *x_dat,      /* Array of abscissa values.       */
            *y_dat;      /* Array of ordinate values.       */

{

    int  i;              /* General-purpose loop counter.   */
```

```
        double  inc,        /* Increment for abscissa values.     */
                angle,      /* Argument for sin function (rads).  */
                t;          /* Abscissa value (degrees).          */


/*-----------------------------------------------------------------*/
/* Set initial value and increment for abscissa values.           */
/*-----------------------------------------------------------------*/
        t = 0.0;
        inc = (num_cycles)/(freq * (np - 1));


/*-----------------------------------------------------------------*/
/* Generate the sine wave.                                         */
/*-----------------------------------------------------------------*/
        for (i = 0; i < np; i++)
            {
            x_dat[i] = t;
            angle = 2.0 * PI * freq * t + theta * (PI / 180.0);
            y_dat[i] = amp * sin(angle) + dc;
            t = t + inc;
            }


/*-----------------------------------------------------------------*/
/* Normal termination.                                             */
/*-----------------------------------------------------------------*/
        return (NORMAL);

}
```

EFFECTIVE BITS OF AN ANALOG-TO-DIGITAL CONVERTER:
Analysis of Sinewave Curve Fitting Algorithms

KAMINI SHENOI

B.E., Bangalore University, India, 1985

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

ABSTRACT

The present work deals with the evaluation of various
algorithms used in sinewave parameter estimation and the
subsequent effective bit computations for Analog to Digital
Converters (ADCs).

Three algorithms have been investigated in this work.
They include the Closed Form Approximation Method, the
Three Parameter Fit Method (with known frequency) and the
Four Parameter Least Squared Fit Method (with unknown
frequency). Both the ADC under consideration as well as
the reference data employed in testing the algorithms are
simulated in software. The test inputs include pure and
corrupted sinewaves. For each case, the degradation in the
effective bits for the ADC is observed.

Our studies indicate that the Closed Form Approxima-
tion is not as simple as anticipated. To obtain even
reasonably good estimates, strict phase unwrapping is
vital. This is not easily accomplished for unknown
sinewaves.

The Three Parameter Fit Method gives excellent results
when the input frequency is known. Moreover, it is simple
and efficient. The only disadvantage is that a correction

factor has to be incorporated in the algorithm to compensate for phase estimation errors.

The standard Four Parameter Least Squared Fit Method is adequate only for input phase values in the neighborhood of zero degrees. Modifications have been made to this algorithm to deal with arbitrary input phase values. These modifications yield satisfactory results.