Phase and Frequency Estimation:
High-Accuracy and Low-Complexity Techniques

by

Yizheng Liao

A Thesis
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Electrical and Computer Engineering
by

May 2011

APPROVED:

Professor D. Richard Brown III, Major Advisor

Professor John A. McNeill, Committee Member

Professor Andrew G. Klein, Committee Member

Abstract

The estimation of the frequency and phase of a complex exponential in additive white Gaussian noise (AWGN) is a fundamental and well-studied problem in signal processing and communications. A variety of approaches to this problem, distinguished primarily by estimation accuracy, computational complexity, and processing latency, have been developed. One class of approaches is based on the Fast Fourier Transform (FFT) due to its connections with the maximum likelihood estimator (MLE) of frequency. This thesis compares several FFT-based approaches to the MLE in terms of their estimation accuracy and computational complexity. While FFT-based frequency estimation tends to be very accurate, the computational complexity of the FFT and the latency associated with performing these computations after the entire signal has been received can be prohibitive in some scenarios. Another class of approaches that addresses some of these shortcomings is based on linear regression of samples of the instantaneous phase of the observation. Linear-regression-based techniques have been shown to be very accurate at moderate to high signal to noise ratios and have the additional benefit of low computational complexity and low latency due to the fact that the processing can be performed as the samples arrive. These techniques, however, typically require the computation of four-quadrant arctangents, which must be approximated to retain low computational complexity. This thesis proposes a new frequency and phase estimator based on simple estimates of the zero-crossing times of the observation. An advantage of this approach is that it does not require arctangent calculations. Simulation results show that the zero-crossing frequency and phase estimator can provide high estimation accuracy, low computational complexity, and low processing latency, making it suitable for real-time applications. Accordingly, this thesis also presents a real-time implementation of the zero-crossing frequency and phase estimator in the context of a time-slotted round-trip carrier synchronization system for distributed beamforming. The experimental results show this approach can outperform a Phase Locked Loop (PLL) implementation of the same distributed beamforming system.

Acknowledgements

First of all, I would like to express my deep and sincere gratitude to my advisor, Professor D. Richard Brown, for providing me this great research opportunity, for giving me the professional and insightful comments and suggestions on my research work, and for encouraging and motivating me to move forward on the research. As an advisor, he taught me how to start a research project, and how to face the challenges during the research work. The research I carried out under Professor Brown has deeply motivated me to pursue my Ph.D degree in the Electrical and Computer Engineering, and to become a faculty member in the future.

Besides my advisor, I would also like to express gratitude to my committee members: Professor John A. McNeill and Professor Andrew G. Klein. Thank you for reviewing my thesis, participating in my defence, and for asking me challenging questions and giving me professional comments on my thesis.

Most importantly, I want to say thanks to my parents. Without your encouragement and support I would not have been able to come to the United States to study and be where I am today. I am forever in debt to your for you unconditional love!

I would also like to thank my fellow spinlab members, Min Ni and Joshua Bacon. Thank you for taking the time to discuss my research and help me with the experiments.

I also want to say thanks to all of my friends at WPI. Because of you, the last three years at WPI have provided me with some of my most memorable moments in life.

Last, but not least, I am grateful for the generous support of Texas Instruments for donating the equipment, and for the financial support of National Science Foundation.

Contents

List of Figures **vi**

List of Tables **viii**

1 Introduction **1**

2 Background **4**
2.1 Cramer-Rao Lower Bound 5
2.2 FFT-Based Maximum Likelihood Estimation 6
2.3 Linear-Regression-Based Maximum Likelihood Estimation 7

3 FFT-Based Phase and Frequency Estimation **11**
3.1 Approximate Maximum Likelihood Estimation using Fast Fourier Transform and No Post-Processing 13
3.2 Approximate Maximum Likelihood Estimation using FFT and Quadratic Interpolation 18
3.3 Approximate Maximum Likelihood Estimation using FFT and Secant Method 22
3.4 Approximate Maximum Likelihood Estimation using FFT and Newton's Method 31
3.4.1 Comparison of the FFT-Secant MLE and the FFT-Newton MLE . . 34
3.5 Approximate Maximum Likelihood Estimation using FFT and Bisection Method 37
3.6 Conclusion 43

4 Zero Crossing Phase and Frequency Estimation **45**
4.1 Algorithm 46
4.1.1 Zero-Crossing Phase and Frequency Estimator 46
4.1.2 Sequential Implementation 48
4.2 Computational Complexity 49
4.3 Numerical Results and Discussion 50
4.4 Conclusion 54

5 Zero Crossing Estimator Refinements **55**
5.1 Zero Crossing Estimator using Local Linear Regression for Zero-crossings . 55
5.1.1 Numerical Results and Discussion 59
5.2 Zero Crossing Estimator using Pre-Filtered Observation 59

5.2.1 Numerical Results and Discussion 59
5.3 Zero Crossing Estimator using Non-Linear Iterative Method 62
5.3.1 Numerical Results and Discussion 63
5.4 Conclusion 66

6 A Software-Defined-Radio Implementation of Time-Slotted Round-Trip Carrier Synchronization for Distributed Beamforming via Zero-Crossing Estimation 68
6.1 Two-Source Synchronization 70
6.2 Experimental Methodology for Two-Source Test in Wire-Connected Channel 71
6.3 Zero-Crossing Estimator Implemented on Source Node 74
6.3.1 Implementation of the Round-Trip Time Table 74
6.3.2 Implementation of Zero-Crossing Estimator 76
6.3.3 Implementation of Holdover 79
6.3.4 Implementation of FIR Filter 80
6.4 Data Analysis Methodology 81
6.5 Experimental Results 82

7 Conclusion 83

A The Specification of Computer and Matlab for Simulation 85
A.1 The Specification of Computer for Simulation 85
A.2 The Specification of Matlab Software for Simulation 85

B Source Code of the TMS320C6713 Source Node in the Two-Source System 86

C Approximate Maximum Likelihood Estimation using Secant Method Matlab Source Code 110
C.1 Main Function 110
C.2 Function of Implementing the FFT-Secant MLE 112
C.3 Function of Computing $A(\omega)$ 114
C.4 Function of Computing $F^{'}(\omega)$ 114
C.5 Function of Phase Unwrapping 116

Bibliography 117

List of Figures

3.1 Application of Maximum Likelihood Estimation. 12
3.2 FFT MLE example with $M = 2^{12}$. 15
3.3 Mean squared frequency estimation error for complex observation of FFT MLE. 16
3.4 Mean squared phase estimation error for complex observation of FFT MLE. 16
3.5 Total execution time for 500 complex observations of FFT MLE. 17
3.6 FFT-Quad MLE Example. 19
3.7 Mean squared frequency estimation error for complex observation of FFT-Quad MLE. 20
3.8 Mean squared phase estimation error for complex observation of FFT-Quad MLE. 21
3.9 The Secant Method example with two iterations. 23
3.10 FFT-Secant MLE example. 26
3.11 Mean squared frequency estimation error for complex signal of FFT-Secant MLE with $\epsilon = 10^{-4}$. 27
3.12 Mean squared phase estimation error for complex signal of FFT-Secant MLE with $\epsilon = 10^{-4}$. 28
3.13 Mean squared frequency estimation error for complex signal of FFT-Secant MLE with $\epsilon = 10^{-6}$. 29
3.14 Mean squared phase estimation error for complex signal of FFT-Secant MLE with $\epsilon = 10^{-6}$. 30
3.15 FFT-Newton MLE example. 33
3.16 Mean squared frequency estimation error for complex signal of FFT-Newton MLE with $\epsilon = 10^{-4}$. 34
3.17 Mean squared phase estimation error for complex signal of FFT-Newton MLE with $\epsilon = 10^{-4}$. 35
3.18 Mean squared frequency estimation error for complex signal of FFT-bisection MLE with $\epsilon = 10^{-4}$. 39
3.19 Mean squared phase estimation error for complex signal of FFT-bisection MLE with $\epsilon = 10^{-4}$. 40
3.20 Mean squared frequency estimation error for complex signal of FFT-bisection MLE with $\epsilon = 10^{-6}$. 41

3.21 Mean squared frequency estimation error for complex signal of FFT-bisection MLE with $\epsilon = 10^{-8}$. 42

4.1 State machine implementation of the zero crossing detector. 47
4.2 Noisy observation with $\alpha = 0.5$. 47
4.3 Mean squared frequency estimation error of ZC estimator with $\alpha = 0.4$. . . 51
4.4 Mean squared phase estimation error of ZC estimator with $\alpha = 0.4$. 52
4.5 Mean squared frequency estimation error of ZC estimator with $\alpha = 0.2$. . . 53
4.6 Mean squared phase estimation error of ZC estimator with $\alpha = 0.2$. 53

5.1 Mean squared frequency estimation error of ZC estimator via local linear interpolation with $\alpha = 0.4$. 57
5.2 Mean squared phase estimation error of ZC estimator via local linear interpolation with $\alpha = 0.4$. 58
5.3 Mean squared frequency estimation error of ZC estimator with $\alpha = 0.4$ and a 64th order FIR filter. 60
5.4 Mean squared phase estimation error of ZC estimator with $\alpha = 0.4$ and a 64th order FIR filter. 61
5.5 Mean squared frequency estimation error of ZC estimator via the Secant Method with $\alpha = 0.4$. 64
5.6 Mean squared phase estimation error of ZC estimator via the Secant Method with $\alpha = 0.4$. 65

6.1 Time-slotted round-trip carrier synchronization system with two source nodes. 69
6.2 Implementation block diagram of the two-source time-slotted round-trip carrier synchronization system where the blue and green lines each represents a different signal wired path 72
6.3 Effect of multipath on ZC estimation and holdover. 81

List of Tables

4.1 Phase of a complex exponential at zero crossing 46
4.2 Computational Complexity of Tretter, Kay, and zero-crossing estimators . . 50

6.1 Two-source round-trip synchronization protocol timing. After detection of the primary beacon, each node keeps time using its sample clock running at 16 kHz. 73
6.2 Experimental results of two-source wired-channel tests. Each experiment consisted of 100 distributed beamforming tests. 82

A.1 The Specification of simulation computer 85
A.2 The Specification of Matlab 85

Chapter 1

Introduction

The estimation of the frequency and phase of a complex exponential in additive white Gaussian noise (AWGN) is a fundamental and well-studied problem in signal processing and communications. Its numerous applications include carrier recovery in a communication system [1], determination of the object position in radar and sonar systems [2, 3], estimation of the heart rate of a fetus in biomedicine [4], and carrier synchronization in a distributed beamforming system [5]. Regardless of the application, poor estimation can lead to disastrous results. For example, in communication system, with the poor carrier frequency estimate, the down-converter may not be able to demodulate the passband signal to baseband [1]. In the smart antenna system and speech processing system, a poor phase estimator may cause the system to fail to identify the direction of arrival of the signal [6, 7].

Nowadays, a variety of approaches to the frequency and phase estimation problem, distinguished primarily by estimation accuracy, computational complexity, and processing latency, have been developed. One class of approaches is based on the Fast Fourier Transform (FFT) due to its connections with the maximum likelihood estimation (MLE) of frequency. The MLE has very high accuracy because it achieves the Cramer-Rao Lower Bound (CRLB), which is the minimum possible error for the unbiased estimator, over a wide range of Signal-to-Noise Ratio (SNR) values. Several FFT-based MLEs have been proposed in [8], [9], [10], [11], and [12]. However, none of the literature compares the computational complexity of each approach. Therefore, in this thesis, we compare the accuracy

and the computational complexity of the approach given in [8] and its refinements. The results show that by using the root-finding algorithms the accuracy of the FFT-based MLE is improved significantly. Also, the computational complexity is reduced by avoiding the FFT with a large number of points. Although the FFT-based MLE can provide a high accuracy of estimation, the latency can be prohibitive in some scenarios because the FFT can only be performed after receiving the entire signal.

Another class of approaches that addresses the shortcomings of the FFT-based MLEs is based on linear regression of samples of the instantaneous phase of the observation. Several approaches have been proposed in [13], [14], [15], and [16]. Linear-regression-based techniques have been shown to be very accurate at moderate to high SNRs. In addition, some of the approaches have been shown to have low computational complexity and low latency due to the fact that the processing can be performed on a sample-by-sample basis. These techniques, however, typically require the computation of four-quadrant arctangents, which must be approximated to retain low computational complexity. In this thesis, we propose a new frequency and phase estimator called the *zero-crossing phase and frequency estimator*. The proposed estimator is based on the simple estimates of the zero-crossing times of the observation. Compared with the estimators presented in [13] and [14], our approach has similar performance, but lower computational complexity. Our proposed estimator avoids the arctangent operation by using the instantaneous phase of each zero crossing, which is known. In addition, rather than computing the instantaneous phase of each received sample, our approach only computes the instantaneous phase of the zero crossings. Therefore, less operations are required. Furthermore, our approach has low latency because it can be implemented in a sequential way. Due to its high accuracy, low computational complexity, and low processing latency, the proposed zero-crossing estimator is suitable for real-time applications.

To demonstrate the real-time applicability of the zero-crossing phase and frequency estimator, this thesis also presents a real-time implementation of the zero-crossing phase and frequency estimator in the context of a distributed beamforming system utilizing the time-slotted round-trip carrier synchronization protocol. Compared with the experimental results of the same distributed beamforming system implemented via a hybrid Phase Locked

Loop (PLL) in [17], our approach offers less signal power lost at the destination. Also, our experimental results are more consistent.

The rest of this thesis is organized as follows. Chapter 2 introduces the Cramer-Rao Lower Bound and gives a review of FFT-based and linear-regression-based estimations. Chapter 3 discusses the FFT-based maximum likelihood estimation and its refinements by using quadratic interpolation and root-finding algorithms. For each estimator in Chapter 3, we provide numerical results as a function of SNR. In addition, the computational complexity is discussed for each estimator. Chapter 4 proposes the algorithm of the Zero-Crossing phase and frequency estimator, and also compares numerical results with Tretter's estimator and Kay's estimator. Chapter 5 presents three refinements of the proposed ZC estimator. Each refinement is analysed numerically, and the performance is compared with the fundamental zero-crossing estimator, Tretter's estimator, and Kay's estimator. In Chapter 6, the zero crossing estimator is implemented in a time-slotted round-trip carrier synchronization distributed beamforming system by software-defined-radio in a wire-connected channel. Finally, the software implementation of the zero crossing algorithm is discussed and the methodology of the experiment is given. This thesis then concludes with experimental results from this chapter.

Chapter 2

Background

In the problem of estimating the unknown parameters of a single tone in noise from the discrete-time observations, we consider the complex-valued received signal

$$z[n] = b_0 \exp(j(\omega_0 nT + \theta_0)) + w[n] \tag{2.1}$$

for $n = n_0, \ldots, n_0 + N - 1$, where frequency ω_0, $\omega_0 \geq 0$, amplitude b_0, and phase offset θ_0, $-\pi \leq \theta_0 < \pi$, are unknown constants. The variable T denotes the sampling period, n_0 denotes the index of the first sample, and $w[n]$ is a zero-mean proper complex Gaussian random variable with $\text{var}\{\Re(w[n])\} = \text{var}\{\Im(w[n])\} = \sigma^2$. The covariance of $\Re(w[n])$ and $\Im(w[n])$ is zero. Therefore, we assume that $w[n]$ are independent and identically distributed (i.i.d) for $n = n_0, n_0 + 1, \ldots, n_0 + N - 1$. The Signal-to-Noise ratio (SNR) is defined as $\text{SNR} := \frac{b_0^2}{2\sigma^2}$.

The N-sample observation of (2.1) is provided as an input to a phase and frequency estimator. The phase and frequency estimates generated by the estimators are denoted as $\hat{\theta}$ and $\hat{\omega}$ respectively. The resulting phase and frequency errors are denoted as $\bar{\theta} := \theta_0 - \hat{\theta}$ and $\bar{\omega} := \omega_0 - \hat{\omega}$, respectively.

2.1 Cramer-Rao Lower Bound

In this paper, we use the Cramer-Rao Lower Bound (CRLB) as a benchmark for the performance of an estimator. An unbiased estimator that achieves the CRLB is said to be "efficient" , in the sense that it achieves the best possible performance in the context of a squared-error cost. The Fisher information matrix for the CRLB for complex signal is [8, 12]

$$\mathbf{J}(\beta) := \frac{1}{\sigma^2} \begin{bmatrix} b_0^2 T^2 (n_0^2 N + 2n_0^2 P + Q) & 0 & b_0^2 T(n_0 N + P) \\ 0 & N & 0 \\ b_0^2 T(n_0 N + P) & 0 & b_0^2 N \end{bmatrix} \tag{2.2}$$

where $\beta = [\omega_0, b_0, \theta_0]^T$,

$$P := \sum_{n=0}^{N-1} n = \frac{N(N-1)}{2} \tag{2.3}$$

$$Q := \sum_{n=0}^{N-1} n^2 = \frac{N(N-1)(2N-1)}{6} \quad \text{and}$$

$$\beta := [\omega_0, b_0, \theta_0]^T. \tag{2.4}$$

When all three parameters are unknown, after inverting all the variations of the information matrix $\mathbf{J}$, the variances obtain the following set of bounds:

$$\text{var}\{\hat{b_0}\} \geq \frac{\sigma^2}{N} \tag{2.5}$$

$$\text{var}\{\hat{\omega}_0\} \geq \frac{12\sigma^2}{b_0^2 T^2 N(N^2-1)} \tag{2.6}$$

$$\text{var}\{\hat{\theta}_0\} \geq \frac{12\sigma^2(n_0^2 N + 2n_0 P + Q)}{b_0^2 N^2 (N^2-1)} \tag{2.7}$$

The CRLB for the covariance of the frequency and phase errors of a complex exponential in AWGN when both of the phase and frequency are unknown is given as [8]

$$\mathrm{cov}\left\{\left[\bar{\omega},\bar{\theta}\right]^T\right\} \geq \frac{\sigma^2}{b_0^2}\begin{bmatrix} \frac{N}{T^2(NP-Q^2)} & \frac{-(Nn_0+P)}{T(NP-Q^2)} \\ \frac{-(Nn_0+P)}{T(NP-Q^2)} & \frac{Nn_0^2+2n_0P+Q}{(NP-Q^2)}. \end{bmatrix} \tag{2.8}$$

where the notation $A \geq B$ means that $A-B$ is positive semi-definite. In order to isolate the phase and frequency errors, we choose the first sample index $n_0 = -P/N$, the off-diagonal terms of (2.8) can be set to zero and the frequency and phase estimation performance of each algorithm can be evaluated independently.

2.2 FFT-Based Maximum Likelihood Estimation

The maximum likelihood (ML) frequency estimator given the observation (2.1) is [8]

$$\hat{\omega}_{ML} = \arg\max_{\omega\in\Omega} |A(\omega)| \tag{2.9}$$

where $\Omega \subseteq [0,\infty)$, and

$$A(\omega) = \frac{1}{N}\sum_{n=0}^{N-1} z[n]\exp(-jn\omega T). \tag{2.10}$$

After finding $\hat{\omega}_{ML}$, the phase estimate is computed by using the follow equation

$$\hat{\theta}_{ML} = \angle\{\exp(-j\hat{\omega}_{ML}t_0)A(\hat{\omega}_{ML})\} \tag{2.11}$$

where $t_0 := n_0T$. The amplitude is estimated by using

$$\hat{b_0}_{ML} = |A(\hat{\omega}_{ML})| \tag{2.12}$$

A well-known numerical method to locate $\hat{\omega}_{ML}$ is given in [8]. In order to locate $\hat{\omega}_{ML}$, they use the discrete Fourier transform (DFT) to find ω which approximately maximizes $|A(\omega)|$. In fact, the M-point DFT is a sampled version of $A(\omega)$ at frequencies $\omega = \frac{2\pi k}{MT}$ for $k := 0, ..., M-1$. Usually, the Fast Fourier transform (FFT) is used to compute $A(\frac{2\pi k}{MT})$

for $k := 0, ..., M-1$. Then they select the index k at which $A(\frac{2\pi k}{MT})$ attains its maximum magnitude. In other words, (2.9) is maximized over a discrete set rather than Ω. Then the frequency estimate can be computed as

$$\hat{\omega}_{Rife} = \frac{2\pi k}{MT}. \tag{2.13}$$

After finding $\hat{\omega}_{Rife}$, we can use (2.11) and (2.12) to find the phase and amplitude estimates.

As presented in [8], when none of the frequency, phase, and amplitude is known, the proposed estimator computes the frequency estimate as first. Then the phase and amplitude estimates are computed based on the frequency estimate. Therefore, the accuracy of the frequency estimator is very important. With a poor frequency estimator, it is difficult to have a good phase and amplitude estimates. Thus, in [8], Rife and Boorstyn proposed a two-part search routine. The first part is called the coarse search which uses the FFT method discussed above to locate the $\hat{\omega}$. The second part locates the local maximum closet to the value of ω picked out by the first part. This part is called the fine search. If the frequency estimate computed by coarse search is accurate, this procedure will locate the global maximum of $|A(\omega)|$ and thus the best approximate ML estimates. In [8], the Secant method is used for the fine search. However, few details are provided about how to implement it. In Section 3.3, a detailed study is provided about using the FFT and the Secant Method to implement the approximate maximum likelihood estimation.

2.3 Linear-Regression-Based Maximum Likelihood Estimation

In the previous section, the discussed maximum likelihood estimator uses the FFT to locate the estimate $\hat{\omega}_{ML}$ approximately, which maximizes the likelihood function $|A(\omega)|$. For the proposed method in [8], the estimator needs to process after receiving all the samples. It is not an efficient algorithm for the applications require low latency. In addition, the computational complexity of the FFT is $\mathcal{O}(M \log M)$, where M is the number of FFT points. This computation of the FFT will increase the latency of estimation as well. Nowadays,

more and more applications require a low-latency algorithm for computation. One of the well-known solutions is the linear-regression-based maximum likelihood estimation. For this category of estimation, the estimators find the phase of the received observation, i.e.

$$\phi[n] := \angle z[n] = \omega_0 nT + \theta_0, \tag{2.14}$$

and then apply the linear regression to $\{\phi[n]|n = n_0+0, n_0+1, \ldots n_0+N-1\}$ for estimating $\hat{\omega}$ and $\hat{\theta}$. Most of the proposed algorithms can attain the CRLB over a wide range of SNR values.

Tretter is the first person to present the idea of using phase samples to estimate frequency and phase [13]. The observed signal in (2.1) can be expressed as

$$z[n] = \{1 + v[n]\}\, b_0 \exp(j(w_0 nT + \theta_0)) \tag{2.15}$$

where

$$v[n] = \frac{1}{b_0} w[n] \exp(-j(w_0 nT + \theta_0)). \tag{2.16}$$

Let $v[n] = v_R[n] + jv_I[n]$, then

$$1 + v[n] = \left\{\{1 + v_R[n]\}^2 + v_I^2[n]\right\}^{1/2} \times \exp\left\{j \arctan \frac{v_I[n]}{1 + v_R[n]}\right\} \tag{2.17}$$

For high SNR, when $1 \gg v_R[n]$ and $1 \gg v_I[n]$, we can write

$$\{(1 + v_R^2[n])^2 + v_I^2[n]\}^{1/2} \simeq 1 \tag{2.18}$$

and

$$1 + v[n] \simeq \exp\{j \arctan v_I[n]\} \simeq \{jv_I[n]\}. \tag{2.19}$$

Hence, $1 + v[n] \simeq jv_I[n]$.

This approximation can then be plugged back into (2.15) to write

$$z[n] \simeq b_0 \exp\left\{j(w_0 nT + \theta_0 + v_I[n])\right\}. \tag{2.20}$$

All the required information is contained in the phase angle of (2.20)

$$\phi[n] := \omega_0 nT + \theta_0 + v_I[n]. \tag{2.21}$$

(2.21) can be computed by applying a phase unwrapping algorithm and using an arctangent operation, i.e. $\phi[n] = \text{unwrap}\{\arctan\{z[n]\}\}$. The parameters $\hat{\omega}_{tretter}$ and $\hat{\theta}_{tretter}$ can be calculated by least squares (LS) regression, i.e.

$$\begin{aligned}\begin{bmatrix}\hat{\omega}_{tretter} \\ \hat{\theta}_{tretter}\end{bmatrix} &= \frac{12}{T^2N^2(N^2-1)} \\ &\times \begin{bmatrix} N & -T(Nn_0+P) \\ -T(Nn_0+P) & T^2(Nn_0^2+2n_0P+Q)\end{bmatrix} \\ &\times \begin{bmatrix}\sum_{n=n_0}^{n_0+N-1} nT\phi(n) & \sum_{n=n_0}^{n_0+N-1}\phi(n)\end{bmatrix}.\end{aligned} \tag{2.22}$$

As discussed in [13], Tretter's estimator can only achieve the CRLB as a sufficiently high SNR.

Compared with the Rife's MLE, Tretter's estimator does not require to use the FFT. Instead, linear regression is applied to the phase of each sample. Therefore, the computational complexity is reduced. In addition, based on (2.22), the proposed estimator can be implemented in sequence. This advantage makes it possible to implement Tretter's estimator with low latency. However, the phase unwrapping is not an efficient operation, which may increase the computational complexity. In addition, the phase unwrapping algorithm may fail at low SNR [14].

In [14], Kay proposed a similar algorithm which still uses the phase of each sample for estimation, but avoids the phase unwrapping by using the differenced phase of two adjacent

points. From (2.20) and (2.21), we have

$$\begin{aligned}\Delta[n+1] &= \angle z[n+1] - \angle z[n] \\ &= \phi[n+1] - \phi[n] \\ &= \omega_0 T + v_I[n+1] - v_I[n] \end{aligned} \tag{2.23}$$

When SNR is high, we know $1 \gg v_I[n]$. Therefore, $v_I[n+1] - v_I[n] \simeq 0$. Thus, as long as SNR is high enough, we can apply the least square regression again to (2.23) to estimate $\hat{\omega}$. In [14], an alternative approach is introduced. Since $\angle z[n+1] - \angle z[n] = \angle \{z[n+1]z^*[n]\}$, in addition the noise $v_I[n]$ is coloured noise, the standard weighted least-squares theory [18] leads to a weighted average estimate, given by

$$\hat{\omega}_{kay} = \frac{1}{(N-1)T} \sum_{n=0}^{N-2} p[n] \arg \{z^*[n]z[n+1]\} \tag{2.24}$$

where

$$p[n] = \frac{6(n+1)\,[N-(n+1)]}{N(N^2-1)}. \tag{2.25}$$

As shown in [14], Kay's estimator can only attain the CRLB at high SNR. The unweighed Kay's estimator is very close to the CRLB but not attain it. In addition, compared with the FFT-based MLE, the threshold of Kay's estimator is larger.

In [14], the estimator for $\hat{\theta}_{kay}$ is not given. Since the least square estimation is equivalent to ML method when the estimate is unbiased, hence, we use (2.11) for phase estimation, i.e.

$$\hat{\theta}_{kay} = \arg \{\exp(-j\hat{\omega}_{kay} n_0) A(\hat{\omega}_{kay})\} \tag{2.26}$$

Compared with Tretter's estimate, Kay's estimate does not require to use the phase unwrapping algorithm, which makes the computation more efficient. In [13], the author does not give a specific phase unwrapping algorithm for Tretter's method. In [19], the author shows that Kay's method is equivalent to Tretter's method with Itoh's phase unwrapping algorithm.

Chapter 3

FFT-Based Phase and Frequency Estimation

The Maximum Likelihood Estimator (MLE) has many applications in the estimation of unknown parameters of a complex exponential because it is asymptotically efficient [18], i.e. it achieves the CRLB as the number of samples becomes large. Many articles, such as [8], [10], and [18], have discussed the methods to search for the value of the unknown variable which maximizes the likelihood function (3.2). However, all of the methods are required to know the entire signal before estimation. Therefore, the MLE is not the primary choice for the applications that require fast estimation or computationally-constrained applications.

In some applications, the MLE can be used to analyse and to evaluate the performance of other estimation techniques when the "truth" is not known. In order to do that, the Mean Squared Error (MSE) of the ML frequency estimator, which is defined as $\mathrm{E}[(\omega_0 - \hat{\omega})^2]$, and the MSE of ML phase estimator, which is defined as $\mathrm{E}[(\theta_0 - \hat{\theta})^2]$, where E denotes the expectation, have to be significantly better than the estimator of interest. The best any unbiased estimator can do is attaining the CRLB over a wide range of SNR values, especially at high SNR.

Figure 3.1 shows an application using the MLE to analyse a hybrid Phase Locked Loop (PLL) algorithm for the carrier synchronization of the distribute beamforming [17, 20]. In this test, two software-defined radios (SDRs) are used as transmitter and receiver separately.

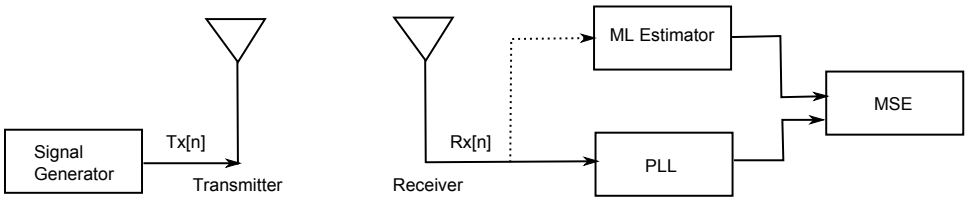

Figure 3.1: Application of Maximum Likelihood Estimation.

The Signal Generator produces the sinusoidal beacon $Tx[n] = \cos(nT\omega + \theta)$, where ω is known frequency and θ is known phase offset. The transmitter reconstructs the beacon $Tx[n]$ and then sends it to the receiver. After receiving the beacon, the receiver samples the received signal and then sends the sampled signal $Rx[n] = \cos(nT\tilde{\omega} + \tilde{\theta})$ to the PLL estimator. After convergence, the PLL estimator outputs the frequency and phase estimates, $\hat{\omega}_{PLL}$ and $\hat{\theta}_{PLL}$ respectively. Finally, we compute the frequency MSE, $\mathrm{E}[(\tilde{\omega} - \hat{\omega}_{PLL})^2]$, and the phase MSE, $\mathrm{E}[(\tilde{\theta} - \hat{\theta}_{PLL})^2]$. In this test, we do not know the true $\tilde{\omega}$ and $\tilde{\theta}$. Therefore, we need to have an estimator with high accuracy to compute the best possible true values of $\tilde{\omega}$ and $\tilde{\theta}$. As discussed in [18], the maximum likelihood estimator is asymptotically efficient. Therefore, this estimator can be used to compute $\tilde{\omega}$ and $\tilde{\theta}$ when the number of samples is large. As shown in Figure 3.1, the received signal $Rx[n]$ is also sent to the ML estimator. The outputs of the ML estimator are $\hat{\omega}_{ML}$ and $\hat{\theta}_{ML}$. Due to its high accuracy, we assume $\tilde{\omega} = \hat{\omega}_{ML}$ and $\tilde{\theta} = \hat{\theta}_{ML}$. Hence, now we have enough knowledge to evaluate the performance of the hybrid PLL.

The example presented above is not singular. There are many other applications requiring a high accurate estimator. However, as discussed in Section 2.2, the frequency estimate $\hat{\omega}_{ML}$ requires numerical methods to compute. Therefore, five approximate maximum likelihood estimators are proposed and discussed in this chapter. They are:

1. Approximate maximum likelihood estimator using Fast Fourier Transform (FFT) and no post-processing (FFT estimator)

2. Approximate maximum likelihood estimator using FFT and quadratic interpolation (FFT-Quad estimator)

3. Approximate maximum likelihood estimator using FFT and Secant method (FFT-Secant estimator)

4. Approximate maximum likelihood estimator using FFT and Newton's method (FFT-Newton estimator)

5. Approximate maximum likelihood estimator using FFT and bisection method (FFT-bisection estimator)

All the estimation techniques are based on the FFT. The only difference among them is the post-processing after the FFT. For instance, the FFT-Quad estimator uses the quadratic interpolation to compute the unknown parameters after FFT. The FFT-Secant estimator and the FFT-Newton estimator use different root-finding algorithms to estimate the unknown parameters. For each estimator, we will evaluate the frequency and phase MSEs as a function of SNR. Also, we will discuss the computational complexity of each algorithm. At the end of this chapter, we will summarize the performance and finally give a guideline for choosing the algorithm for approximate MLE.

3.1 Approximate Maximum Likelihood Estimation using Fast Fourier Transform and No Post-Processing

As discussed in Section 2.2, the frequency estimate of the maximum likelihood estimator is

$$\hat{\omega}_{ML} = \arg\max_{\omega\in\Omega}|A(\omega)| \tag{3.1}$$

where

$$A(\omega) = \frac{1}{N}\sum_{n=0}^{N-1} z[n]\exp(-jn\omega T) \tag{3.2}$$

and $\Omega \subseteq [0, \infty)$.

In [8] and [9], Rife presents an approximate method which uses the Discrete Fourier Transform (DFT) to approximately locate the value of ω, which maximizes $|A(\omega)|$. In fact, the M-point DFT is a sampled version of $A(\omega)$ at frequencies $\omega = \frac{2\pi k}{MT}$ for $k \in \mathcal{K}$ and

$\mathcal{K} := \{0, 1, \ldots, M-1\}$. Usually, we use the Fast Fourier Transform (FFT) to compute $A(\frac{2\pi k}{MT})$ for $k \in \mathcal{K}$ [8]. Hence, (3.1) becomes to

$$\hat{k} = \arg\max_{k \in \mathcal{K}} \left| A\left(\frac{2\pi k}{MT}\right) \right|. \tag{3.3}$$

After finding $\hat{k}$, we can form an approximate estimate $\hat{\omega}_{ML}$, $\hat{\omega}_{FFT}$, by using (3.4)

$$\hat{\omega}_{FFT} = \frac{2\pi \hat{k}}{MT}, \tag{3.4}$$

which requires an exhaustive search, but over a finite set of discrete points. Then the approximate phase $\hat{\theta}_{FFT}$ can be estimated by using (3.5)

$$\hat{\theta}_{FFT} = \angle\{\exp(-j\hat{\omega}_{FFT} t_0) A(\hat{\omega}_{FFT})\} \tag{3.5}$$

where $t_0 := n_0 T$.

In order to show how the FFT ML estimator works, we present a numerical example here. For example, an observation without noise has $N = 513$ samples, sampling period $T = 1$ second, first sample index $n_0 = -256$, the frequency $\omega_0 = 0.1 \times 2\pi = 0.6283$ rad/second (rad/s) and the phase offset $\theta_0 = 0$ rad. We use a $M = 2^{12}$ points FFT to estimate the frequency. As shown in Figure 3.2, the peak is located at $\hat{k} = 410$. After calculation by using (3.4), we find $\hat{\omega}_{FFT} = 0.6289$ rad/s, and $\hat{\theta}_{FFT} = 0$ rad, which is as same as θ_0. The squared frequency estimate error is 3.7650×10^{-7}.

If we increase M to 2^{14}, the peak is located at $\hat{k} = 1638$. Then $\hat{\omega}_{FFT} = 0.6282$ rad/s, which is much closer to ω_0, and $\hat{\theta}_{FFT} = 0$ rad. The squared error of the frequency estimate is 2.3531×10^{-8}, which is improved by 10 times from the previous case. If we increase M to 2^{18}, the peak is located at $\hat{k} = 26214$. Then $\hat{\omega}_{FFT} = 0.6283$ rad/s and $\hat{\theta}_{FFT} = 0$ rad. The squared frequency estimate error is again improved to 9.1918×10^{-11}.

Figure 3.3 and Figure 3.4 show the MSE of the frequency and phase estimates of the FFT MLE respectively as a function of SNR $:= 10\log_{10}(b_0^2/2\sigma^2)$ for five values of M, $M = 2^{10}$, $M = 2^{12}, M = 2^{14}, M = 2^{16}$, and $M = 2^{20}$, for a complex observation with AWGN. All the results assume an observation with $N = 513$, sampling period $T = 1$ second, and the first

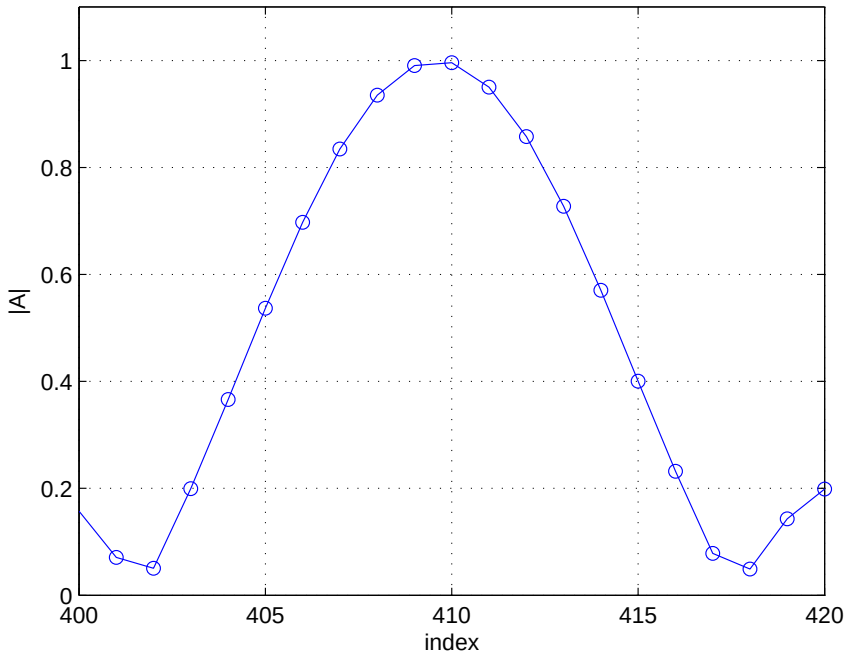

Figure 3.2: FFT MLE example with $M = 2^{12}$.

sample time $n_0 = -256$. The frequency is an independent random variable for a uniform distribution

$$\omega_0 \sim \mathcal{U}(0.09 \times 2\pi, 0.11 \times 2\pi). \tag{3.6}$$

The phase is an independent random variable for a uniform distribution

$$\theta_0 \sim \mathcal{U}(-\pi, \pi). \tag{3.7}$$

500 realizations of the random complex exponential signal using (2.1) and AWGN are generated with fixed $b_0 = 1$.

In Figure 3.3, the estimators with $M = 2^{10}$ and $M = 2^{12}$ fail to track on the CRLB over the entire SNR range. The low-SNR threshold of all of the rest estimators is a SNR of about -9dB. The estimator with $M = 2^{14}$ closely tracks the CRLB up to a SNR of about 0dB. The estimator with $M = 2^{16}$ closely tracks the CRLB up to a SNR of about 12dB. For the estimator with $M = 2^{20}$, the estimation range is much wider than other four estimators.

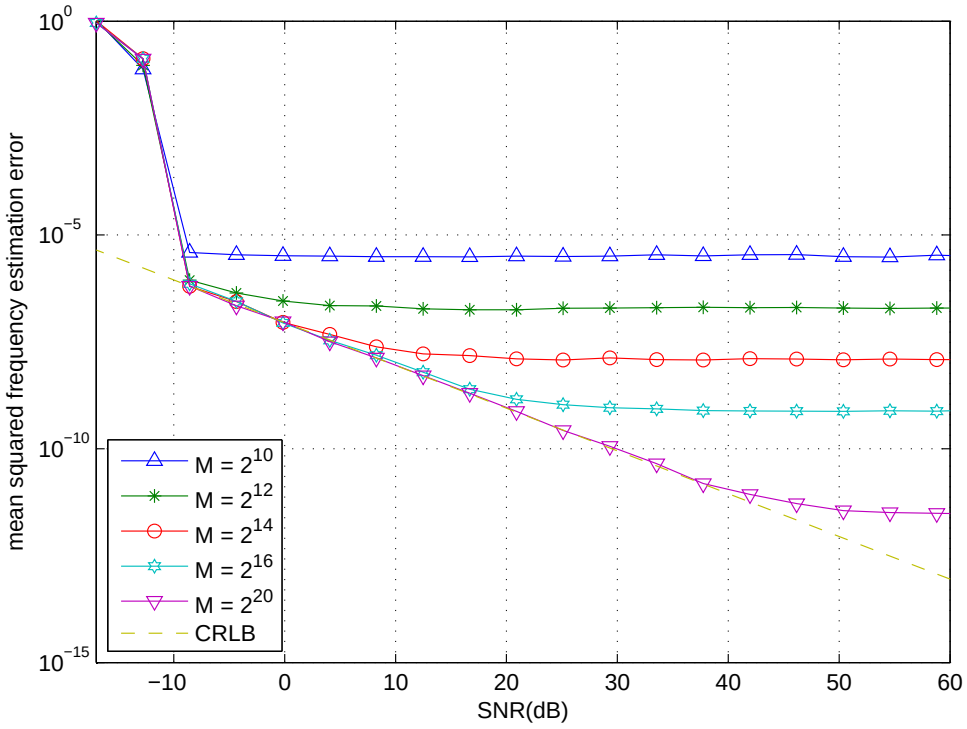

Figure 3.3: Mean squared frequency estimation error for complex observation of FFT MLE.

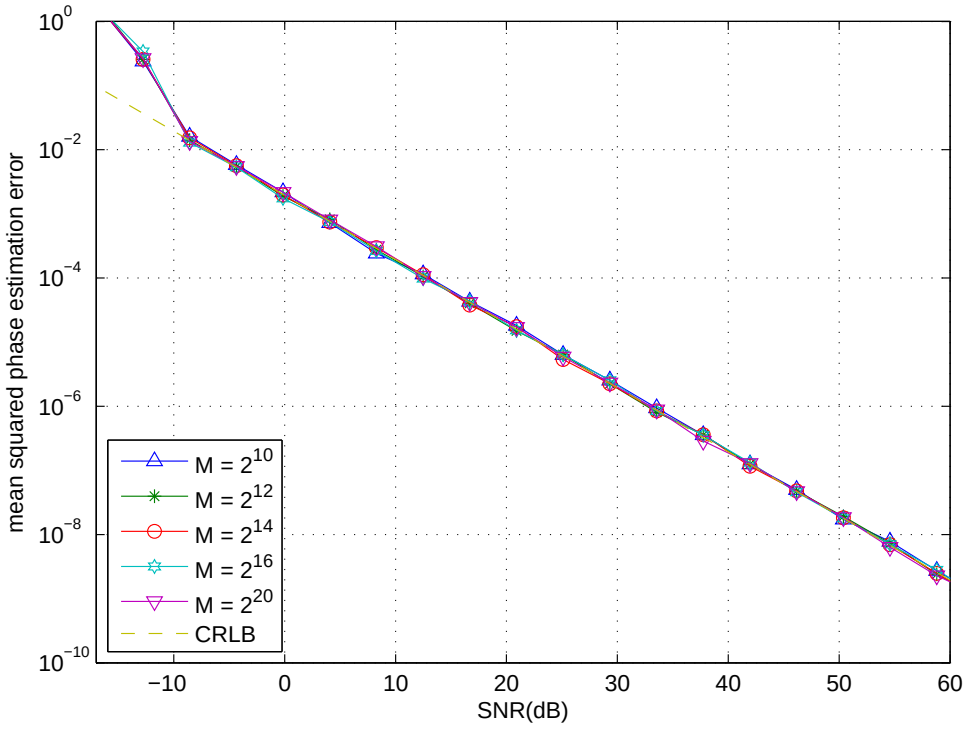

Figure 3.4: Mean squared phase estimation error for complex observation of FFT MLE.

It tracks the CRLB for up to a SNR of about 38dB. In Figure 3.4, which shows the phase estimation performance, all of the estimators can track the CRLB from -9dB to 60dB.

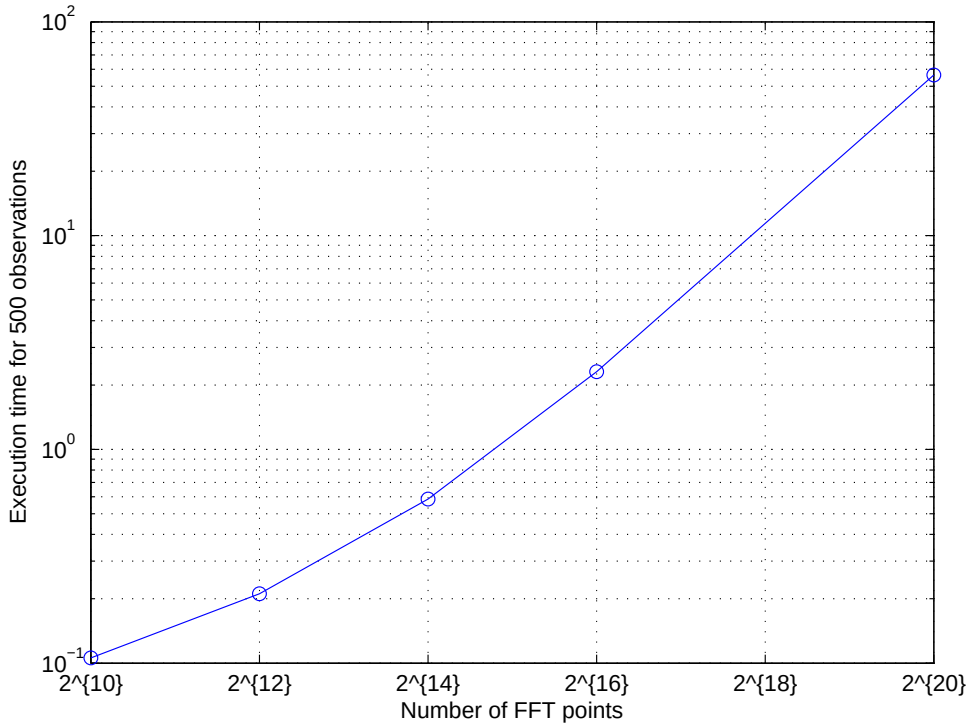

Figure 3.5: Total execution time for 500 complex observations of FFT MLE.

Figure 3.3 shows that an increase of the value of M can improve the estimation accuracy. However, when we increase M, the execution time also increases. Figure 3.5 shows the total execution time for 500 complex observations of the FFT MLE. As shown in the figure, the execution time increases when M increases. For the FFT MLE, the most time consumption part is the computation of FFT. The computational complexity of the FFT is $\mathcal{O}(M \log M)$. If we increase M from $M = 2^{12}$ to $M = 2^{14}$, the execution time will be increased by a factor of 4.67. Figure 3.5 shows that if we increase M from 2^{12} to 2^{20}, the execution time is increased by more than 100 times. One thing we should notice is since the Matlab uses the multiple threads computation automatically, the plot in Figure 3.5 does not exactly follow the ratio of $M \log M$. In addition, memory space is an issue when using a large value of M. In our simulation, we use double precision complex variable to store the output of one FFT computation. Assume each double precision complex variable requires 128 bits. Then for

a 2^{14} points FFT, we need 256 KB memory to store the values. For a 2^{20} points FFT, we need 32 MB memory to store the values. If we want to use single 2^{28} points FFT, we need to have a 4 GB memory to store the variables. Therefore, in order to achieve high accuracy, the FFT ML estimator requires a high performance computer with large memory space.

3.2 Approximate Maximum Likelihood Estimation using FFT and Quadratic Interpolation

Let us reconsider the FFT MLE example with $M = 2^{12}$ from Section 3.1. When $\omega_0 = 0.1 \times 2\pi = 0.6283$ rad/s, by using (3.8), we find that the peak of $|A(\omega)|$ is located between $k = 409$ and $k = 410$.

$$k_{ML} = \frac{\omega_0 MT}{2\pi} = \frac{0.2\pi \times 2^{12} \times 1}{2\pi} = 409.6 \tag{3.8}$$

However, the index of FFT cannot be an non-integer number. Therefore, in order to improve the accuracy, one approach is to interpolate between points near the peak of the FFT. In [12], the FFT magnitude is interpolated by quadratic polynomial to improve the estimation accuracy. For example, we find the peak of FFT locates at FFT index $\hat{k}$. A quadratic fit $y = a + bx + cx^2$ in the neighbourhood of the maximum can be computed given the frequencies $x \in \left\{ \frac{2\pi(\hat{k}-1)}{MT}, \frac{2\pi(\hat{k})}{MT}, \frac{2\pi(\hat{k}+1)}{MT} \right\}$ and FFT magnitudes $y = |A(x)|$. Then the peak of the quadratic fit, which is also the frequency estimate, is $\hat{\omega}_{Quad} = \frac{-b}{2c}$. Finally, we can estimate phase $\hat{\theta}_{Quad}$ by using (3.10). This method is described as the FFT-Quad MLE in this thesis.

We use the example inFigure 3.2 to demonstrate how the FFT-Quad MLE works. We use a $M = 2^{12}$ points FFT estimator to locate the peak at $\hat{k} = k_1 = 410$, as shown in Figure 3.6.

We can compute the quadratic polynomial coefficients a, b, c by solving the linear system

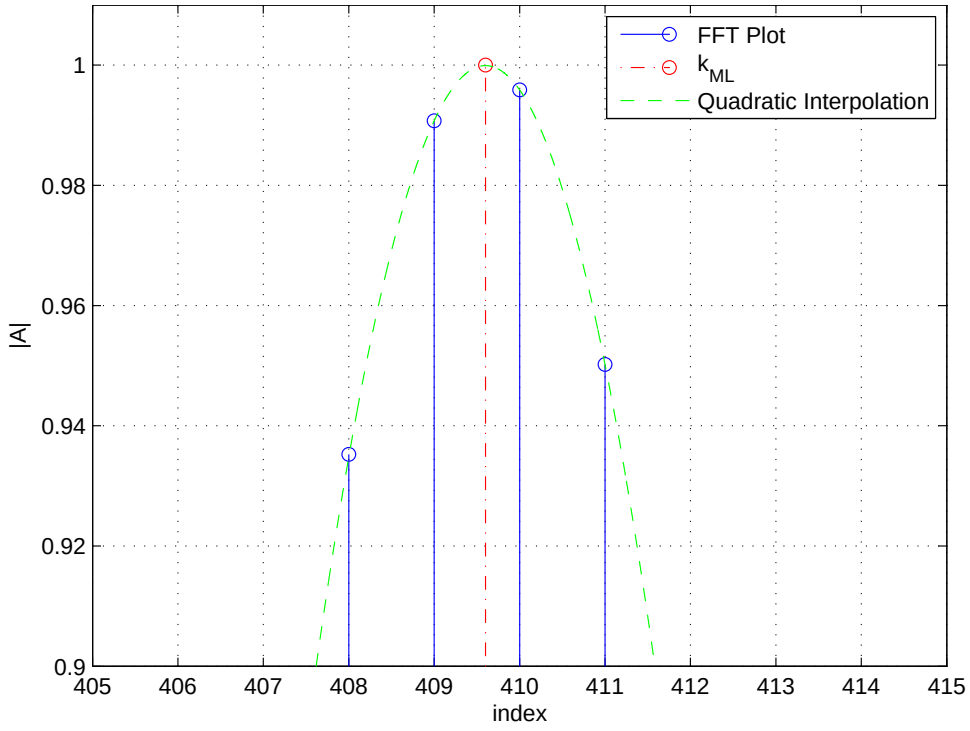

Figure 3.6: FFT-Quad MLE Example.

(3.9),

$$\begin{bmatrix} \left(\frac{2\pi(\hat{k}-1)}{MT}\right)^2 & \frac{2\pi(\hat{k}-1)}{MT} & 1 \\ \left(\frac{2\pi(\hat{k})}{MT}\right)^2 & \frac{2\pi(\hat{k})}{MT} & 1 \\ \left(\frac{2\pi(\hat{k}+1)}{MT}\right)^2 & \frac{2\pi(\hat{k}+1)}{MT} & 1 \end{bmatrix} \begin{bmatrix} c \\ b \\ a \end{bmatrix} = \begin{bmatrix} |A(\frac{2\pi(\hat{k}-1)}{MT})| \\ |A(\frac{2\pi(\hat{k})}{MT})| \\ |A(\frac{2\pi(\hat{k}+1)}{MT})| \end{bmatrix} \tag{3.9}$$

which gives an exact quadratic fit to the data. The matrix on the left is commonly referred as a Vandermonde matrix [21]. After solving the linear system in (3.9), we compute $\hat{\omega}_{Quad} = \frac{-b}{2c}$, which is the peak of the quadratic fit and also the frequency estimate.

In our example, the maximum found by the quadratic interpolation is $\hat{\omega}_{Quad} = 0.6283$. Then, we can estimate $\hat{\theta}_{Quad}$ by calculating (3.10)

$$\hat{\theta}_{Quad} = \angle\{\exp(-j\hat{\omega}_{Quad}t_0)A(\hat{\omega}_{Quad})\} = 0. \tag{3.10}$$

The associated squared error of the frequency estimate is 2.9591×10^{-12}. This squared

error is much smaller than that of the FFT estimator with the same M value and also smaller than that of the FFT estimator with $M = 2^{18}$.

Figure 3.7 and Figure 3.8 show the MSE of the frequency and phase estimates of the FFT-Quad ML respectively as a function of SNR for different values of M, for a complex observation with AWGN. All the results assume an observation with $N = 513$, sampling period $T = 1$ second, and the first sample time $n_0 = -256$. The frequency and phase are independent random variables for the uniform distributions (3.6) and (3.7) respectively. 500 realizations of the random complex exponential signal using (2.1) and AWGN are generated with fixed $b_0 = 1$. Since we have known that with the increase of the value of M, the accuracy of the estimation will be improved as well. Hence, we only consider about the cases with small values of M. Here we only consider $M = 2^{10}$, $M = 2^{12}$, and $M = 2^{14}$.

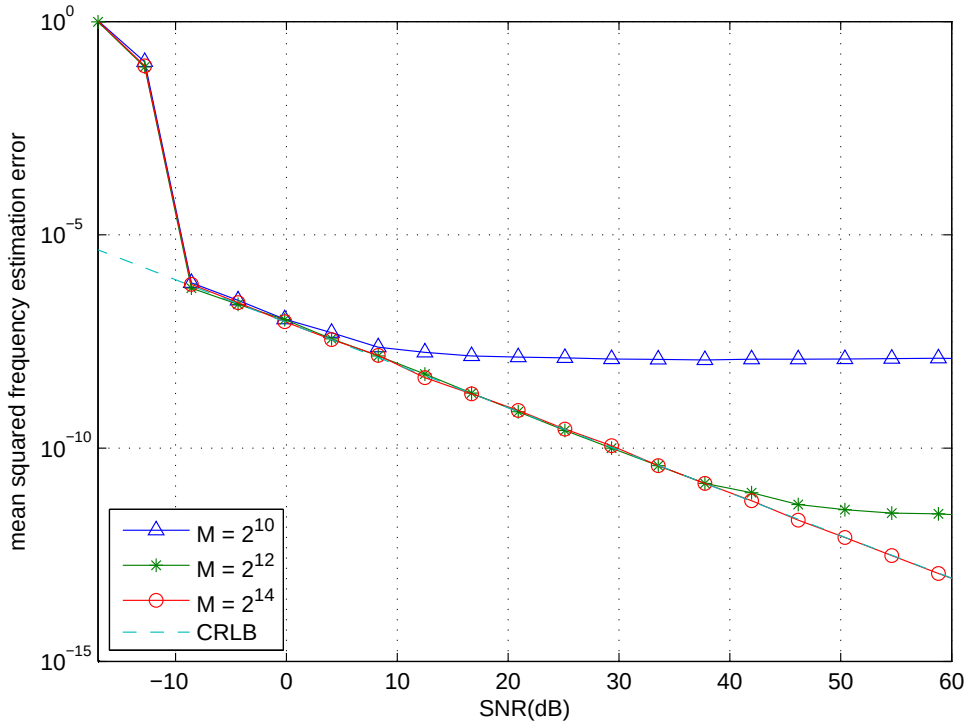

Figure 3.7: Mean squared frequency estimation error for complex observation of FFT-Quad MLE.

In Figure 3.7, all estimators have the same threshold, which is -9dB. However, the estimator with $M = 2^{10}$ fails to achieve the CRLB after 0dB. Compared with the FFT

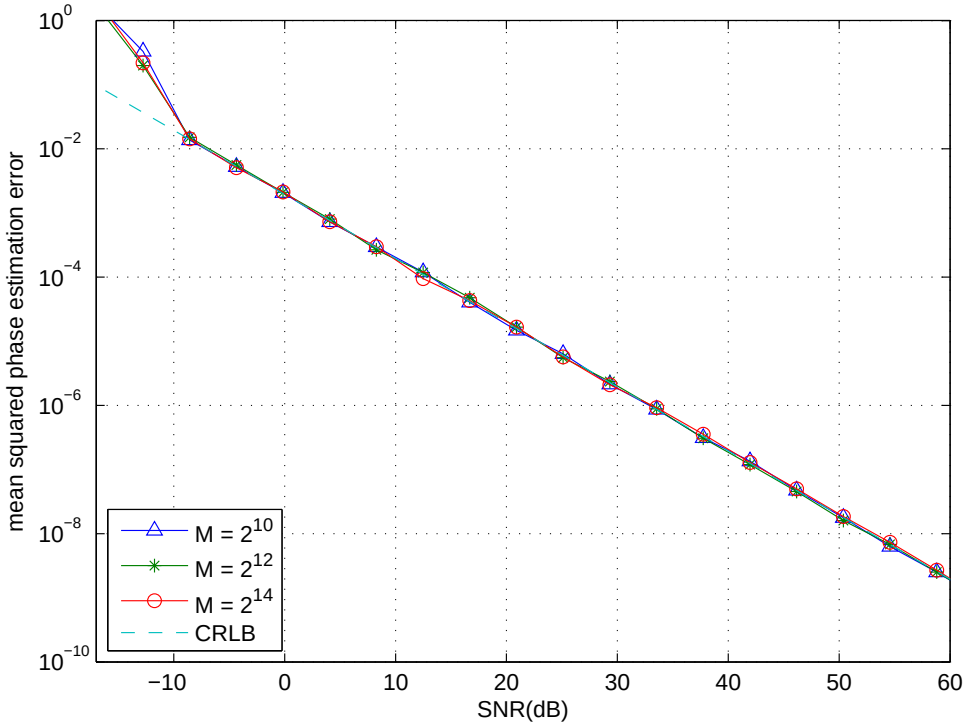

Figure 3.8: Mean squared phase estimation error for complex observation of FFT-Quad MLE.

MLEs, the estimators with $M = 2^{12}$ and $M = 2^{14}$ have a wider estimation range. The former one tracks the CRLB up to a SNR of about 38dB. And the latter one tracks the CRLB up to a high SNR of 60dB. Figure 3.8 shows that all of the phase estimators can attain the CRLB.

Compared with Figure 3.3 and Figure 3.4, Figure 3.7 and Figure 3.8 show that the estimation accuracy is improved by a significant level because of the quadratic interpolation post-processing. In order to form the Vandermonde matrix in (3.9), we need four multiplications. The computational complexity of matrix inversion via Gaussian Elimination is $\mathcal{O}(N^3)$ and the approximate number of operations is $2n^3/3$ [22]. Therefore, for a 3×3 matrix, the number of operations of matrix inversion is 18. For the matrix multiplication, the number of multiplication is 27. Therefore, finally, the computational complexity for FFT-Quad MLE is $\mathcal{O}(M \log M) + 45 \simeq \mathcal{O}(M \log M) + \mathcal{O}(1)$.

3.3 Approximate Maximum Likelihood Estimation using FFT and Secant Method

In [8] and [9], the maximum likelihood estimation search routine has two parts. The first search part is called the *coarse search*, which is the FFT MLE described in Section 3.1. The accuracy of the coarse search is strongly affected by the number of points of FFT, as discussed in Section 3.1. In order to improve the estimation accuracy, the coarse search is followed by a fine search. One example of a fine search is the FFT-Quad MLE presented in Section 3.2. In [8], another fine search approach was described: the Secant method. This section discusses the Secant method and its application to ML frequency estimation.

The Secant method is a iterative method used to find roots of a equation $f : \Re \to \Re$. The iteration formula is [23]

$$\begin{aligned} x^{(m)} &:= \frac{x^{(m-2)} f(x^{(m-1)}) - x^{(m-1)} f(x^{(m-2)})}{f(x^{(m-1)}) - f(x^{(m-2)})}, \quad m \geq 2 \\ &= x^{(m-1)} - f(x^{(m-1)}) \frac{x^{(m-1)} - x^{(m-2)}}{f(x^{(m-1)}) - f(x^{(m-2)})}. \end{aligned} \tag{3.11}$$

The two initial values, $x^{(0)}$ and $x^{(1)}$, are chosen to lie close to the desired root. As shown

in Figure 3.9, we want to find the root of $y = 4(x+1)(x-0.5)(-x+2)(-x+2.8)$ near $x = 0.5$. We picked up the initial point $x^{(0)} = 1.5$ and $x^{(1)} = -0.5$. Then we apply (3.11) to compute $x^{(2)}$ and continue. After two iterations, $x^{(3)}$ converges to a value which is very close to the root at $x = 0.5$.

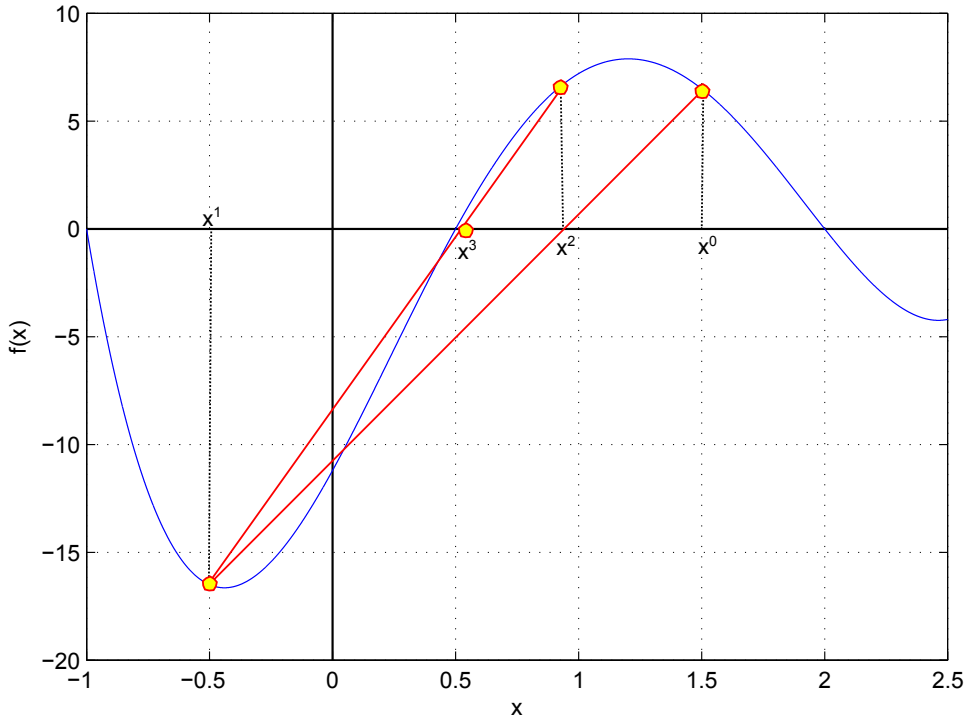

Figure 3.9: The Secant Method example with two iterations.

Finding ω which maximizes $|A(\omega)|$ is equivalent to finding a root of the first order derivative of $|A(\omega)|$ or a root of the first order derivative of $|A(\omega)|^2$. The DFT in (2.10) can be written in rectangular form as:

$$A(\omega) := B(\omega) + jC(\omega) \tag{3.12}$$

where

$$B(\omega) := \frac{1}{N}\sum_{n=0}^{N-1} \beta_n \cos(n\omega T) + \gamma_n \sin(n\omega T), \tag{3.13}$$

$$C(\omega) := \frac{-1}{N}\sum_{n=0}^{N-1} \beta_n \sin(n\omega T) - \gamma_n \cos(n\omega T), \tag{3.14}$$

and where

$$\beta_n = \Re\{z[n]\} \text{ and} \tag{3.15}$$

$$\gamma_n = \Im\{z[n]\}. \tag{3.16}$$

Hence,

$$F(\omega) := |A(\omega)|^2 = A(\omega)A^*(\omega) = B^2(\omega) + C^2(\omega). \tag{3.17}$$

Now, we have

$$F'(\omega) = \frac{dF}{d\omega} := 2B\frac{dB}{d\omega} + 2C\frac{dC}{d\omega} \tag{3.18}$$

where

$$\frac{dB}{d\omega} := \frac{T}{N}\sum_{n=0}^{N-1} n(-\beta_n \sin(Tn\omega) + \gamma_n \cos(Tn\omega))$$

$$\frac{dC}{d\omega} := \frac{-T}{N}\sum_{n=0}^{N-1} n(\beta_n \cos(Tn\omega) + \gamma_n \sin(Tn\omega))$$

Therefore, (3.11) becomes

$$\hat{\omega}^{(m)} = \hat{\omega}^{(m-1)} - F'(\hat{\omega}^{(m-1)})\frac{\hat{\omega}^{(m-1)} - \hat{\omega}^{(m-2)}}{F'(\hat{\omega}^{(m-1)}) - F'(\hat{\omega}^{(m-2)})}. \tag{3.19}$$

The stopping criteria for the Secant Method includes three conditions [22]:

1. $|\hat{\omega}^{(m)} - \hat{\omega}^{(m-1)}| \leq \epsilon$, where ϵ is the error tolerance
2. $F'(\hat{\omega}^{(m)}) = F'(\hat{\omega}^{(m-1)})$
3. $F'(\hat{\omega}^{(m)}) = 0$.

When any of them is satisfied, the iteration will stop and output the latest $\hat{\omega}^{(m)}$, which is the approximate maximum likelihood frequency estimate $\hat{\omega}_{Secant}$. The phase estimate is then

$$\hat{\theta}_{Secant} = \angle\{\exp(-j\hat{\omega}_{Secant}t_0)A(\hat{\omega}_{Secant})\}. \tag{3.20}$$

The algorithm of the Secant method is described in Algorithm 1.

Algorithm 1 The Secant Method

Import $\hat{\omega}^{(0)}, \hat{\omega}^{(1)}$
$\Delta = 1, n = 1$
while $|\Delta| \geq \epsilon$ **do**
  $n = n + 1$
  $\Delta = F^{'}(\hat{\omega}^{(m-1)}) \frac{\hat{\omega}^{(m-1)} - \hat{\omega}^{(m-2)}}{F^{'}(\hat{\omega}^{(m-1)}) - F^{'}(\hat{\omega}^{(m-2)})}$
  $\hat{\omega}^{(m)} = \hat{\omega}^{(m-1)} - \Delta$
  if $F^{'}(\hat{\omega}^{(m-1)}) == F^{'}(\hat{\omega}^{(m)})$ OR $F^{'}(\hat{\omega}^{(m)}) == 0$ **then**
    BREAK;
  end if
end while
return $\hat{\omega}^{(m)}$

We use the example presented in Section 3.1 to show how the FFT-Secant MLE works. As shown in Figure 3.6, the peak is located at $k = 410$. Then we pick up the neighbourhood points and assign them to be $\hat{\omega}^{(0)}$ and $\hat{\omega}^{(1)}$. Thus, $\hat{\omega}^{(0)} = 2\pi \times 409/(MT)$ and $\hat{\omega}^{(1)} = 2\pi \times 411/(MT)$. Now we start the iteration. As shown in Figure 3.10, the estimator only takes two steps to converge within $\epsilon = 10^{-4}$. The iteration finds $\hat{\omega}_{Secant} = 0.6284$. Since the error tolerance is a parameter that we control, we can ensure that the squared error of frequency estimate is less than 10^{-8}. In fact, the squared error of frequency estimate is 1.8948×10^{-9}. When we increase ϵ to 10^{-6}, the iteration takes 4 steps to converge.

Figure 3.11 and Figure 3.12 show the MSE of the frequency and phase estimators of the FFT-Secant MLE respectively as a function of SNR for three values of M, $M = 2^{10}$, $M = 2^{12}$, and $M = 2^{14}$, for a complex observation with AWGN. All the results assume an observation with $N = 513$, sampling period $T = 1$ second, and the first sample time $t_0 = -256$. The frequency and the phase are independent random variables for uniform distributions given by (3.6) and (3.7) respectively. 500 realizations of the complex exponential signal using (2.1) and AWGN are generated with fixed $b_0 = 1$. The error tolerance ϵ is set to 10^{-4} for the iteration. The initial values are $\hat{\omega}^{(0)} = (k-1)2\pi/(MT)$ and $\hat{\omega}^{(1)} = (k+1)2\pi/(MT)$, where k is the peak index estimated by the coarse search.

In Figure 3.11, the thresholds of the estimators with $M = 2^{12}$ and $M = 2^{14}$ are both about a SNR of -9dB. For the estimator with $M = 2^{12}$, it tracks the CRLB up to a SNR about 35dB, which is better than the FFT-Quad estimator with the same value of M. The

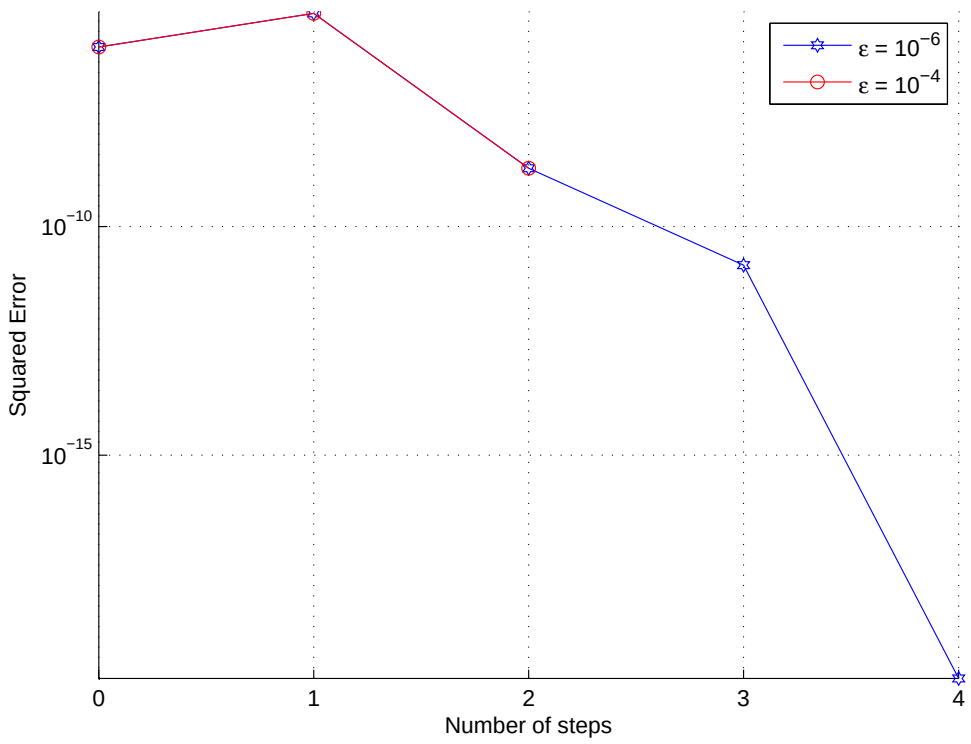

Figure 3.10: FFT-Secant MLE example.

FFT-Secant estimator with $M = 2^{14}$ can achieve the CRLB up to 60dB. As shown in Figure 3.12, since the FFT-Secant MLE has a bad frequency estimation when $M = 2^{10}$, the phase estimation also fails to attain the CRLB.

The performance of the iteration method is affected by the value ϵ. In Figure 3.13 and Figure 3.14, we reduce ϵ from 10^{-4} to 10^{-6}. In Figure 3.13, the thresholds of the FFT-Secant estimators with $M = 2^{12}$ and $M = 2^{14}$ remain the same. However, the FFT-Secant estimator with $M = 2^{12}$ can track the CRLB up to 60dB.

In both cases, the estimator with $M = 2^{10}$ does not converge. Therefore, it is worth discussing the convergence condition. Assume $\hat{\omega}^{(0)} < \hat{\omega}^{(1)}$. Because $F : [\hat{\omega}^{(0)}, \hat{\omega}^{(1)}] \to \Re$ and $F \in \mathcal{C}^2\{[\hat{\omega}^{(0)}, \hat{\omega}^{(1)}]\}$, therefore, $F'(\omega)$ is a continuous function on the interval $[\hat{\omega}^{(0)}, \hat{\omega}^{(1)}]$. In addition, we pick up $\hat{\omega}^{(0)}$ and $\hat{\omega}^{(1)}$ from the left side and the right side of the peak point, which is located by the coarse search. Hence, we know $F^{'}(\hat{\omega}^{(0)})F^{'}(\hat{\omega}^{(1)}) < 0$. According to theorem of zeros for continuous functions, we can ensure that there exists $\tau \in (\hat{\omega}^{(0)}, \hat{\omega}^{(1)})$ such that $F'(\tau) = 0$. In addition, since $F(\tau)$ is the maximum, therefore, $F^{''}(\tau) \neq 0$.

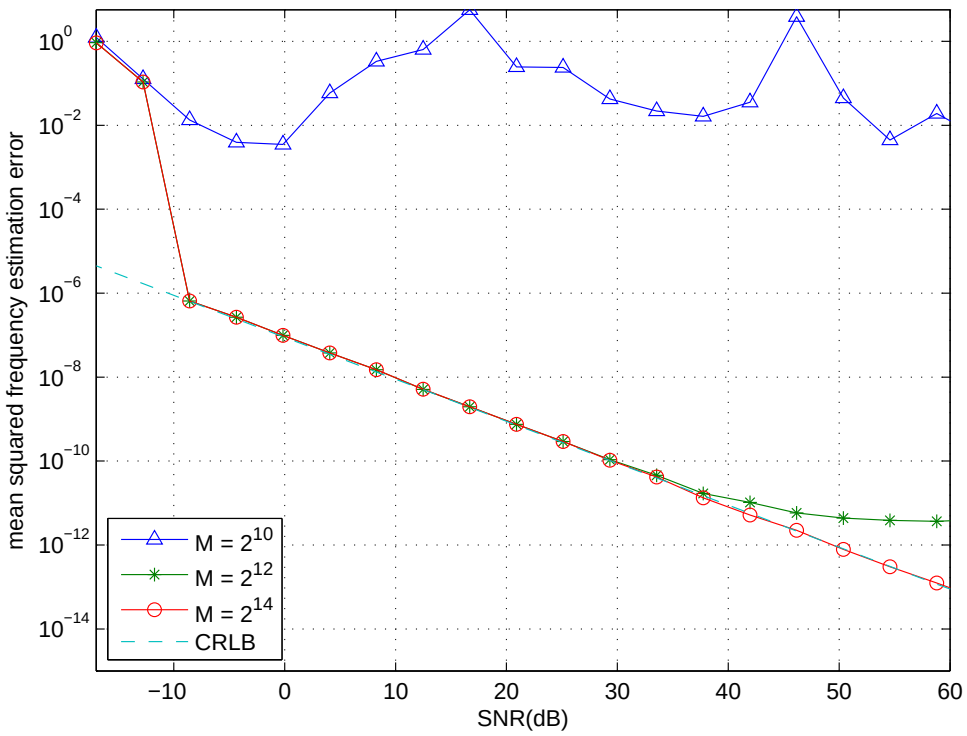

Figure 3.11: Mean squared frequency estimation error for complex signal of FFT-Secant MLE with $\epsilon = 10^{-4}$.

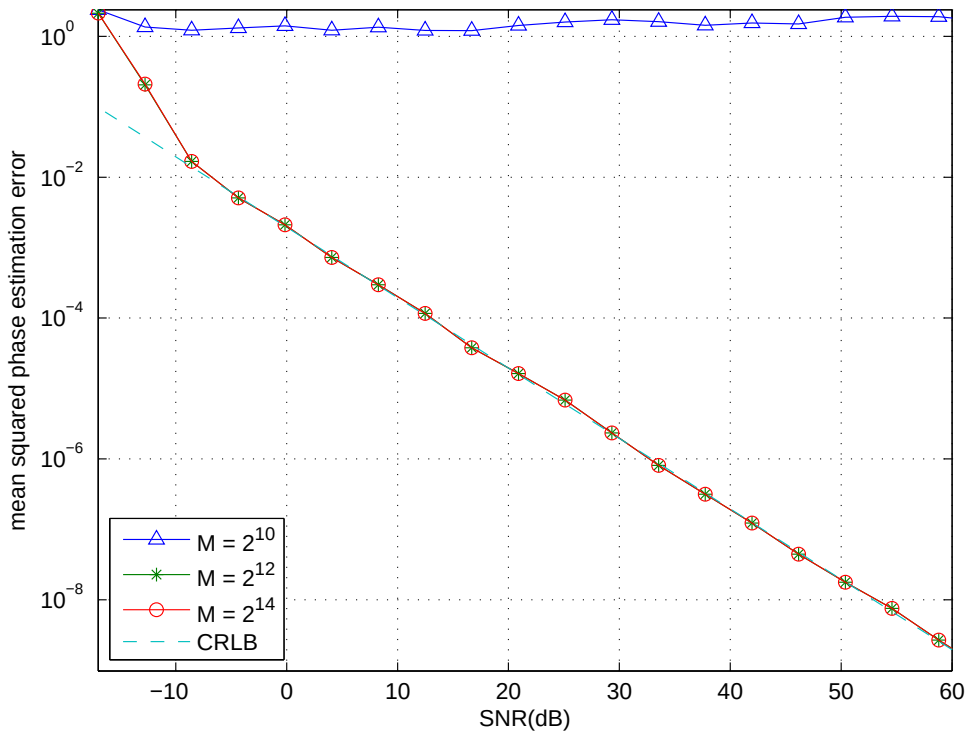

Figure 3.12: Mean squared phase estimation error for complex signal of FFT-Secant MLE with $\epsilon = 10^{-4}$.

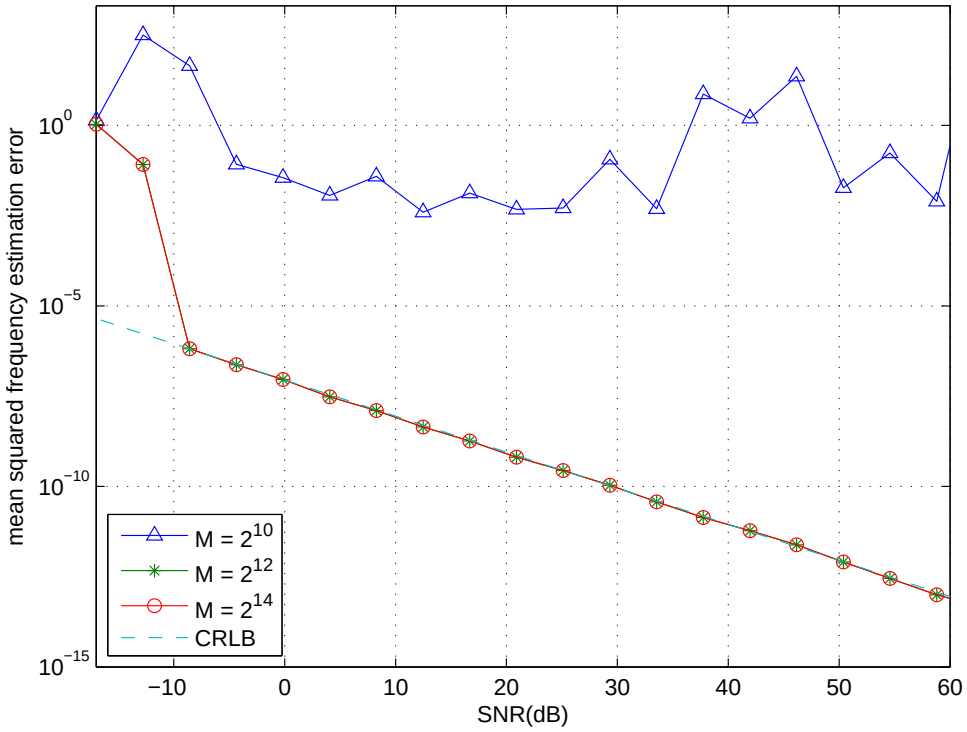

Figure 3.13: Mean squared frequency estimation error for complex signal of FFT-Secant MLE with $\epsilon = 10^{-6}$.

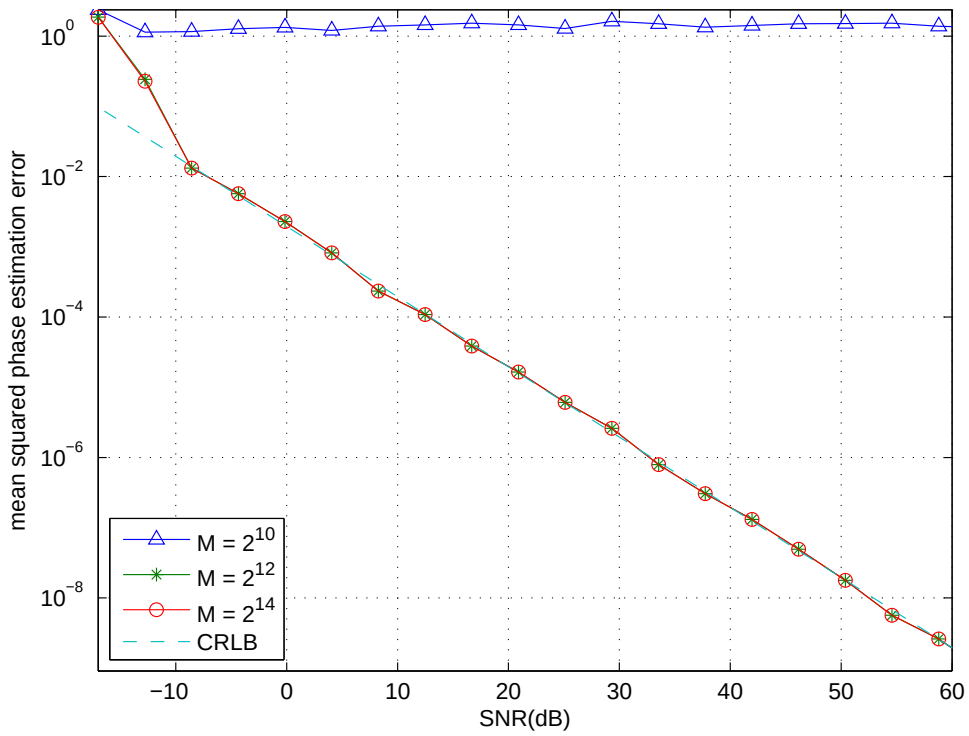

Figure 3.14: Mean squared phase estimation error for complex signal of FFT-Secant MLE with $\epsilon = 10^{-6}$.

Therefore, if the FFT gives the points which are sufficiently close to τ, then we can ensure the convergence [22]. The reason that the estimator with $M = 2^{10}$ does not converge is the initial points we pick up are not close enough to the root. It also explains why when we increase the value of M to 2^{12}, the estimator can attain the CRLB for a wide range of SNR values. In addition, when

$$F^{'}(\hat{\omega}^{(m-1)})\frac{\hat{\omega}^{(m-1)} - \hat{\omega}^{(m-2)}}{F'(\hat{\omega}^{(m-1)}) - F'(\hat{\omega}^{(m-2)})} \simeq 0, \tag{3.21}$$

the FFT-Secant MLE will not converge either.

In summary, compared with the FFT ML estimator and the FFT-Quad ML estimator, the estimation accuracy of the FFT-Secant ML estimator is improved significantly for small values of M. With a suitable error tolerance, the FFT-Secant ML estimator can track the CRLB over a wide range of SNR with a relatively coarse FFT, as shown in Figure 3.14.

3.4 Approximate Maximum Likelihood Estimation using FFT and Newton's Method

Newton's Method, also known as Newton-Raphson Method, is another method used to find the root of the function $f : \Re \to \Re$. The iterative formula for Newton's method is [22]:

$$x^{(m)} = x^{(m-1)} - \frac{f(x^{(m-1)})}{f^{'}(x^{(m-1)})} \quad \forall m \geq 1 \tag{3.22}$$

where $f^{'}(x)$ is the first order derivative of the function f respect to the variable x. Rather than two initial values in the Secant method, only one is needed in Newton's method.

In order to apply Newton's method for finding a root of the first order derivative of $F^{'}(\omega)$(3.18), we need to compute $F^{''}(\omega) = \frac{d^2F}{d\omega^2}$. We have

$$\frac{d^2F}{d\omega^2} = 2\left(\frac{dB}{d\omega}\right)^2 + 2B\frac{d^2B}{d^2\omega} + 2\left(\frac{dC}{d\omega}\right)^2 + 2C\frac{dC^2}{d^2\omega} \tag{3.23}$$

where

$$\frac{dB}{d\omega} := \frac{T}{N}\sum_{n=0}^{N-1} n(-\beta_n \sin(Tn\omega) + \gamma_n \cos(Tn\omega)), \tag{3.24}$$

$$\frac{dC}{d\omega} := \frac{-T}{N}\sum_{n=0}^{N-1} n(\beta_n \cos(Tn\omega) + \gamma_n \sin(Tn\omega)), \tag{3.25}$$

$$\frac{d^2B}{d\omega^2} := \frac{T^2}{N}\sum_{n=0}^{N-1} n^2(-\beta_n \cos(Tn\omega) - \gamma_n \sin(Tn\omega)), \text{and} \tag{3.26}$$

$$\frac{d^2C}{d\omega^2} := \frac{T^2}{N}\sum_{n=0}^{N-1} n^2(-\beta_n \sin(Tn\omega) + \gamma_n \cos(Tn\omega)). \tag{3.27}$$

Then the iterative formula becomes to

$$\hat{\omega}^{(m)} = \hat{\omega}^{(m-1)} - \frac{F^{'}(\hat{\omega}^{(m-1)})}{F^{''}(\hat{\omega}^{(m-1)})}. \tag{3.28}$$

The stopping criteria of Newton's Method are the same as those of the Secant Method. Then we know the approximate maximum likelihood frequency estimate $\hat{\omega}_{Newton} = \hat{\omega}^{(m)}$. Finally, we can find $\hat{\theta}_{Newton}$ by

$$\hat{\theta}_{Newton} = \angle\{\exp(-j\hat{\omega}_{Newton}t_0)A(\hat{\omega}_{Newton})\}. \tag{3.29}$$

The algorithm of the Newton's method is described in Algorithm 2.

Algorithm 2 Newton's Method

```
Import ω̂^(0)
Δ = 1, n = 0
while |Δ| ≥ ε do
  n = n + 1
  Δ = F'(ω̂^(m−1)) / F''(ω̂^(m−1))
  ω̂^(m) = ω̂^(m−1) − Δ
  if F'(ω̂^(m−1)) == F'(ω̂^(m)) OR F'(ω̂^(m)) == 0 then
    BREAK;
  end if
end while
return ω̂^(m)
```

We use Figure 3.6 as an example to show how the FFT-Newton estimator works. As

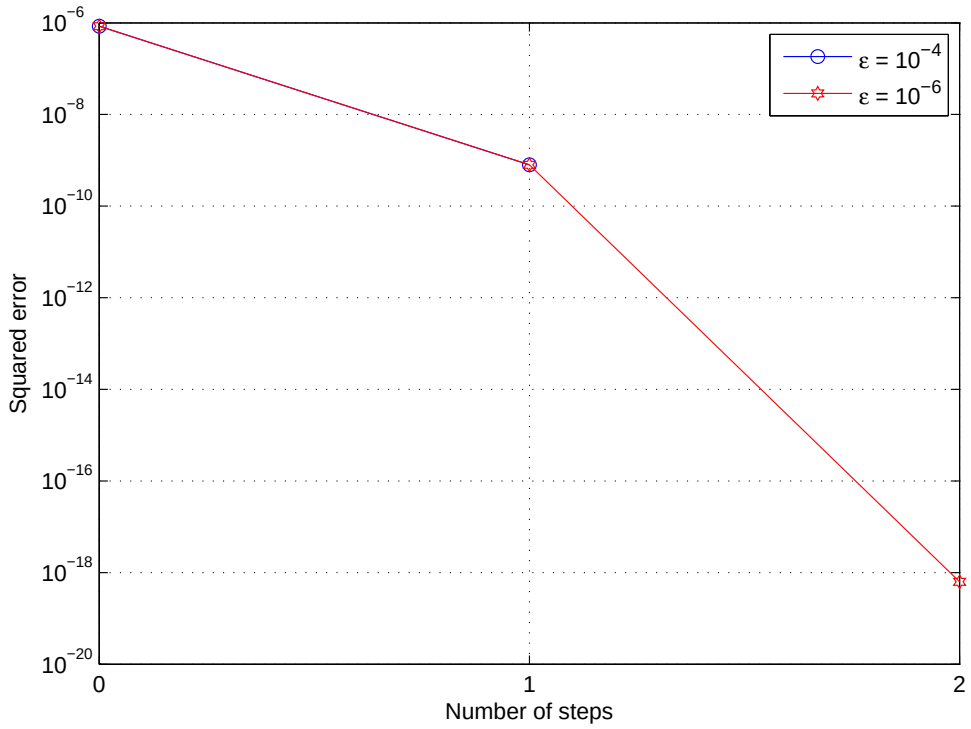

Figure 3.15: FFT-Newton MLE example.

shown in Figure 3.6, the peak is located at $k = 410$. Then we pick up one neighbourhood point and assign it to be $\hat{\omega}^{(0)}$. Thus, $\hat{\omega}^{(0)} = 2\pi \times 409/(MT)$. Now we start the iteration. As shown in Figure 3.15, the estimator only takes one step to converge within $\epsilon = 10^{-4}$. The iteration finds $\hat{\omega}_{Newton} = 0.6283$. Since the error tolerance is controllable, we can ensure that the squared error of frequency estimate is less than 10^{-8}. In fact, the squared error of frequency estimate is 7.9984×10^{-10}. If we increase ϵ to 10^{-6}, the iteration only takes one more step to converge.

Figure 3.16 and Figure 3.17 show the MSE of the frequency and phase estimators of the FFT-Newton MLE respectively as a function of SNR for three values of M, $M = 2^{10}$, $M = 2^{12}$, and $M = 2^{14}$, for a complex observation with AWGN. All the results assume an observation with $N = 513$, sampling period $T = 1$ second, and the first sample time $t_0 = -256$. The frequency and the phase are independent random variables for uniform distributions given by (3.6) and (3.7) respectively. 500 realizations of the complex exponential signal using (2.1) and AWGN are generated with fixed $b_0 = 1$. The error tolerance ϵ is set

to 10^{-4} for the iteration. The initial value is $\hat{\omega}^{(0)} = (k-1)2\pi/(MT)$, where k is the peak index estimated by the coarse search.

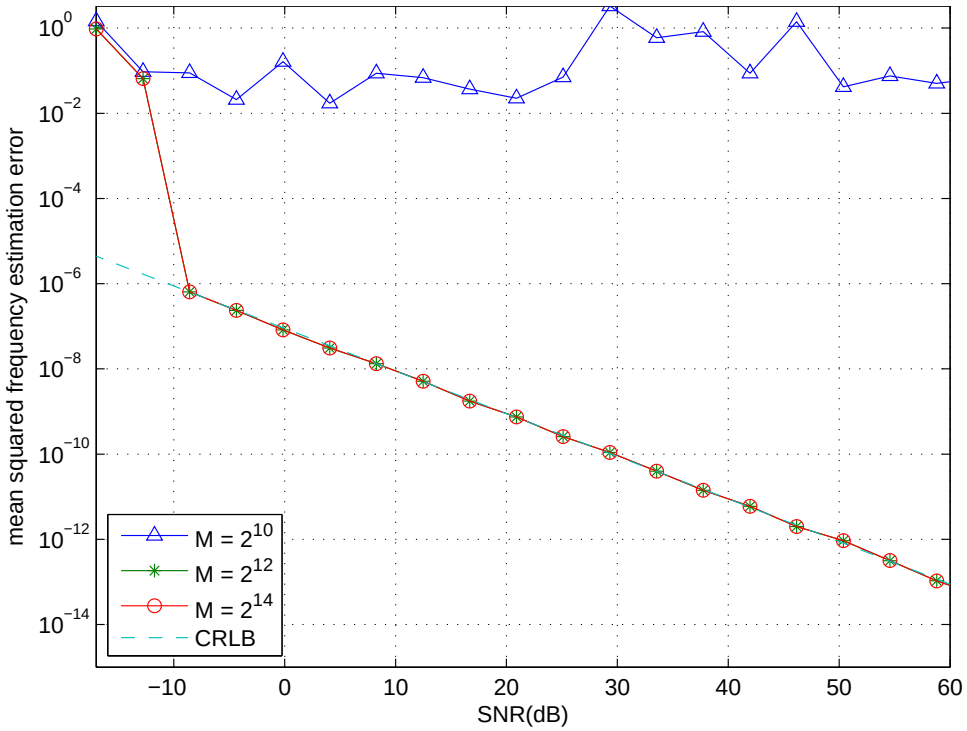

Figure 3.16: Mean squared frequency estimation error for complex signal of FFT-Newton MLE with $\epsilon = 10^{-4}$.

Compared with the FFT-Secant estimator with the same M, the FFT-Newton estimator with $M = 2^{12}$ can track the CRLB up to the SNR of 60dB while the iteration has $\epsilon = 10^{-4}$. It shows that the FFT-Newton estimator has a better iteration performance than the FFT-Secant estimator for small M. Figure 3.17 shows the mean squared phase estimation error. Similar to the FFT-Secant MLE, the phase estimator with $M = 2^{10}$ fails to track the CRLB over the entire range. It is caused by the bad estimation of the frequency.

3.4.1 Comparison of the FFT-Secant MLE and the FFT-Newton MLE

As shown in Figure 3.13 and Figure 3.16, both MLEs can attain the CRLB up a high SNR. Therefore, a discussion of their errors, computational complexity, and convergence is necessary.

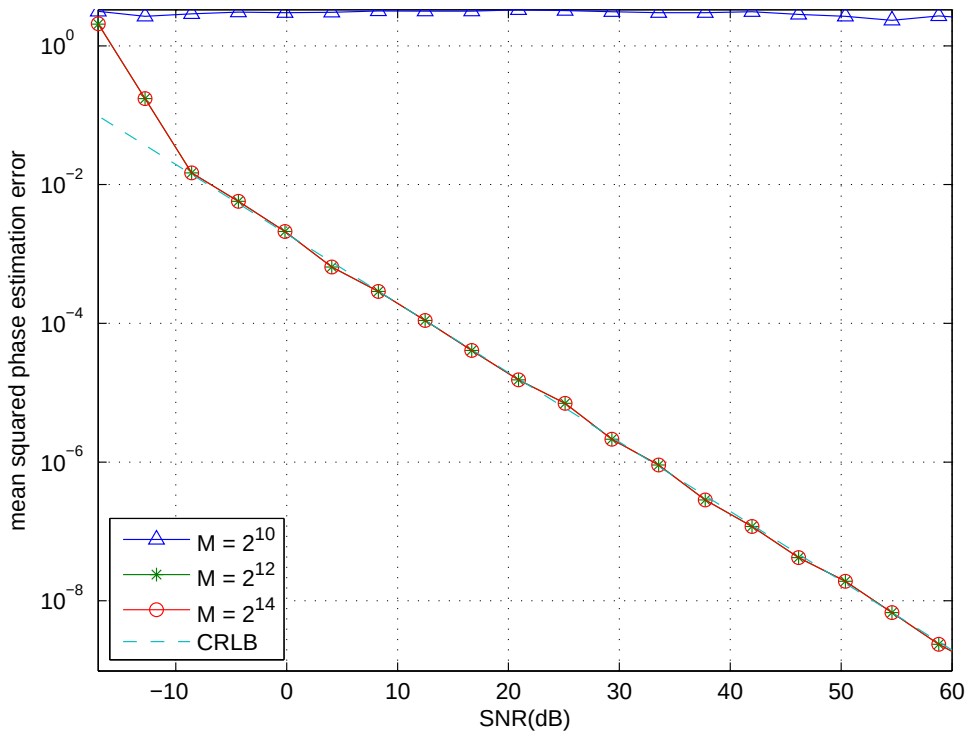

Figure 3.17: Mean squared phase estimation error for complex signal of FFT-Newton MLE with $\epsilon = 10^{-4}$.

The error formula of the Secant method is

$$\alpha - x_{n+1} = -(\alpha - x_{n-1})(\alpha - x_n)\frac{f''(\zeta_n)}{2f'(\xi_n)} \tag{3.30}$$

with ξ_n between x_{n-1} and x_n, and ζ_n between x_{n-1} , x_n, and α [21].

For Newton's method, the error formula is

$$\alpha - x_{n+1} = -(\alpha - x_n)^2\frac{f''(\zeta_n}{f'(x_n)}. \tag{3.31}$$

For both methods, the error formulas are similar. If we pick up the initial point(s) very close to the root α, the converge errors should be very close.

As shown in the iteration formulas (3.11) and (3.22), for each iterate, the Secant Method requires one function evaluation, $f(x)$, and Newton's method requires two function evaluations, $f(x)$ and $f'(x)$. Therefore, Newton' s method is generally more expensive per iteration.

Newton's method tends to converge faster, however, than the Secant method. Recall the definition of the order of convergence. A sequence of iterates $\{x_n | n \geq 0\}$ is said to converge with order $p \geq 1$ to a point α if

$$|\alpha - x_{n+1}| \leq c|\alpha - x_n|^p \quad n \geq 0 \tag{3.32}$$

for some $c \geq 0$. The constant c is called the rate of convergence of x_n at α [21]. According to this definition, Newton's method has an order of convergence $p = 2$. For the Secant method, the order of convergence is $p = (1 + \sqrt{5})/2 \simeq 1.62$. Therefore, Newton's method trends to converge more rapidly, and consequently it will require fewer iterations to attain a given desired accuracy, which has been shown in Figure 3.10 and Figure 3.15. The reason is that $\frac{x^{(m-1)} - x^{(m-2)}}{f(x^{(m-1)}) - f(x^{(m-2)})}$ is an approximation of $1/f'(x)$.

In fact, as discussed in [21], if the execution time to evaluate $f'(x)$ is more than 44 percent of that to evaluate $f(x)$, then the Secant method is more efficient. In our application, the function evaluation of $F'(x)$ requires $2N$ evaluations on trigonometric functions, $2(4N+1)+2(5N+2)+4 = 18N+10$ multiplications, and $8N+1$ additions. The functions evaluation

of $F''(x)$ requires $2N$ evaluations on trigonometric functions, $2(5N+2)+2(6N+2)+12 = 26N+20$ multiplications, and $8N+3$ additions. Therefore, for the computation of multiplications and additions, the ratios of evaluation are $\frac{26N+20}{18N+10}$ and $\frac{8N+3}{8N+1}$ respectively. Therefore, the Secant method tends to be more efficient in this application when N is large.

3.5 Approximate Maximum Likelihood Estimation using FFT and Bisection Method

Although we have introduced two iterative ML estimators which can track the CRLB over a wide range of SNR, both of them have a common drawback. Without the knowledge of the observation, the estimator may pick up the points which are not close enough to the peak. Therefore, in order to ensure the converge, we introduce another root-finding method called the bisection method. If f is a continuous function on the interval $[a,b]$ and $f(a)f(b) < 0$, then the bisection method converges to the root of f by halving the range. The best estimation is the midpoint of the smallest range found. Therefore, after n steps, the absolute error is

$$|x^{(n)} - x^{(n-1)}| = \frac{|b-a|}{2^n}.$$

In our application, we firstly use the coarse search to locate the peak index, k, of the Fourier Transform observation. Then we initialize $\omega^{(0)} = (k-1)2\pi/(MT)$ and $\omega^{(1)} = (k+1)2\pi/(MT)$. Since $\omega^{(0)}$ and $\omega^{(1)}$ locate on two opposite sides of $\hat{\omega}_{bisection}$, which maximizes $F(x)$ approximately, we can ensure convergence. The stop conditions for the FFT-bisection MLE is the same as those of the FFT-Secant MLE. The algorithm of the bisection method is described in Algorithm 3.

Figure 3.18 and Figure 3.19 show the MSE of the frequency and phase estimators of the FFT-bisection MLE respectively as a function of SNR for $M = 2^{10}$, $M = 2^{12}$, and $M = 2^{14}$, for a complex observation. All the results assume an observation with $N = 513$, sampling period $T = 1$ second, and the first sample time $t_0 = -256$. The frequency and the phase are independent random variables for the uniform distributions (3.6) and (3.7) respectively. 500 realization of the complex exponential signal using (2.1) and AWGN are generated with

Algorithm 3 The Bisection Method

Import $\omega^{(0)}, \omega^{(1)}$
if $F^{'}(\omega^{(0)}) \leq 0$ **then**
  $lo = \omega^{(0)}$
  $hi = \omega^{(1)}$
else
  $lo = \omega^{(1)}$
  $hi = \omega^{(0)}$
end if
$\omega^{(2)} = lo + \frac{(hi-lo)}{2}$
$m = 2$
while $(\omega^{(m-1)} \neq lo) AND (\omega^{(m-1)} \neq hi)$ **do**
  $m = m + 1$
  if $F^{'}(\omega^{(m-1)}) \leq 0$ **then**
    $lo = \omega^{(m-1)}$
  else
    $hi = \omega^{(m-1)}$
  end if
  $\Delta = \frac{hi-lo}{2}$
  $\omega^{(m)} = lo + \Delta$
  if $|\Delta| \leq \epsilon$ OR $F^{'}(\hat{\omega}^{(m-1)}) == F^{'}(\hat{\omega}^{(m)})$ OR $F^{'}(\hat{\omega}^{(m)}) == 0$ **then**
    BREAK
  end if
end while
return $\omega^{(m)}$

fixed $b_0 = 1$. The error tolerance ϵ is set to 10^{-4} for the iteration. The initial values are $\hat{\omega}^{(0)} = (k-1)2\pi/(MT)$ and $\hat{\omega}^{(1)} = (k+1)2\pi/(MT)$, where k is the peak index estimated by the coarse search.

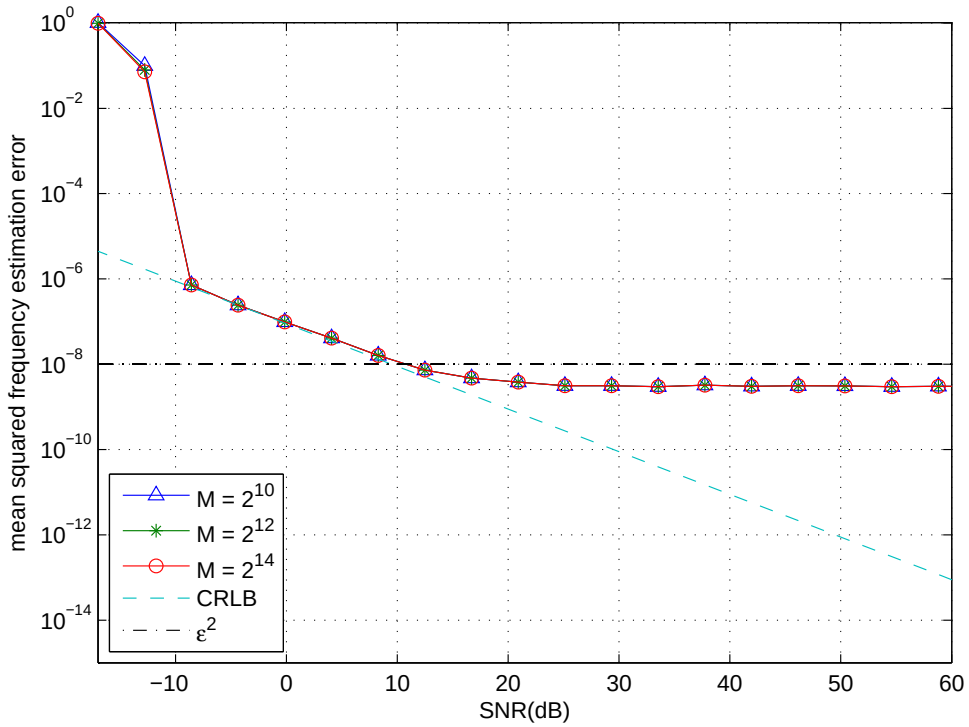

Figure 3.18: Mean squared frequency estimation error for complex signal of FFT-bisection MLE with $\epsilon = 10^{-4}$.

Compared with the FFT-Secant ML estimator and the FFT-Newton ML estimator, the most obvious advantage of the FFT-bisection ML estimator is that the estimator converges even with small values of M. As shown in Figure 3.18, the estimator with $M = 2^{10}$ can track the CRLB over a small range of SNR. However, compared with the FFT-Secant and the FFT-Newton estimators with the same value of M, the estimators with $M = 2^{12}$ and $M = 2^{14}$ have worse performances. They can only track the CRLB up to a SNR about 9dB, which is a smaller estimation range. In order to improve the performance, we increase ϵ from 10^{-4} to 10^{-6}. As shown in Figure 3.20, the frequency estimation range is extended to the SNR of 50dB. We try to increase ϵ further to 10^{-8}. As shown in Figure 3.21, the frequency estimation range extends to the SNR of 60dB.

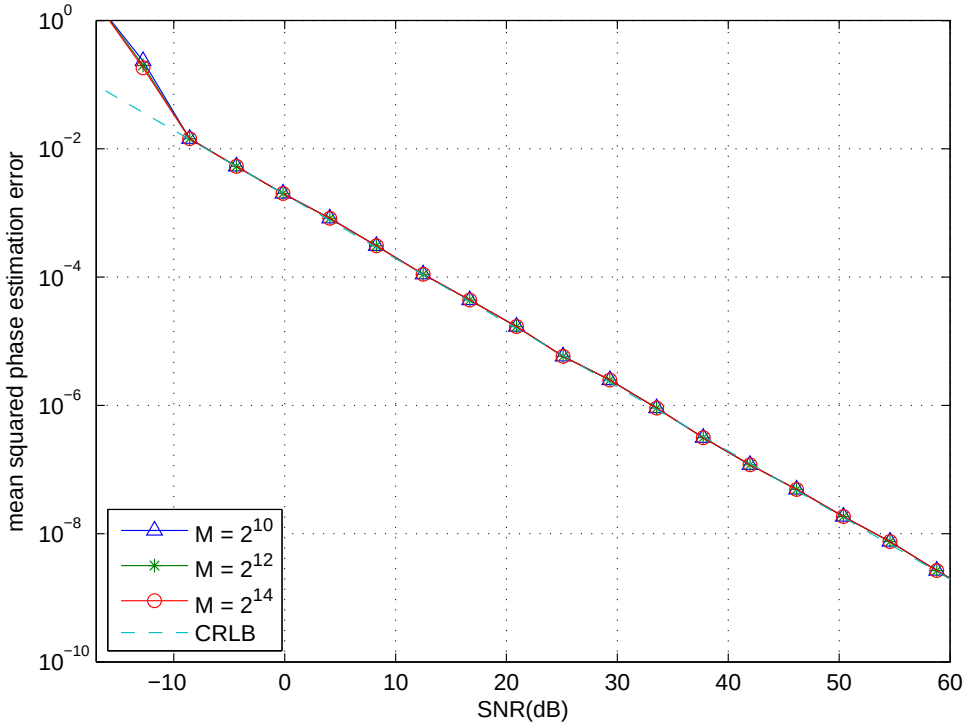

Figure 3.19: Mean squared phase estimation error for complex signal of FFT-bisection MLE with $\epsilon = 10^{-4}$.

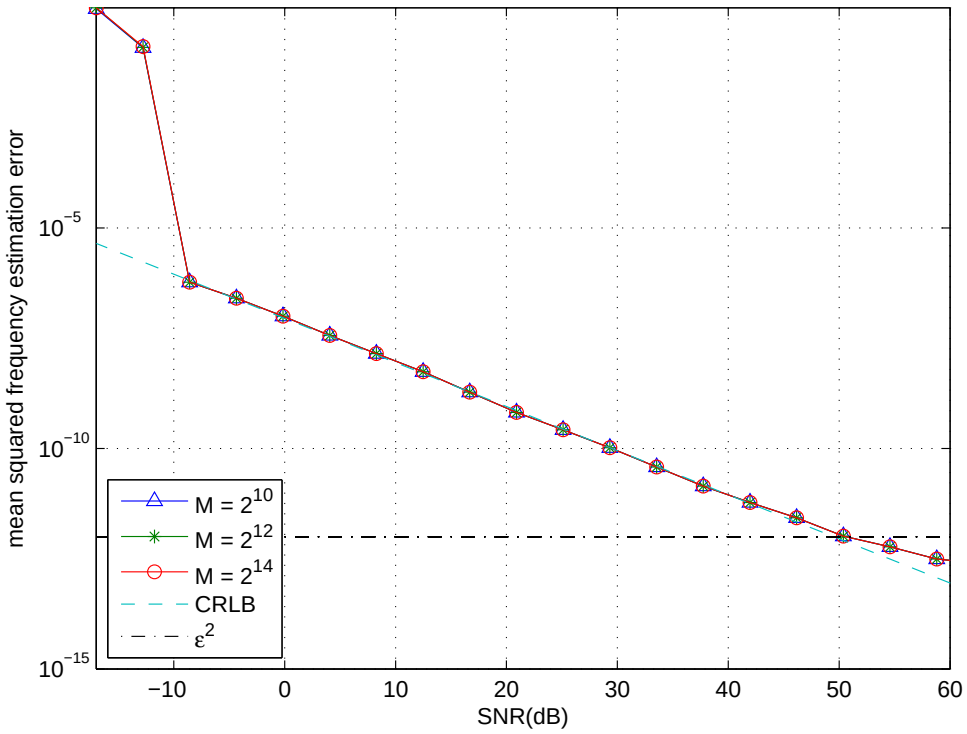

Figure 3.20: Mean squared frequency estimation error for complex signal of FFT-bisection MLE with $\epsilon = 10^{-6}$.

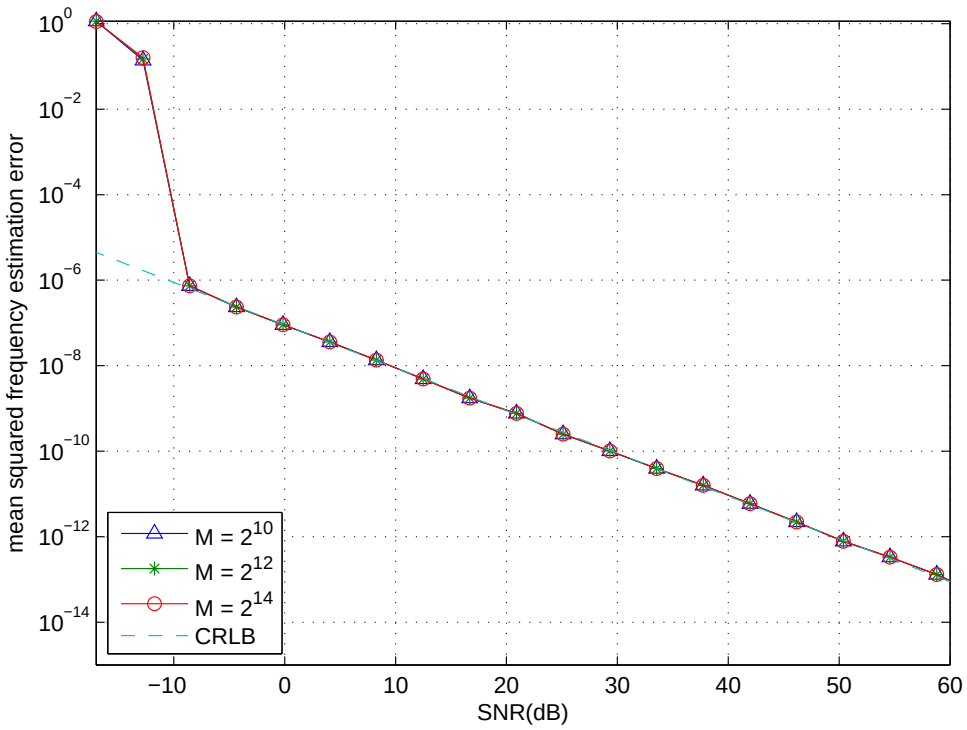

Figure 3.21: Mean squared frequency estimation error for complex signal of FFT-bisection MLE with $\epsilon = 10^{-8}$.

Although decreasing the ϵ can improve the estimation accuracy, at the same time, the execution time also increases. Compared with other two iterative estimators, the FFT-bisection estimator has the slowest convergence speed. By using the definition of the order of convergence (3.32) given in Section 3.4.1, we can find the order of convergence of the bisection method is 1. It means the bisection converges linearly. Compared with other two estimators, which converge quadratically, the FFT-bisection estimator does not have advantage on convergence speed. Therefore, the bisection method is not the first choice. When other two estimators do not converge, then we can apply the FFT-bisection MLE for estimation because the convergence is guaranteed.

3.6 Conclusion

In this chapter, we discussed five approximate maximum likelihood estimators and analysed the their performance in terms of the mean squared frequency and phase estimation errors as well as the computational complexity. Firstly, we introduced the Fast Fourier Transform Maximum Likelihood Estimator (FFT MLE), which cannot track the CRLB at high SNR unless the number of FFT points becomes very large. Then we presented four fine search techniques, each of them uses the FFT MLE as the coarse search and refines this coarse estimate to improve the performance. The first one is using the quadratic interpolation to locate the approximate maximum, which has better estimation performance but still cannot track the CRLB at the high SNR unless M is large. Then we provided a detailed study about the FFT-Secant MLE, which is presented by Rife in [8]. After that, we proposed other two iterative estimators: the FFT-Newton MLE and the FFT-bisection MLE. We discussed the MSE of these three iterative MLE as well as the computational complexity of them.

Besides the FFT-Secant MLE and the FFT-Newton MLE with bad initial values, all the proposed FFT-based MLEs have similar threshold, which is about a SNR of -9dB. All of them can attain the CRLB at low SNR. However, when the SNR is high, some MLEs fail to achieve the CRLB. For a small value of M, the FFT MLE cannot achieve the CRLB at high SNR values. Compared with the FFT MLE, the FFT-Quad MLE has a wider estimation

range. However, it only achieves the CRLB at 60dB when $M = 2^{14}$. For the iterative MLEs, the FFT-Secant MLE and the FFT-Newton MLE do not converge when $M = 2^{10}$. The reason is the initial points are not sufficiently close to the root of $F^{'}(\omega)$. When M is larger, both FFT-Secant and FFT-Newton MLEs can attain the CRLB up to 60dB SNR with reasonable error tolerance. The FFT-bisection MLE can always attain the CRLB at 60dB SNR because its algorithm guarantees the convergence.

For the estimation at low SNR, we recommend the user to choose the FFT MLE or the Quad MLE with two reasons. The first reason is both MLEs can attain the CRLB at low SNR. In addition, the computational complexity of the FFT MLE and the Quad MLE is less than the iterative MLE because the function evaluation of $F^{'}(\omega)$ is not efficient. For the estimation at high SNR, neither the FFT MLE nor the Quad MLE is a good fit because both of them fail to achieve the CRLB at high SNR unless we use a large value of M. Among the three iterative MLEs, we encourage the user to try the FFT-Secant MLE at first. The FFT-Secant needs less computation than the FFT-Newton MLE and has faster convergence speed than the FFT-bisection MLE. When the FFT-Secant MLE does not converge, an alternative estimator is the FFT-bisection MLE, which can guarantee the convergence.

We believe the these discussed methodologies will be useful in applications such as data analysis and performance evaluation. Therefore, for a system with low sampling rate, the proposed estimators can be applied to the real-time applications via hardware accelerating such as Field-Programmable Gate Array (FPGA) and Graphics Processing Unit (GPU). By using these devices, the most time consuming part, the computation of the FFT, can be reduced significantly.

Chapter 4

Zero Crossing Phase and Frequency Estimation

The common feature of the ML estimators presented in previous chapter is the estimator requires to know the entire signal before doing estimation. However, in a real-time application, this approach may lead to undesired latency in the computation of the estimates. In real-time applications, it is desirable to estimate the unknown parameters in a sequential way. This section presents a new low-complexity and low-latency algorithm for estimating the phase and frequency of a single tone in white noise. The proposed "zero-crossing" (ZC) phase and frequency estimator is based on zero-crossing detection of the real and imaginary parts of the complex observation. Like Tretter and Kay estimators in [13] and [14], the ZC estimator is based on the linear regression of the instantaneous phase of the observation of (2.1). A key difference is that the ZC estimator operates only on the time/phase coordinates of the real and imaginary zero crossings of (2.1). An advantage of this approach is that no arctangent operations are required since the phase of the signal at zero crossings is known (modulo 2π). Another advantage of this approach is that the number of zero crossing is (2.1) is typically less than N. A sequential least squares implementation of the zero-crossing estimator is also presented in which the phase and frequency estimates are updated as each new crossing is detected. This feature provides the availability for real-time applications.

4.1 Algorithm

4.1.1 Zero-Crossing Phase and Frequency Estimator

The zero-crossing estimator operates by detecting zero crossings in the real and imaginary components of (2.1) and storing the estimated time and phase of these zero crossing points to a coordinate set $\mathcal{S} = \{(t_1, \phi_1), \ldots, (t_L, \phi_L)\}$ where L is the number of detected zero crossings. The zero crossing occur at known phase (modulo 2π) according to Table 4.1. The ZC estimate is the obtained by performing linear regression on the coordinate set $\mathcal{S}$.

Table 4.1: Phase of a complex exponential at zero crossing

type of zero crossing	phase
imag part, positive	$k2\pi$
real part, negative	$k2\pi + \pi/2$
imag part, negative	$k2\pi + \pi$
real part, positive	$k2\pi + 3\pi/2$

In order to avoid the incorrect zero crossing detection, which can be caused by the noise, a state machine shown in Figure 4.1 is used to detect zero crossings. The variable $x[n]$ denotes either the real part or imaginary part of $z[n]$. A negative-slope zero crossing is detected on state transitions $1 \rightarrow 2$ and $3 \rightarrow 2$. A positive-slope zero crossing is detected on state transitions $2 \rightarrow 1$ and $4 \rightarrow 1$. The hysteresis parameter $\alpha \geq 0$ sets the threshold at which zero crossings are detected. Two of these state machines run simultaneously, each appending coordinates to $\mathcal{S}$ as zero crossings are detected in the real and imaginary parts of the observation.

In order to illustrate how the state machine works, consider the example shown in Figure 4.2 with $\alpha = 0.5$. We initialize the coordinate set $\mathcal{S} = \emptyset$. The first zero crossing is detected by the imaginary state machine as a positive-slope zero crossing of the imaginary part of the signal when state of the imaginary signal transitions from state 4 to state 1 at sample index $n = 8$. The time of this crossing is estimated by using a simple linear interpolation between the last sample in state 2, i.e. $n = 4$, and the first sample in state 1, i.e. $n = 8$.

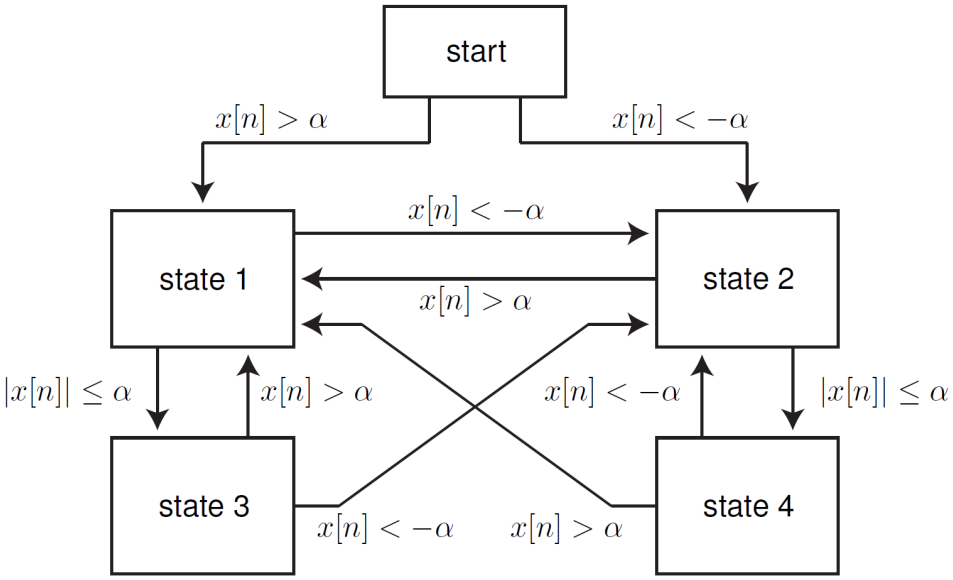

Figure 4.1: State machine implementation of the zero crossing detector.

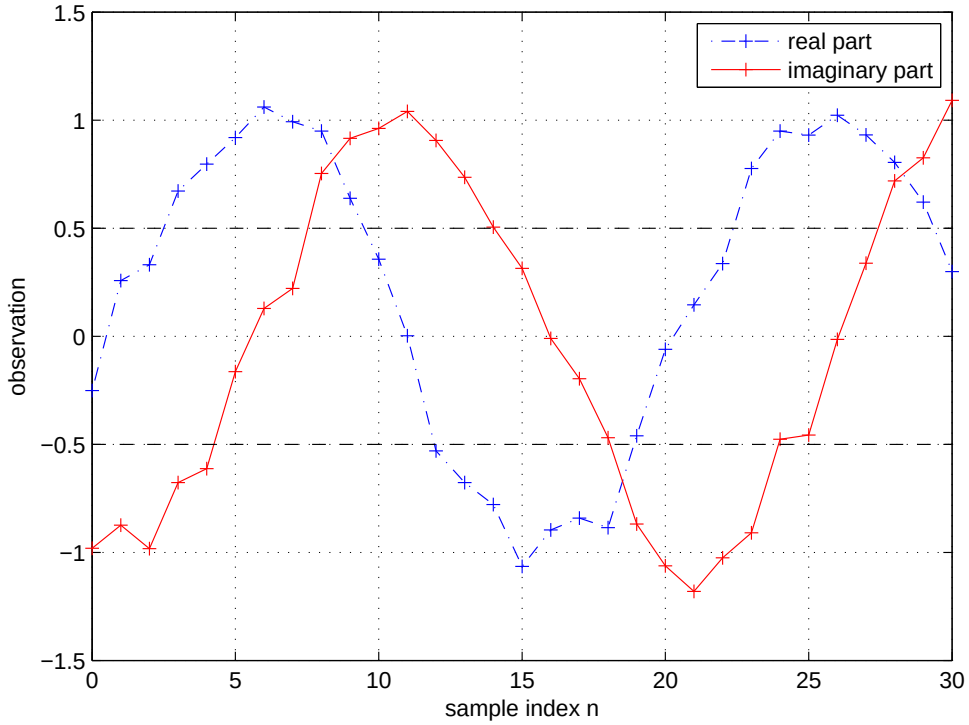

Figure 4.2: Noisy observation with $\alpha = 0.5$.

Therefore, the time t_1 is computed by (4.1)

$$t_1 = t_{n_1} - \frac{\Im\{z[n_1]\}}{\frac{\Im\{z[n_2]\}-\Im\{z[n_1]\}}{t_{n_2}-t_{n_1}}} \tag{4.1}$$

where $t_{n_1} = 4T$, $t_{n_2} = 8T$ and $\Im\{z[n]\}$ denotes the value of the imaginary part of the complex exponential signal $z[n]$ at sample index n. The phase of this positive-slope zero crossing of the imaginary signal is set to zero. Hence, the first coordinate $(t_1, 0)$ is added to the coordinate set $\mathcal{S}$.

The next zero crossing in Figure 4.2 is detected by the real state machine as a negative-slope zero crossing of the real part of the signal when state of the real signal transitions from state 3 to state 2 at sample index $n = 12$. The time of this zero crossing is estimated by the simple linear interpolation between the last sample in state 1, i.e. $n = 9$, and the first sample in state 2, i.e. $n = 12$. Therefore, the time t_2 is computed by (4.2)

$$t_2 = t_{n_2} - \frac{\Re\{z[n_2]\}}{\frac{\Re\{z[n_1]\}-\Re\{z[n_2]\}}{t_{n_1}-t_{n_2}}} \tag{4.2}$$

where $t_{n_1} = 9T$, $t_{n_2} = 12T$ and $\Re\{z[n]\}$ denotes the value of the real part of the complex exponential signal $z[n]$ at sample index n. The phase of this negative-slope zero crossing of the real signal is set to $\pi/2$ and the second coordinate $(t_2, \pi/2)$ is added to the coordinate set $\mathcal{S}$.

This processing is repeated as additional zero crossings are detected by the real and imaginary state machines. In the example shown in Figure 4.2, a total of five zero crossings are detected. Therefore, $\mathcal{S} = \{(t_1, 0), (t_2, \pi/2), (t_3, \pi), (t_4, 3\pi/2), (t_5, 2\pi)\}$. At any point in time once $L \geq 2$, the zero crossing phase and frequency estimates can be generated by performing a linear regression [24] on $\mathcal{S}$ to determine the least-squares fit for the slope and intercept. the stop and intercept are set to $\hat{\omega}_{ZC}$ and $\hat{\theta}_{ZC}$, respectively.

4.1.2 Sequential Implementation

Like other linear-regression-based estimators, such as Tretter's method, the ZC estimator also can be implemented efficiently on a sample-by-sample basis. Prior to the first sample

of the observation, we initialize the tracking variables $A = B = C = D = E = 0$. When a new coordinate (t_i, θ_i) is added to the coordinate set $\mathcal{S}$, the tracking variables are also updated as follows:

$$\begin{aligned} A &= A + t_i^2, \\ B &= B + 1, \\ C &= C + t_i \\ D &= D - t_i\theta_i, \text{and} \\ E &= E - \theta_i \end{aligned} \tag{4.3}$$

As shown in [24], the intercept and slope can be calculated directly from the tracking variables as

$$\hat{\theta}_{ZC} = \text{intercept} = -\frac{EA - CD}{BA - C^2}, \text{and} \tag{4.4}$$

$$\hat{\omega}_{ZC} = \text{slope} = -\frac{C}{A}\hat{\theta}_{ZC} - \frac{D}{A}. \tag{4.5}$$

4.2 Computational Complexity

Since there are typically four real and imaginary zero crossings per period, the number of real and imaginary zero crossings in (2.1) is approximated as $L \simeq \frac{4Nf_0}{fs}$, where $f_0 = \frac{\omega_0}{2\pi}$. At each detection of zero crossing, the estimator calculates the time of the zero crossing (via interpolation) and also updates the tracking variables. The simple linear interpolation, as shown in (4.1) and (4.2), requires one real-valued multiplication and three real-valued additions. The update of the tracking variables requires two real-valued multiplications and five real-valued additions. Computing the intercept and slope in (4.4) and (4.5) requires eight real-valued multiplications and five real-valued additions. Therefore, the total number of multiplications required to implement the zero-crossing estimator can be approximated as

$$\text{number of real multiplication} \simeq 3L + 8 = \frac{12Nf_0}{fs} + 8, \tag{4.6}$$

and

$$\text{number of real addition} \simeq 8L + 5 = \frac{32Nf_0}{fs} + 5. \tag{4.7}$$

Note that these operations (except for the last eight multiplications and the last five additions) can be performed as the sample arrives. Also, there are no arctangent operations, which are typically performed with a lookup Table 4.2.

Table 4.2: Computational Complexity of Tretter, Kay, and zero-crossing estimators

Algorithm	Number of real multiplication	Number of arctangent
Zero-crossing	$\frac{12Nf_0}{fs} + \mathcal{O}(1)$	0
Kay with exact atan2	$5N + \mathcal{O}(1)$	$N - 1$
Tretter with exact atan2	$N + \mathcal{O}(1)$	N
Kay with approx atan2	$7N + \mathcal{O}(1)$	0
Tretter with approx atan2	$3N + \mathcal{O}(1)$	0

For the approximations for the arctangent function were discussed in [25], each approximate arctangent can be computed with one to three real-valued multiplications. In the table above, we assume two real multiplications per arctangent operation. For the complex multiplication, we use the fact that one complex multiplication is equal to four real-valued multiplications.

4.3 Numerical Results and Discussion

This section presents numerical results comparing the proposed zero-crossing estimator to Tretter's and Kay's estimators. For Tretter's and Kay's estimators, we use both the exact arctangent and the approximate four quadrant arctangent [25].

Figure 4.3 and Figure 4.4 show the MSE of the frequency and phase estimators of the zero-crossing estimator, Kay's estimator, and Tretter's estimator respectively as a function of SNR for a complex observation. All the results assume an observation with $N = 513$, sampling period $T = 1$ second, and the first sample time $t_0 = -256$. The frequency is an independent random variable for a uniform distribution

$$\omega_0 \sim \mathcal{U}(0.09 \times 2\pi, 0.11 \times 2\pi). \tag{4.8}$$

The phase is an independent random variable for a uniform distribution

$$\theta_0 \sim \mathcal{U}(-\pi, \pi). \tag{4.9}$$

500 realizations of the complex exponential signal using (2.1) and AWGN are generated with fixed $b_0 = 1$. The hysteresis parameter of the ZC estimator is $\alpha = 0.4$. Since Tretter's method using the exact arctangent is as same as Kay's method using the exact arctangent [19], we use the same plot for these two methods.

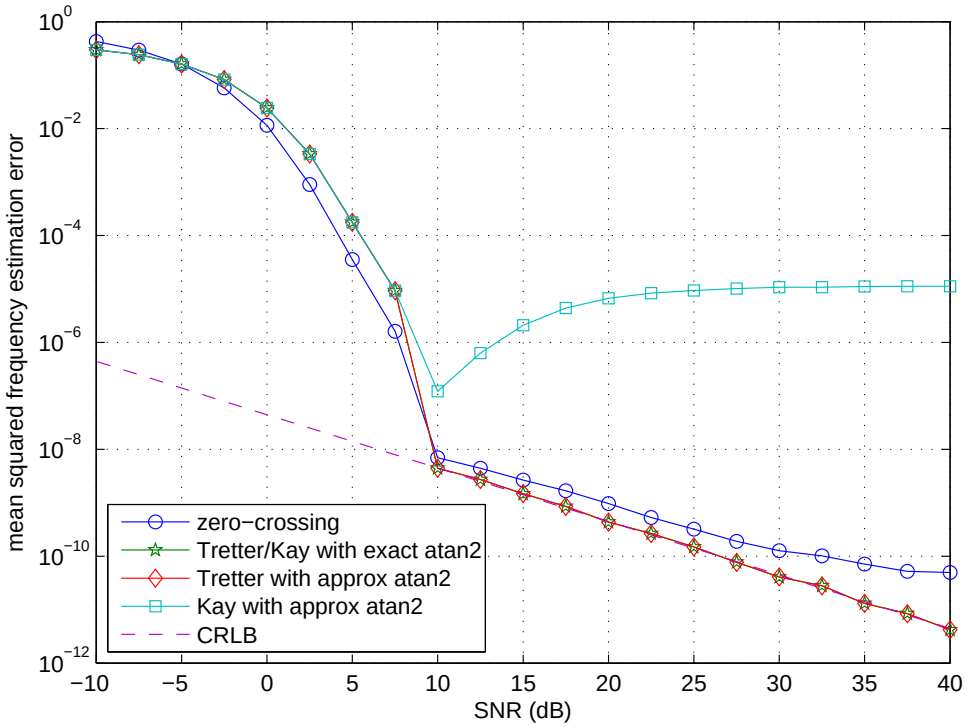

Figure 4.3: Mean squared frequency estimation error of ZC estimator with $\alpha = 0.4$.

Figure 4.3 shows the mean squared frequency estimation error as a function of SNR. Besides Kay's estimator with the approximate arctangent, all the estimators have the same threshold, which is about a SNR of 10dB. The ZC estimator provides frequency within about 2.5dB of the CRLB for $10 \leq$ SNR $\leq$ 30dB. Above 30dB SNR, the ZC estimator's performance continues to improve, but deviates from the CRLB due to the error of linear interpolation as discussed in [26]. Tretter's estimator, independent of which method is used

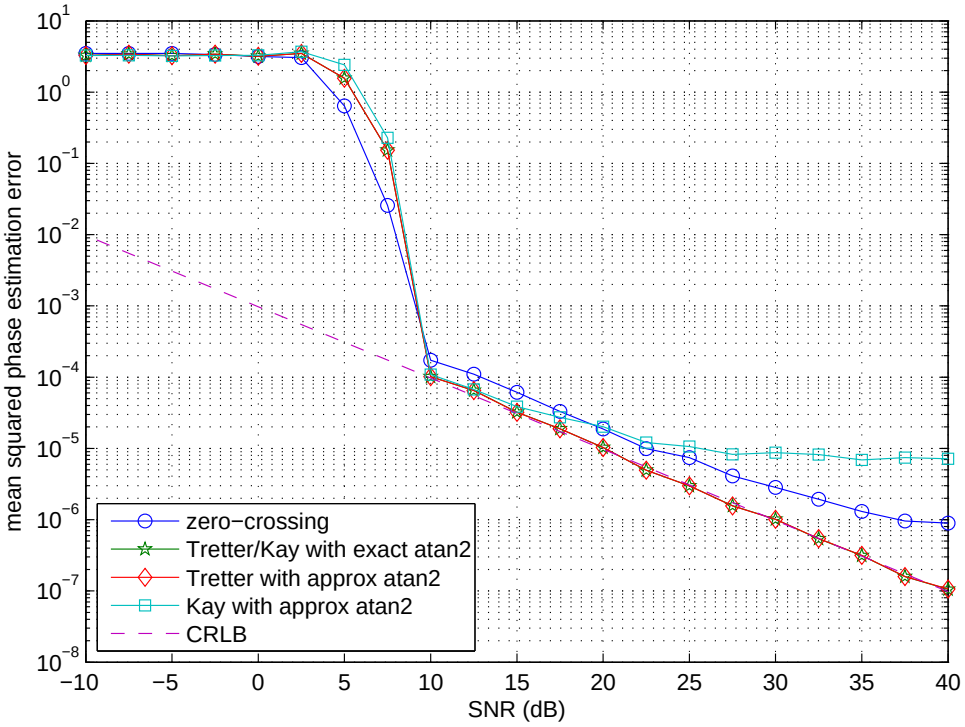

Figure 4.4: Mean squared phase estimation error of ZC estimator with $\alpha = 0.4$.

to calculate arctangent function, can always track the CRLB up to a SNR of 40dB. Kay's estimator with the approximate arctangent function fails to track the CRLB. The reason is that the Kay estimator performs arctangent operations on the phase increments between two consecutive samples of the observation. At high SNR the sequence of phase increments tends to be very similar. Hence, any error in the arctangent approximation is repeated and the frequency estimate becomes biased.

Figure 4.4 shows the mean squared phase estimation error as a function of SNR. The threshold of all the estimators is about a SNR of 10dB. The ZC estimator tracks the CRLB within about 2.5 dB up to a SNR of 30dB. After 30 dB, the ZC estimator's performance still improves but deviates from the CRLB due to the error of linear interpolation. Since the phase estimator of Kay's method depends on the estimated frequency, the phase plot of the Kay's method with approximate arctangent starts to deviate from the CRLB at 20dB. Other estimators can track the CRLB up to a SNR of 40dB.

Figure 4.5 shows the frequency estimate MSE as a function of SNR with $\alpha = 0.2$.

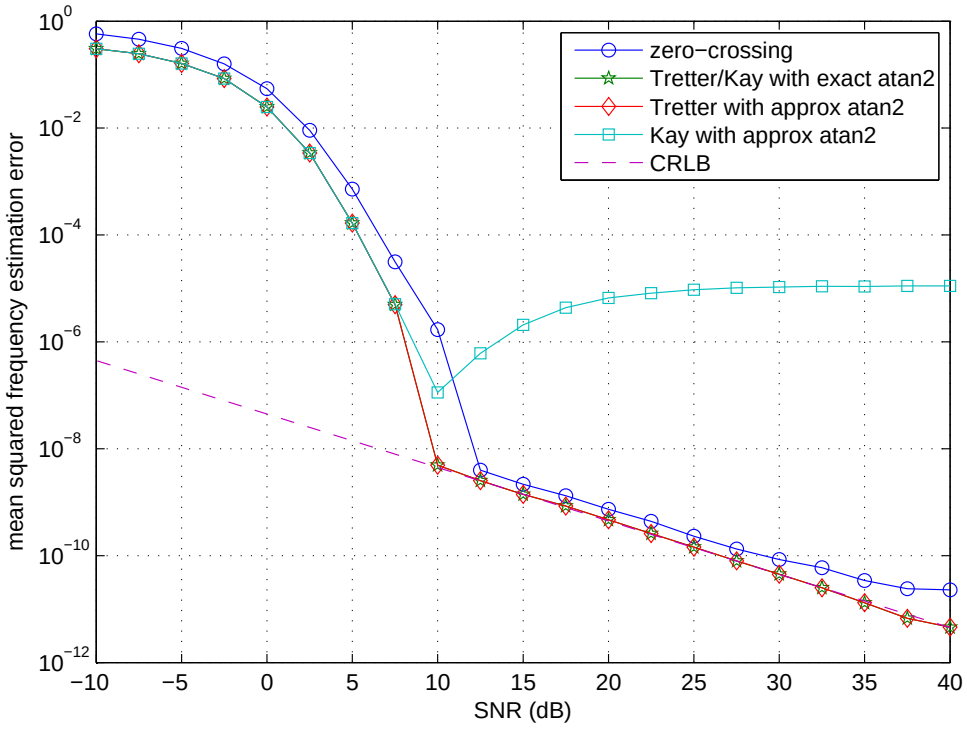

Figure 4.5: Mean squared frequency estimation error of ZC estimator with $\alpha = 0.2$.

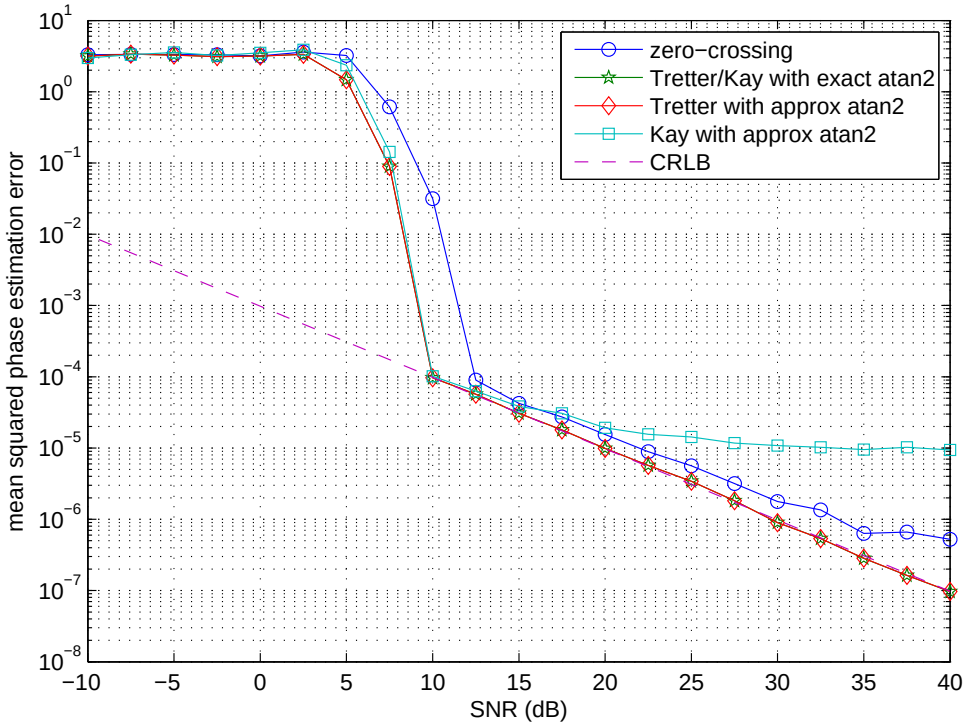

Figure 4.6: Mean squared phase estimation error of ZC estimator with $\alpha = 0.2$.

Compared with Figure 4.3, the threshold is increased to 12.5dB. However, the gap between the ZC estimator and the CRLB is smaller. For example, at 40dB, the ZC estimator with $\alpha = 0.4$ has a MSE about 7×10^{-11}. At the same time, the ZC estimator with $\alpha = 0.2$ has a MSE about 2×10^{-11}.

Figure 4.6 shows the phase estimate MSE as a function of SNR with $\alpha = 0.2$. Compared with Figure 4.4, the threshold is increases to 12.5dB. But the ZC estimator with $\alpha = 0.2$ is closer to the CRLB. For example, at 40dB, the ZC estimator with $\alpha = 0.4$ has a MSE about 8×10^{-7} in Figure 4.4. In Figure 4.6, the error is reduced to 5×10^{-7}.

In summary, when the hysteresis parameter α is reduced from 0.4 to 0.2, the threshold of the estimation is increased. However, at the same time, the ZC estimator is closer to the CRLB. Therefore, it is a trade-off to decide the value of the hysteresis parameter α.

4.4 Conclusion

This chapter proposes a new algorithm for the phase and frequency estimation of a single tone complex observation in white noise. The proposed zero-crossing phase and frequency estimator is based on the detection of zero crossing of the real and imaginary parts of the complex observation and sequential least squares processing. Compared with other linear-regression-based estimators, such as Tretter's method and Kay's method, the zero-crossing estimator does not require any arctangent operations. In addition, the proposed estimator provides low latency with the fact that phase and frequency estimates can be updated sequentially as new zero crossings are detected.

Chapter 5

Zero Crossing Estimator Refinements

The algorithm introduced in Section 4.1 is the fundamental algorithm of the zero-crossing estimator. With some additional improvements and modifications, the performance of the zero-crossing estimator can be improved significantly. In this section, three improvement methods are introduced. The first refined estimator uses a new approach for interpolation with better accuracy. The second approach uses a bandpass filter as the pre-processing. The third approach uses the ZC estimator as the coarse search and the root-finding algorithm as the fine search. For each new approach, we compute the mean squared frequency and phase estimation errors as a function of SNR. Also, we compare the MSEs of the new approaches with the fundamental approach. The numerical results show that all of the new approaches have better performance.

5.1 Zero Crossing Estimator using Local Linear Regression for Zero-crossings

Since the performance of the ZC estimator depends on the accuracy of the zero crossing time estimates, the choice of the interpolation technique used to estimate the zero crossing times is very important [26]. In Section 4.1.1, we discuss about using the simple linear

interpolation to estimate the time of crossing, i.e. (4.1) and (4.2). However, in this interpolation method, only two points are used. Hence, the accuracy of the interpolation is low. A better interpolation method is proposed in this section.

In this new interpolation method, we use a **local linear regression**, which uses all of the samples from the last sample above/below to the first sample below/above the hysteresis threshold. Depending on the normalized frequency of the observation and the hysteresis parameter, α, local linear regression can produce significantly better zero crossing time estimates by exploiting the information contained in the samples between the hysteresis thresholds.

Let's recalled the example in Section 4.1.1, in order to estimate the time of the first zero crossing by using this new method, we need to collect the sample indices and the associated magnitudes and put into a coordinate set $\mathcal{L}$. Therefore, when state machine of the imaginary signal transitions from state 4 to state 1 at sample index $n = 8$, the coordinate set $\mathcal{L}$ is

$$\mathcal{L} = \{(t_{n_1}, \Im\{z[n_1]\}), (t_{n_2}, \Im\{z[n_2]\}), (t_{n_3}, \Im\{z[n_3]\}), (t_{n_4}, \Im\{z[n_4]\}), (t_{n_5}, \Im\{z[n_5]\})\} \quad (5.1)$$

where n_1, n_2, n_3, n_4, n_5 are equal to $4, 5, 6, 7, 8$ respectively, $t_{n_i} = n_i \times T$, and $\Im\{z[n_i]\}$ is the imaginary part of the complex exponential signal $z[n]$ at sample index n_i. Now we can apply the linear regression to the coordinate set $\mathcal{L}$. Since this regression can be achieved on a sample-by-sample basis, we can use the similar idea presented Section 4.1.2. Five new tracking variables, $\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}$, and $\tilde{E}$, are set and updated as shown in (4.3). After the zero crossing is detected, the intercept, b, and the slope, g, are estimated by using (5.2) and (5.3).

$$b = \text{intercept} = -\frac{\tilde{E}\tilde{A} - \tilde{C}\tilde{D}}{\tilde{B}\tilde{A} - \tilde{C}^2}, \text{and} \quad (5.2)$$

$$g = \text{slope} = -\frac{\tilde{C}}{\tilde{A}}b - \frac{\tilde{D}}{\tilde{A}}. \quad (5.3)$$

Then, the time of crossing is (5.4)

$$t = -\frac{b}{g} \quad (5.4)$$

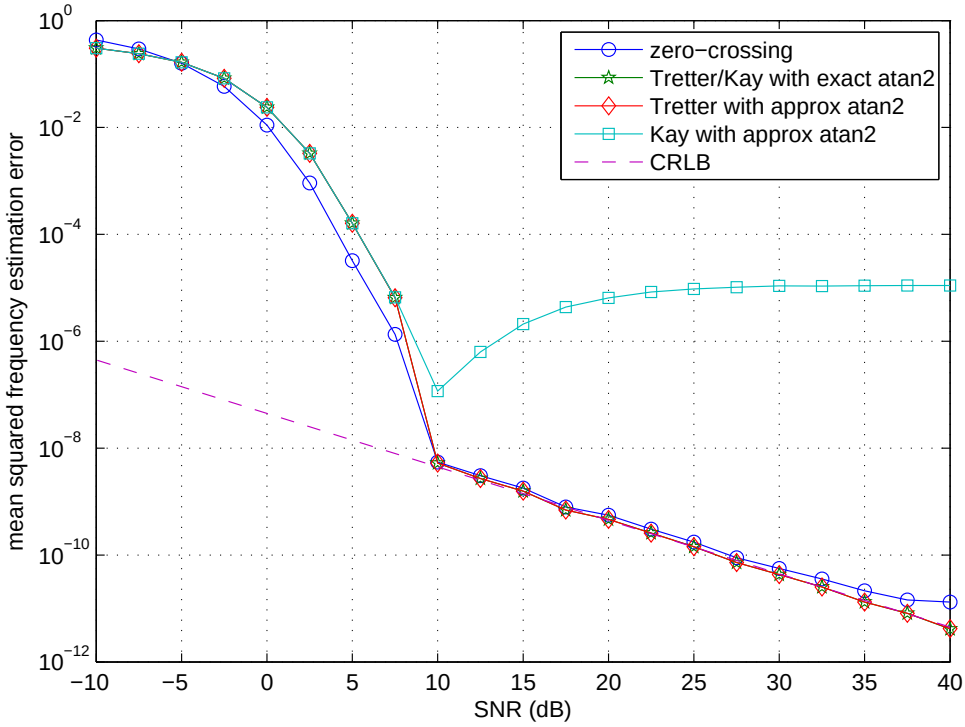

Figure 5.1: Mean squared frequency estimation error of ZC estimator via local linear interpolation with $\alpha = 0.4$.

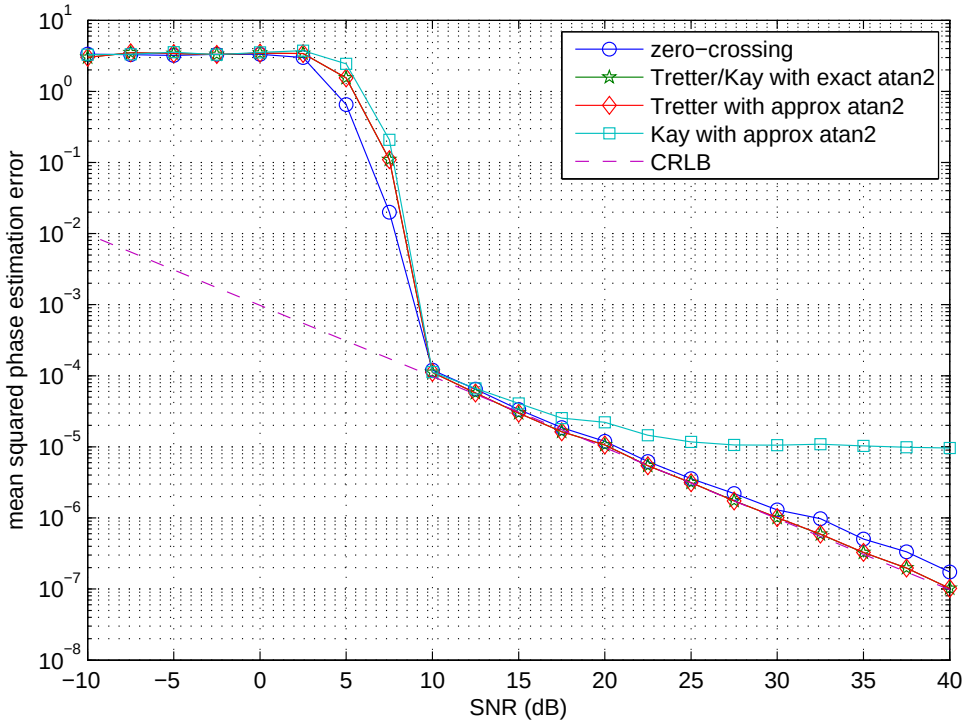

Figure 5.2: Mean squared phase estimation error of ZC estimator via local linear interpolation with $\alpha = 0.4$.

5.1.1 Numerical Results and Discussion

Figure 5.1 and Figure 5.2 show the mean squared frequency and phase estimation errors as a function of SNR for a complex observation. The simulation parameters are the same as what we used in Section 4.3. For the interpolation, we use the local interpolation on all the samples between the last sample above/below and the first sample below/above the hysteresis threshold. Then the zero crossing time can be estimated. In this simulation, the hysteresis threshold is $\alpha = 0.4$.

Compared with Figure 4.3 and Figure 4.4, the performance of the ZC estimator using the new interpolation method is improved significantly. These results show that the zero-crossing estimator provides frequency and phase estimates within about one dB of the CRLB for $10 \leq \text{SNR} \leq 30\text{dB}$. Above 30dB SNR, the zero-crossing estimator's performance continues to improve, but deviates from the CRLB due to the error of interpolation. However, compared with Figure 4.3 and Figure 4.4, the improvement is significant. For example, in Figure 4.3, the mean squared frequency estimation error at 40dB is about 7×10^{-11}. For the ZC estimator using the local interpolation technique, the error is reduced to a MSE about 1×10^{-11}.

5.2 Zero Crossing Estimator using Pre-Filtered Observation

In many communication and signal processing applications, the central frequency is known by both transmitter and receiver. For example, under IEEE 802.11g standard, we know the central frequency of each channel. Therefore, a bandpass filter can be pre-computed and used as the pre-processing of estimation. Since the noise is filtered, the detection of the zero crossing is more accurate. Consequently, the accuracy of the ZC estimator is improved.

5.2.1 Numerical Results and Discussion

Figure 5.3 and Figure 5.4 show the mean squared frequency and phase estimate errors as a function of SNR for a complex observation. All the results assume an observation with $N = 513$, sampling frequency $fs = 16000$ Hz, and the first sample time $t_0 = -256$. The

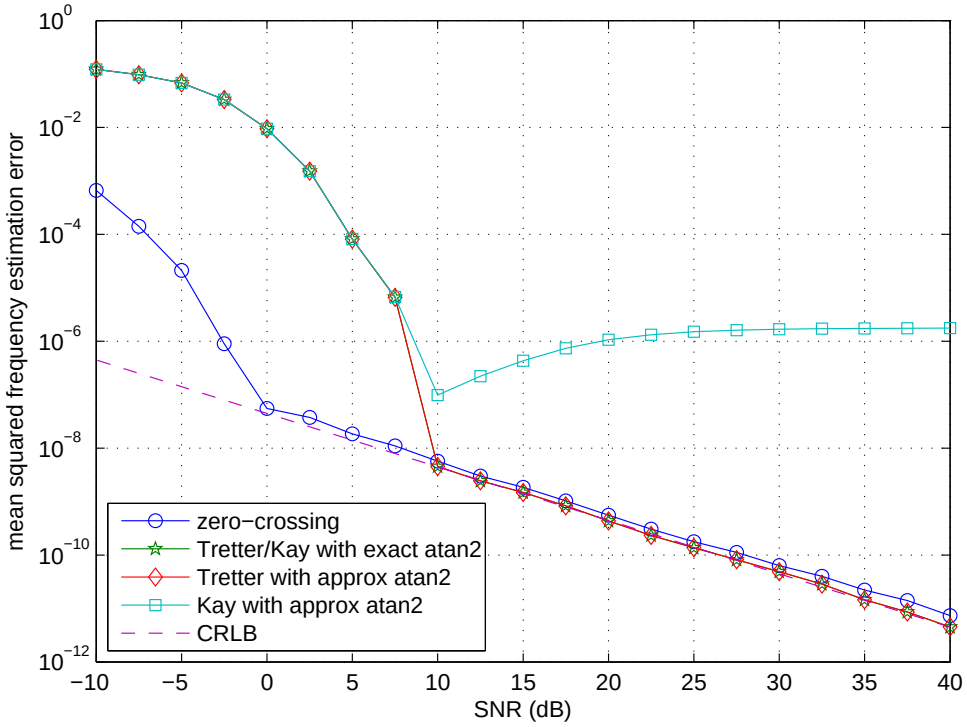

Figure 5.3: Mean squared frequency estimation error of ZC estimator with $\alpha = 0.4$ and a 64th order FIR filter.

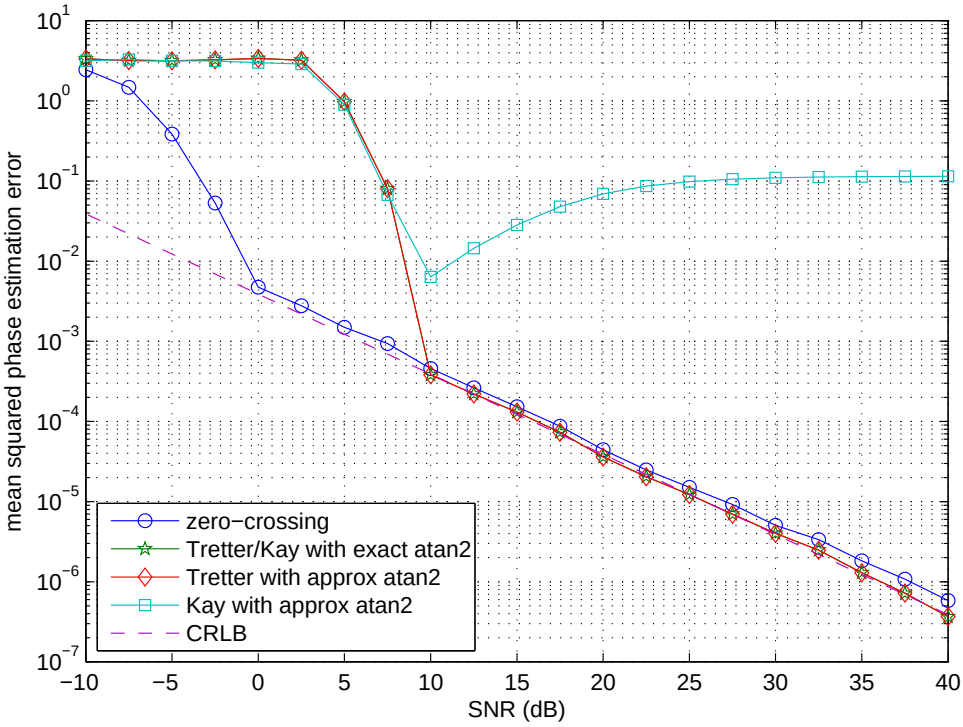

Figure 5.4: Mean squared phase estimation error of ZC estimator with $\alpha = 0.4$ and a 64th order FIR filter.

frequency is an independent random variable for a uniform distribution

$$\omega_0 \sim \mathcal{U}(1019 \times 2\pi, 1021 \times 2\pi). \tag{5.5}$$

The phase is an independent random variable for a uniform distribution

$$\theta_0 \sim \mathcal{U}(-\pi, \pi). \tag{5.6}$$

500 realizations of the complex exponential signal using (2.1) and AWGN are generated with fixed $b_0 = 1$. The hysteresis parameter of the ZC estimator is $\alpha = 0.4$. Simple linear interpolation is used in this simulation. In addition, a 64th order Finite Impulse Response (FIR) filter is used to pre-process the observation. Therefore, the observation will be filtered by the FIR filter and the output of the filter will be used for the estimation. In this simulation, the passband of the bandpass filter is $[920, 1120]$ Hz.

Compared with Figure 4.3 and Figure 4.4, the performance of the ZC estimator is improved significantly. In Figure 5.3 and Figure 5.4, the threshold is decreased from a SNR of 10dB to a SNR of 0dB, which is better than that of Tretter and Kay estimators. In addition, the frequency and phase estimates within about one dB of the CRLB up to a SNR of 40dB, which is much better than the fundamental ZC estimator without filter.

5.3 Zero Crossing Estimator using Non-Linear Iterative Method

In Chapter 3, we presented five approximate maximum likelihood estimators. The FFT-Secant MLE, the FFT-Newton MLE, and the FFT-bisection MLE use the Fast Fourier Transform (FFT) as the coarse search and different root-finding algorithms as the fine search. Compared with FFT MLE, the performance of these three MLEs are improved significantly due to iterative methods. However, the computation of the FFT is slow and the FFT estimators require to know the entire observation before estimation. Therefore, the latency may be too high for real-time application. Nevertheless, these estimators provide a new idea which uses the ZC estimator as the coarse search and the root-finding algorithm as the fine search.

In this new approach, we still implement the zero-crossing estimator as we proposed in Section 4.1. After computing (4.4) and (4.5), the estimated frequency $\hat{\omega}_{ZC}$ are used as the coarse search frequency estimate for the fine search. Then we can use the Secant Method, as the fine search, to find the root of $F^{'}(\omega)$ in (3.18) because it has fast convergence speed and needs less computation. In the example shown in Section 3.3, only two steps are required to converge with error tolerance $\epsilon = 10^{-4}$. Now the initial values of the Secant Method are $\hat{\omega}^{(0)} = \hat{\omega}_{ZC}$ and $\hat{\omega}^{(1)} = \hat{\omega}_{ZC} - \Delta\omega$, where $\Delta\omega$ is a very small real number. As shown in Figure 4.3, the error of ZC estimator is very small. Thus, we can ensure that the initial points we pick are sufficiently close to the root. Consequently, we can ensure the convergence of the iteration. One thing we should notice is that in this approach, we do not need to compute the FFT. Therefore, the computational complexity of this new approach is less than the FFT-based MLEs. The algorithm of the Secant Method is shown Algorithm 1. The converged value will be $\hat{\omega}_{ZC-Secant}$ and finally we can compute $\hat{\theta}_{ZC-Secant}$ by using (5.7)

$$\hat{\theta}_{ZC-Secant} = \angle\{\exp(-j\hat{\omega}_{ZC-Secant}t_0)A(\hat{\omega}_{ZC-Secant})\}. \tag{5.7}$$

5.3.1 Numerical Results and Discussion

Figure 5.5 and Figure 5.6 show the mean squared frequency and phase estimation errors as a function of SNR for a complex observation. All the results assume an observation with $N = 513$, sampling period $T = 1$ second, and the first sample time $t_0 = -256$. The frequency is an independent random variable for a uniform distribution

$$\omega_0 \sim \mathcal{U}(0.09 \times 2\pi, 0.11 \times 2\pi). \tag{5.8}$$

The phase is an independent random variable for a uniform distribution

$$\theta_0 \sim \mathcal{U}(-\pi, \pi). \tag{5.9}$$

500 realizations of the complex exponential signal using (2.1) and AWGN are generated with fixed $b_0 = 1$. The hysteresis parameter of the ZC estimator is $\alpha = 0.4$. The Secant

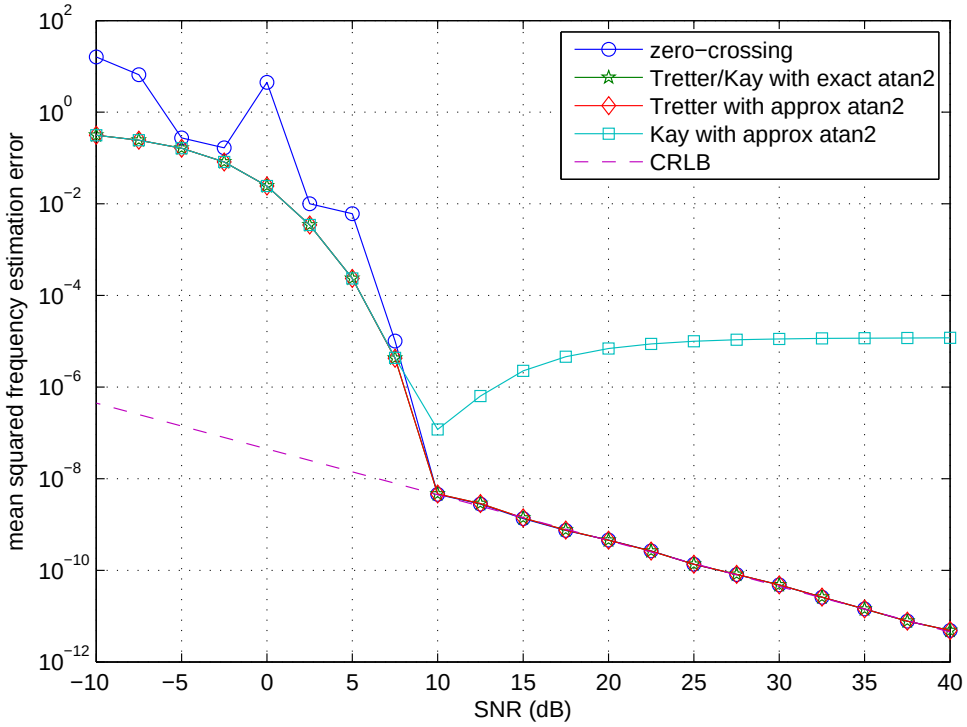

Figure 5.5: Mean squared frequency estimation error of ZC estimator via the Secant Method with $\alpha = 0.4$.

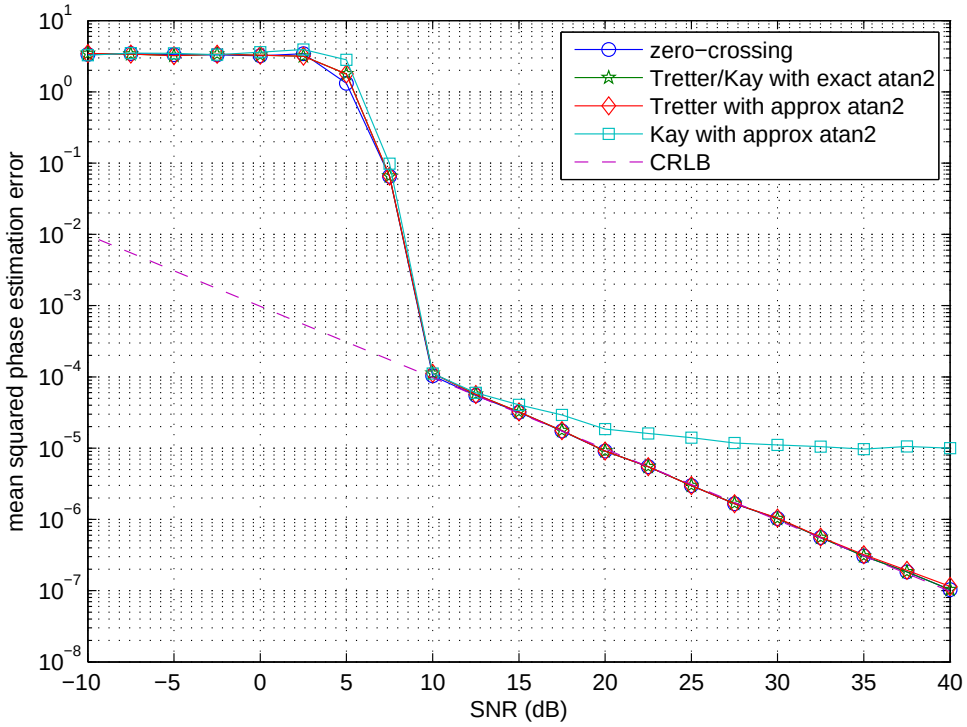

Figure 5.6: Mean squared phase estimation error of ZC estimator via the Secant Method with $\alpha = 0.4$.

Method is used as the post-processing of estimation. The error tolerance is $\epsilon = 10^{-4}$. The initial values of the Secant Method are $\hat{\omega}^{(0)} = \hat{\omega}_{ZC}$ and $\hat{\omega}^{(1)} = \hat{\omega}_{ZC} - 5 \times 10^{-4}$.

Compared with Figure 4.3 and Figure 4.4, the zero-crossing estimator with non-iterative method has much better performance. Although the threshold of estimation remains the same, the new approach has smaller mean squared error. As shown in Figure 5.5 and Figure 5.6, the new approach tracks the CRLB up to a SNR of 40dB.

5.4 Conclusion

In this chapter, we propose three improved approaches for the zero-crossing estimator. Compared with the mean squared frequency and phase estimation errors given in Figure 4.3 and Figure 4.4, all of the new approaches have better performance. Compared with the fundamental zero crossing estimator in Section 4.1, the first approach uses a local interpolation to improve the performance. Rather than using the two points to estimate the zero crossing time, the new interpolation algorithm uses all the points between the last sample above/below and the first sample below/above the hysteresis threshold to locate the time of zero crossing via a local linear interpolation. The numerical results show that the gap between the MSE and the CRLB is smaller than the fundamental estimator. The second approach uses a filter to improve the SNR. In the example we demonstrated, a 64th order FIR bandpass filter is used to reduce the power of noise. The numerical results show that compared with the fundamental approach and the ZC estimator with accurate interpolation, the ZC estimator with filter has a smaller threshold and MSEs. The third approach is using a root-finding algorithm as the fine search to improve the performance of the fundamental estimator. Although the threshold of this new method remains the same as the fundamental method, this approach has the least MSE among all the zero-crossing estimation refinements. The numerical results show that the estimator tracks the CRLB up a high SNR of 40dB.

Among all of the improvements, many common applications will benefit from the second one, which is uses the filter to pre-process the samples as they arrive. As what we will be discussed in the following chapter, this approach can be implemented in a sequential way

by using the circular buffer [27]. Although the ZC estimator with accurate interpolation can be applied in a real-time application as well, the performance of this estimator is worse than that of the one with filter. For the ZC estimator with non-linear iterative method, although the number of iterations is low, the function evaluation of $F^{'}(\omega)$ is not efficient. Also, this approach may fail without the prior knowledge about the SNR of communication channel.

Chapter 6

A Software-Defined-Radio Implementation of Time-Slotted Round-Trip Carrier Synchronization for Distributed Beamforming via Zero-Crossing Estimation

Transmit beamforming is an energy-efficient communication technique in which a transmitter sends a beacon over two or more antennas and aligns the phases of the transmissions across the antennas such as, after propagation, the signals combine constructively at the intended direction [5]. The recent idea about achieving transmit beamforming has been extended from a single antenna array to a network of cooperating single-antenna sources which behave in a distributed fashion, which is described as the distributed beamforming. Unlike conventional transmit beamforming, each single-antenna transmitter has its local oscillator, which is independent from other source nodes and also imperfect. Therefore, in

order to achieve beamforming at the destination in the desired direction, the carrier phase and frequency synchronization among the transmitter nodes is necessary [17].

In [28], the round-trip carrier synchronization algorithm was proposed for the carrier synchronization of the distributed beamforming. This algorithm uses the equivalence of round-trip propagation delays through a multihop chain of source nodes to achieve phase and frequency synchronization [17, 28]. [17] and [20] show an experiment of implementing the round-trip carrier synchronization by using software-defined radios (SDRs). The experiment includes two-source synchronization and three-source synchronization in both wired channel and acoustic channel. The hybrid Phase Locked Loop (PLL) is used for estimating unknown parameters. A two-source round-trip model is shown in Figure 6.1.

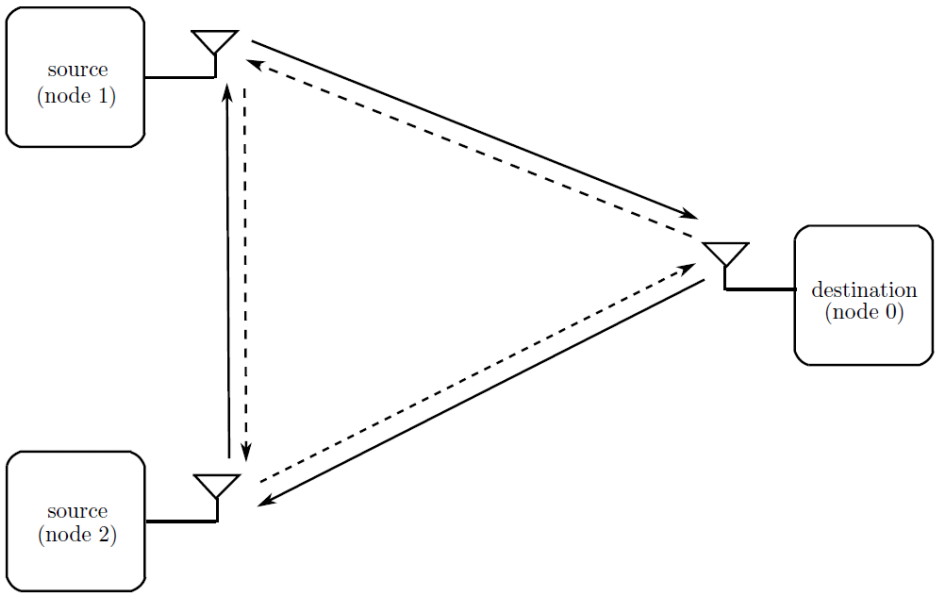

Figure 6.1: Time-slotted round-trip carrier synchronization system with two source nodes.

As shown in Figure 6.1, the fundamental concept of this algorithm is that an unmodulated carrier transmitted by the destination node and "bounced" around the clockwise circuit will incur the same total phase shift as an unmodulated carrier transmitted by the destination node "bounced" around the counter-clockwise circuit. In practice, the "bouncing" of carrier can be performed actively by having each source estimated the unknown parameters of the received beacon transmitted by other nodes. Then each source transmit-

ted a periodic extension of the signal received in a previous timeslot. Coherent combing is achieved since the destination is receiving the sum of two carriers, modulated by the common message, after they have propagated through circuits with identical phase shifts.

In the algorithm above, each node estimates the frequency and phase of the received signal in a specific timeslot. In order to reduce the total time for synchronization, a fast frequency and phase estimator is required for each timeslot. Therefore, a sequential estimator satisfies such requirement. Hence, we implement the proposed zero-crossing phase and frequency estimator in the time-slotted round-trip carrier synchronization and analyse the performance of this estimator in a wired channel.

In our implementation, we will use the methodology for the two-source case in wired channel discussed in [17]. The following sections will introduce the protocol of two source nodes synchronization system. Then the specific timeslot schedule and the map of equipment connection are given. In addition, the C functions of each key component of the round-trip synchronization system are discussed. Finally, the test results will be analysed and compared with the results from [17].

6.1 Two-Source Synchronization

As illustrated in [17], in a two-source system, there are four timeslots for the round-trip synchronization protocol. Among them, the first three slots are used for synchronizing the local oscillator of each node and the last slot is used for beamforming.

$\mathbf{TS}^{(0)}$: The destination node transmits the sinusoidal primary beacon to both source nodes. Both source nodes estimate phase and frequency from their local observation, denoted as $\hat{\theta}_{0i}$ and $\hat{\omega}_{0i}$, respectively, at source node $\mathrm{S_i}$ for $i \in \{1, 2\}$.

$\mathbf{TS}^{(1)}$: Source node S_1 transmits a sinusoidal secondary beacon to source node S_2. This secondary beacon is transmitted as a periodic extension of the beacon received in $\mathbf{TS}^{(0)}$. S_2 estimates local phase and frequency from this received observation which are denoted by $\hat{\theta}_{12}$ and $\hat{\omega}_{12}$, respectively.

$\mathbf{TS}^{(2)}$: Source node S_2 transmits a sinusoidal secondary beacon to source node S_1. This sec-

ondary beacon is transmitted as a periodic extension of the beacon received in $\mathbf{TS}^{(0)}$, with initial phase extrapolated from the phase and frequency estimates obtained by S_2 in $\mathbf{TS}^{(0)}$. S_1 generates local phase and frequency estimates from this observation denoted as $\hat{\theta}_{21}$ and $\hat{\omega}_{21}$, respectively.

$\mathbf{TS}^{(3)}$: Both sources transmit simultaneously to the destination as a distributed beamformer. The carrier frequency of each source is based on both local frequency estimates obtained in the prior timeslots. The initial phase of the carrier at each source is extrapolated from the local phase and frequency estimates from the secondary beacon observation.

6.2 Experimental Methodology for Two-Source Test in Wire-Connected Channel

Each source node in the wired two-source time-slotted round-trip carrier synchronization system is implemented by using one Taxes Instrument TMS320C6713DSK board at a sampling frequency 16kHz. Besides the DSK boards, the experiment kit also includes a CD player (SONY CD Walkman D-CJ01), a Behringer EURORACK UB1201 Mixer, a Tektronix TDS 3014 oscilloscope, a Marntz PMD 661 solid state recorder, various RCA, BNC connectors, and cables for connecting between the electronic computers' inputs and outputs. The whole system is shown in Figure 6.2.

According to the discussion in Section 6.1, after the estimation, each source node needs to keep its estimates for transmission. Therefore, we put the estimates in holdover. In addition, each node needs to estimate the unknown parameters twice in the two-source system. Therefore, we set two estimators and two holdovers on each node and they can operate simultaneously.

The detector for the first beacon, the round-trip protocol, and the ZC estimator are implemented by programming the TMS320C6713DSK in C language using Texas Instrument's Code Composer Studio integrated development environment. Each source node runs the system independently and there is not connection between each two nodes. The "des-

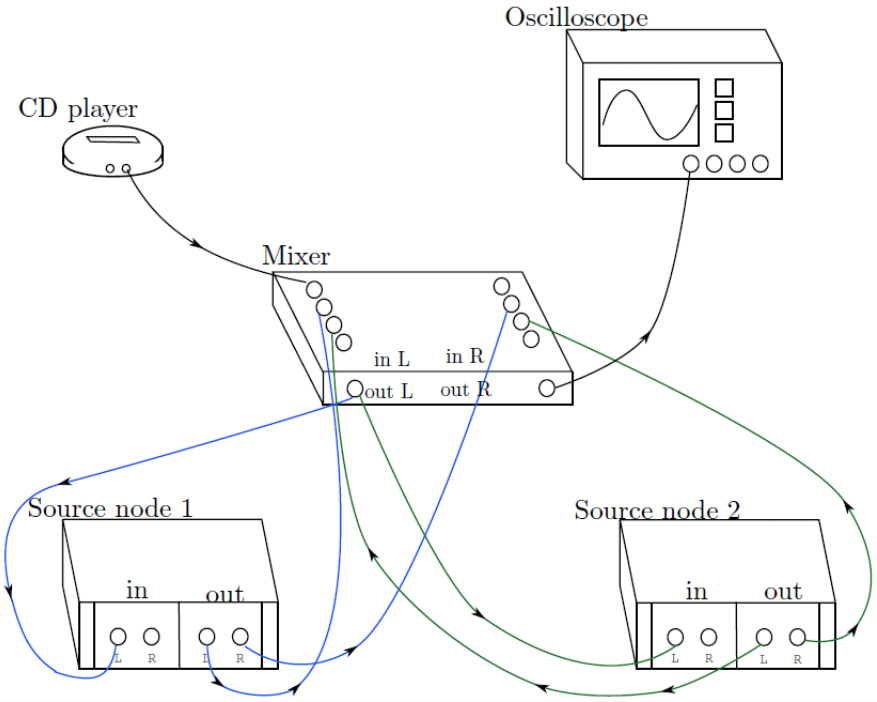

Figure 6.2: Implementation block diagram of the two-source time-slotted round-trip carrier synchronization system where the blue and green lines each represents a different signal wired path

tination node" was created by using the CD player for playing primary beacon, as well as the recorder for recording the beacons. An oscilloscope was connected to the output of the recorder for real-time monitoring. Each test is a complete execution of the 4 timeslots of the round-trip protocol. At the beginning, each source node enters into a state where it waits for the primary beacon. Once the primary beacon is detected, the nodes start to execute the round-trip protocol according to the schedule shown in Table 6.1.

Timeslot	Time	S_1	S_2
TS0	0.00s	detect primary beacon	detect primary beacon
	0.00-0.10s	wait	wait
	0.10-0.60s	update tracking variables of ZC1	update tracking variables of ZC1
	0.6s	estimate frequency and phase for ZC1	estimate frequency and phase for ZC1
	0.60-1.25s	holdover ZC1	holdover ZC1
TS1	1.25-1.35s	transmit holdover ZC1	holdover ZC1
	1.35-1.85s	transmit holdover ZC1	holdover ZC1 and update tracking variable of ZC2
	1.85s	transmit holdover ZC1	estimate frequency and phase for ZC2 and holdover ZC1
	1.85-2.25s	transmit holdover ZC1	holdover ZC1 and ZC2
	2.25-2.50s	wait	holdover ZC1 and ZC2
TS2	2.50-2.60s	wait	transmit holdover ZC1; also holdover ZC2
	2.60-3.10s	update tracking variables of ZC2	transmit holdover ZC1; also holdover ZC2
	3.10	estimate frequency and phase for ZC2	transmit holdover ZC1; also holdover ZC2
	3.10-3.50s	holdover ZC2	transmit holdover ZC1; also holdover ZC2
	3.50-3.80s	holdover ZC2	holdover ZC2
TS3	3.80-5.80s	transmit holdover ZC2	transmit holdover ZC2
	5.80-6.80s	clear state	clear state
	6.80s	re-arm primary beacon detector	re-arm primary beacon detector

Table 6.1: Two-source round-trip synchronization protocol timing. After detection of the primary beacon, each node keeps time using its sample clock running at 16 kHz.

6.3 Zero-Crossing Estimator Implemented on Source Node

The round-trip protocol and the ZC estimator are implemented on the TMS320C6713DSK board via C language. Based on the architecture and the design of the TMS320C6713DSK board, when a new sample arrives, the interrupt service routine (ISR) will be executed [29]. Therefore, the number of execution is equal to the sampling frequency in one second. In order to implement the sequential estimation, all the C codes are included in the ISR. Because of two estimators and two holdovers, all the estimation variables, such as the tracking variables and the estimates, are stored in array and indicated by the index of the estimator. The following parts will give detailed discussion of the implementations of the round-trip time table, the FIR filter, and the zero-crossing estimator. The competed code is presented in Appendix.

6.3.1 Implementation of the Round-Trip Time Table

According to Table 6.1, the two-source round-trip synchronization protocol is divided into four timeslots. There is a global variable called TS, which is used to identify the timeslot. Since the ISR is executed when a new sample is arrived, in each time slot, we use a unsigned integer *counter* to track the time. The variable *counter* is increased by one when the ISR is executed. The time of each event is pre-loaded and we compare *counter* with the pre-loaded schedule in each ISR call. Once they are equal, we will change to next event. Here is an example of how to implement it in Time Slot 0 (TS0).

```
1
2 //define time for each event
3
4 const unsigned int start_slot = 0; //start of time slot
5 const unsigned int start_listen = 1600; //start to listen @ 0.1s
6 const unsigned int stop_listen = 9600; //stop to listen @ 0.6s
7 const unsigned int end_slot = 20000; //end of time slot @ 1.25s
8 const unsigned int start_play = 0; //start to play
9 const unsigned int stop_play = 16000; //stop to play @ 1s
```

```
10
11 //end
12
13 //schedule protocol
14 if ((counter <start_listen)&&(counter>=start_slot))  //0.1s wait
15 {
16         // wait
17         counter++;
18 }
19 //0.5s track
20 else if((counter >= start_listen)&&(counter < stop_listen))
21 {
22         // update tracking variable for ZC1
23         counter++;
24
25 }
26 else if (counter == stop_listen)
27 {
28         //estimate frequency and phase for ZC1
29
30         counter++;
31         phase_updater(0);
32 }
33 else if ((counter > stop_listen) && (counter < end_slot))
34 {
35         //holdover ZC1
36         counter ++;
37 }
38 else if (counter == end_slot)
```

```
39 {
40         //holdover ZC1
41         TS++; //change to next time slot
42         counter = 0;
43 }
```

6.3.2 Implementation of Zero-Crossing Estimator

The implementation of zero crossing estimator includes four part: the state machine, the interpolation, the update of tracking variables, and the estimation based on tracking variables. The first three components are implemented sequentially. They are executed when a new sample arrives. The last component, the estimation, is only called at 0.6 second.

Implementation of State Machine

As discussed in Section 4.1, each received sample will be sent to the state machine and compared with the hysteresis parameter α. Since the transmitted beacon is sinusoid signal, therefore, only one state machine is needed. We use the if-else statement to implement the state machine. Once the zero crossing is detected, the time and the phase will be generated at first based on Table 4.1. Then the tracking variables will be updated according to (4.3). Here is an example of showing the State 3 in C code.

```
1 if (state[ZCnum] == 3)
2 {
3         if (input > A)
4         {
5                 // transition to state 1
6         }
7         else if ((input > A_i) && (input < A))
8         {
9                 // stay in state 3
```

```
10          }
11          else if (input < A_i)
12          {
13                  // detect zero-crossing
14
15                  // compute time and phase via
16                  // simple linear interpolation
17
18                  // update the tracking variables
19                  // transition to state 2
20
21          }
22  }
```

Implementation of Simple Linear Interpolation

In order to estimate the time of zero crossing, we implement the simple linear interpolation on the last sample above the upper hysteresis parameter and the first sample below the lower hysteresis parameter, which has been presented in Section 4.1. In the software implementation, we use two variables to track these two samples. Once a zero crossing is detected, we apply either (4.1) or (4.2) to estimate the time. The phase of the zero crossing is decided according to Table 4.1. Here is an example of showing linear interpolation in State 3.

```
1
2  //Interpolation
3  slope = (i2_value[ZCnum] - i1_value[ZCnum])*fs;
4  temp = i2[ZCnum] - i1[ZCnum];
5  slope = slope / (double)(temp);
6  zt[ZCnum] = (double)i1[ZCnum]*inv_fs;
7  zt[ZCnum] = zt[ZCnum] - i1_value[ZCnum]/slope;
```

```
8  //END of interpolation
9
10 n[ZCnum]++; //update number of zero crossings
11
12 //Decide phase
13 if (n[ZCnum]>=2)
14 {
15         ph[ZCnum] = ph[ZCnum] + pi;
16 }
17 else ph[ZCnum] = pi;
18
19
20 //shift to state 2
21 state [ZCnum]= 2;
```

Implementation of Updating Tracking Variables

The tracking variables are initialized to zero before starting the estimation process. In the process, the tracking variables are updated when a zero crossing is detected. Then the tracking variables will be updated according to (4.3). Here is a C function for updating tracking variables.

```
1 void ZC_updatepar(unsigned short ZCnum)
2 {
3         AA[ZCnum] = AA [ZCnum]+ zt[ZCnum]*zt[ZCnum];
4         BB[ZCnum]++;
5         CC[ZCnum] = CC[ZCnum] + zt[ZCnum];
6         DD[ZCnum] = DD[ZCnum] - zt[ZCnum]*ph[ZCnum];
7         EE[ZCnum] = EE[ZCnum] - ph[ZCnum];
8 }
```

Implementation of Frequency and Phase Estimation

After receiving 0.5 second beacon, the frequency and phase estimates are computed based on the tracking variables. (4.4) and (4.5) are used to estimate the frequency and phase. The C code for estimating unknown parameters is listed as:

```
1  void ZC_estimation(unsigned short ZCnum)
2  {
3  delta[ZCnum] = BB[ZCnum]-CC[ZCnum]*CC[ZCnum]/AA[ZCnum];
4  phhat[ZCnum] = -(EE[ZCnum]-CC[ZCnum]*DD[ZCnum]/AA[ZCnum])
5                                              /delta[ZCnum];
6  fhat[ZCnum] = -CC[ZCnum]/AA[ZCnum]*phhat[ZCnum]-
7                                              DD[ZCnum]/AA[ZCnum];
8  }
```

6.3.3 Implementation of Holdover

For each node, the a sinusoid beacon is generated based on the estimates and transmitted to other nodes. As we discussed before, the transmitted beacon is a periodic extension of the received beacon. Therefore, the phase, $\phi = \hat{\omega} nT + \hat{\theta}$, is a large number since n is large. In order to avoid overflow of using the C standard math function *sin*, we need to keep ϕ within the range $[0, 2\pi]$. Hence, the unwrapping algorithm is needed. However, it is not efficient to unwrap a large ϕ. Thus, we unwrap the phase ϕ under holdover mode. The listing below shows the C function for updating phase in holdover mode.

```
1  void phase_updater(unsigned short ZCnum)
2  {
3  phase[ZCnum] = phase[ZCnum] + fhat[ZCnum]*inv_fs;
4
5  //phase unwrap
6  while (phase[ZCnum] > 2*pi)
7          phase[ZCnum] = phase[ZCnum] - 2*pi;
8  while (phase[ZCnum] < 0)
```

```
9           phase[ZCnum] = phase[ZCnum] + 2*pi;
10  }
```

6.3.4 Implementation of FIR Filter

For the FIR bandpass filter, we use the Matlab to compute the filter coefficients and pre-load them to the DSK. The discrete-time convolution is

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{6.1}$$

where $y[n]$ is the output of a linear, time-invariant, discrete-time system (LTI), $x[n]$ is the input, and $h[n]$ is the unit pulse response. By using the circular buffers, we can implement this convolution in sequence [27]. Here is an example of implementing the FIR filter via the circular buffers.

```
1  /******************FIR Filter*********************/
2
3  if (cont == BL)
4   cont= 0;
5
6  //writes data in right channel into buffer
7  filter[cont] = reading;
8
9  j = 0;
10 input = 0;
11
12 for (i = cont; i>= 0; i--)
13 {
14          input += B[j]*filter[i];
15          j++;
16 }
```

```
17
18 for (i = BL-1;i>cont;i--)
19 {
20         input += B[j]*filter[i];
21         j++;
22 }
```

In the example above, array B stores the filter coefficients, BL is the order of filter, and array $filter$ with size BL stores the last BL received samples.

6.4 Data Analysis Methodology

As discussed in [17], the round-trip synchronization system has multipath channels. Therefore, each source node should delay tracking the beacon until the transient effects of the channel have become negligible. Then, during the steady-state portion of the observation, the source node starts the tracking process and sends the received samples to the state machine for zero crossing detection. The estimates are put into holdover mode prior to the end of the steady-state portion of the beacon. This is illustrated in Figure 6.3.

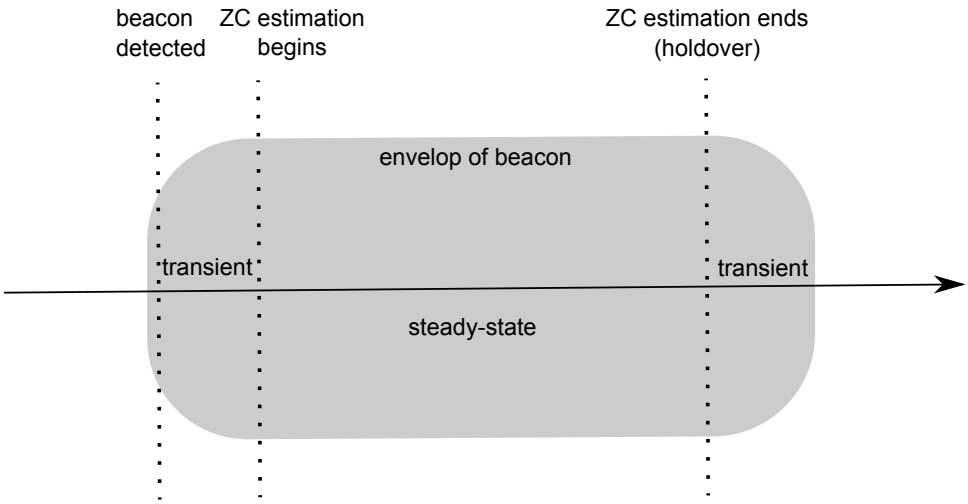

Figure 6.3: Effect of multipath on ZC estimation and holdover.

For the experiment with N wired beamforming tests, the uncompressed .wav recording

of the experiment was transferred to a PC and analysed in MATLAB to generated the statistical results. The "power ratio" ρ of the beamformer is calculated by estimating the power received during beamforming [17]. A power ratio of one means an ideal beamformer without any frequency offset and phase offset. A power ratio of zero corresponds to the case where the carriers completely cancel at the destination. In two source nodes system, the formula for computing power ratio is

$$\rho[n] = \left(\frac{\hat{b}_0^{bf}[n]}{\hat{b}_0^{12}[n] + \hat{b}_0^{21}[n]} \right)^2 . \tag{6.2}$$

The amplitude estimates in each test are obtained via the FFT-Secant MLE described in Section 3.3.

6.5 Experimental Results

Three wired-channel experiments were performed and the experimental results can be founded in Table 6.2. In this test, the central frequency of the beacon is 1020Hz. The

Variable	Experiment 1	Experiment 2	Experiment 3
Min(ρ)	0.9996	0.9995	0.9990
Max(ρ)	1.0000	1.0000	1.0000
Mean(ρ)	0.9998	0.9998	0.9997
Std(ρ)	$7.5866e-5$	$1.1948e-4$	$2.2029e-4$

Table 6.2: Experimental results of two-source wired-channel tests. Each experiment consisted of 100 distributed beamforming tests.

sampling frequency is 16000Hz. For the zero-crossing estimator, the hysteresis parameter is 0.2. A bandpass FIR filter with 64 orders is used. The passband of the filter is $[920, 1120]$Hz. The filter coefficients are generated by Matlab. Each experiment contains 100 tests.

Compared with the proposed experimental results in [17], our proposed estimator has much better results. The minimum and mean of the power ratios are larger than those of the PLL estimator. In addition, our power ratio is more consistent. The worst case of ZC estimator is still 10 times better than the best case of PLL estimator.

Chapter 7

Conclusion

In the first part of this thesis we discussed two classes of maximum likelihood estimation. The first category discussed is FFT-based maximum likelihood estimation (FFT MLE). We based our discussion on the FFT MLE given by Rife and Boorstyn in [8]. The numerical results show that their estimator can only achieve the CRLB at 60dB SNR with a large value of M. In order to reduce the computational complexity, we proposed and discussed four refined approximate maximum likelihood estimators, the FFT-Quad estimator, the FFT-Secant estimator, the FFT-Newton estimator, and the FFT-bisection estimator. The numerical results show that for the small value of M, all the refinements can provide better performance. Also, the refinements can achieve the CRLB at 60dB SNR with less computational complexity.

The second part of this thesis proposed a new linear-regression-based maximum likelihood estimator called the Zero-Crossing Phase and Frequency Estimator. Compared with the well-known estimators in this category, Tretter's estimator [13] and Kay's estimator [14], our proposed estimator has similar performance. In addition, the computational complexity of our approach is less than the others for two main reasons. First, instead of using every received sample, our approach only uses the zero crossings. Second, our approach avoids the arctangent operation since the phase of each zero crossing is known. The refinements of the zero-crossing estimator show better performance than that of Tretter's and Kay's estimators. Finally, our estimator is implemented in the time-slotted round-trip

carrier synchronization system of distributed beamforming using software-defined-radios. Compared with the published results in [17], which use the hybrid Phase Lock Loop as the estimator, our estimator has better and more consistent results.

There are many potential future research opportunities that stem from this thesis. One such opportunity improves the accuracy of interpolation. Although both the simple interpolation and the local linear regression have given a good performance, at high SNR, the error of interpolation causes the mean squared errors to deviate from the CRLB. Therefore, a more accurate algorithm is needed to find the time of each zero crossing.

Appendix A

The Specification of Computer and Matlab for Simulation

A.1 The Specification of Computer for Simulation

Table A.1: The Specification of simulation computer

Operation System (OS)	32-bit Microsoft Windows 7 Enterprise
Processor	Two Intel(R) Core(TM) i7 870 processors with 2.93GHz
Number of Threads	8
Memory	4GB

A.2 The Specification of Matlab Software for Simulation

Table A.2: The Specification of Matlab

Version	7.11.0.584 (32-bit)
Maximum Possible Array	1249MB
Memory Available for All Arrays	1417MB

Appendix B

Source Code of the TMS320C6713 Source Node in the Two-Source System

The following C code is the software implementation of the two-source time-slotted round-trip carrier synchronization system. When the switch 1 on the board is pressed, the DSK board performs as source node 1, whereas switch 2 corresponds to source node 2.

```
1  /* z_c.c by Yizheng Liao 12-OCT-2010 */
2
3  /*
4  This program is used to implement the two-source time-slotted
5  round-trip carrier synchronization system. The C code is loaded to
6  TMS320C6713 Digital Signal Processing Kit (DSK). When the switch 1
7  on the DSK is pressed, the board performs as source node 1. When
8  switch 2 is pressed, the board performs as source node 2.
9  */
10
11 /* Defines */
12 #define CHIP_6713               // define the chip
13 #define thresh 0.2              // hysteresis parameter
14 #define bz 80                   // size of detector buffer
15 #define N 2                     //Number of node
```

```
16 #define PI 3.141592654  //phi
17
18 /* Header Files */
19 #include <stdio.h>
20 #include <stdlib.h>
21 #include <math.h>
22 #include <c6x.h>
23 #include <csl.h>
24 #include <csl_mcbsp.h>
25 #include <csl_irq.h>
26
27 //DSK Configuration Files
28 #include "dsk6713.h"
29 #include "dsk6713_aic23.h"
30 #include "fastrts67x.h"
31
32
33 //contains coeff of filter
34 #include "FIRcoeff.h"
35
36 extern const int BL;
37 extern const double B[65];
38
39 /* Codec Configurations */
40 DSK6713_AIC23_CodecHandle hCodec;
41 // Codec configuration with default settings
42 DSK6713_AIC23_Config config = DSK6713_AIC23_DEFAULTCONFIG;
43
44
45
46
47 /*********Global constant**************/
48
49 /***************Time Table for Each Time Slot*****************/
50 /*For TS0, TS1, TS2*/
51
52 //start of time slot
```

```
53 const unsigned int start_slot = 0;
54
55 //start to listen @ 0.1s
56 const unsigned int start_listen = 1600;
57
58 //stop to listen @ 0.6s
59 const unsigned int stop_listen = 9600;
60
61 //end of time slot @ 1.25s
62 const unsigned int end_slot = 20000;
63
64 //start to play
65 const unsigned int start_play = 0;
66
67 //stop to play @ 1s
68 const unsigned int stop_play = 16000;
69
70 /*For TS3*/
71
72 //start of time slot
73 const unsigned int start_slot_last = 0;
74
75 //wait 0.05s before playing
76 const unsigned int start_play_last = 800;
77
78 //160800; //play 10s
79 const unsigned int stop_play_last = 32800;
80
81 //180000;
82 const unsigned int end_slot_last = 48800;
83 //wait 1.2s for end of time slot
84
85 const double pi = PI;
86 const double fs = 16000.00;                    //sampling frequency
87 const double A = thresh;
88 const double A_i = -thresh;
89 const double max = 32767;
```

```
90  const double det_sum = 0.01;                    //threshold for signal detector
91
92
93  /* Global Variables and Initializations */
94
95  /*FOR each Zero-Crossing estimator*/
96
97  //threshold index
98  unsigned int i1[N], i2[N];
99
100 //sample asscoiated to threshold index
101 double i1_value[N], i2_value[N];
102
103 //tracking variables
104 double AA[N], BB[N], CC[N], DD[N], EE[N],delta[N];
105
106 //zero corssing time and phase
107 double zt[N], ph[N];
108
109 //observation index, state of state machine
110 unsigned int n[N], ind[N], state[N];
111
112 //estimates
113 double fhat[N], phhat[N];
114
115 //max of input beacon
116 double peak[N];
117
118 //1/peak
119 double scale[N];
120
121 /*FOR FIR Filter*/
122
123 // Buffer for filter count
124 unsigned int cont;
125
126 //buffer for filter, size 65
```

```
127 double filter[65];
128
129 /*FOR Detector*/
130
131 //0.5ms for detection
132 double detect_buffer[bz];
133
134 //detection index
135 int detector_index;
136
137 //sum of detection
138 double detector_sum = 0;
139
140
141 /*Other Parameters*/
142 double inv_max;
143 unsigned int counter;
144 unsigned int start;
145 double inv_fs;
146 unsigned short TS;
147 unsigned short node;
148 double acme;
149
150 /*Functions*/
151 interrupt void serialPortRcvISR(void);
152 void ZC_track (double input, unsigned short ZCnum);
153 void ZC_updatepar(unsigned short ZCnum);
154 void ZC_init(unsigned short ZCnum);
155 void ZC_estimation(unsigned short ZCnum);
156 double ZC_play(unsigned short ZCnum);
157 void phase_updater(unsigned short ZCnum);
158
159 void main()
160 {
161     int i = 0;
162
163         DSK6713_init();                 // Initialize the board support library
```

```
164         DSK6713_DIP_init();             // Initializes DIP switches
165         DSK6713_LED_init();             // Initializes LEDs
166
167         hCodec = DSK6713_AIC23_openCodec(0, &config);   // Open the codec
168
169         // Configure buffered serial port for 32-bit operation
170         // This allows transfer of both right and left channels
171         // in one read/write
172         MCBSP_FSETS(SPCR1, RINTM, FRM);
173         MCBSP_FSETS(SPCR1, XINTM, FRM);
174         MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);
175         MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
176
177         //set the sampling rate
178         DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_16KHZ);
179
180
181         // Initialize right and left channel buffers with zeros
182         // So no garbage is in the channel buffers
183         for (i = 0; i < BL; i++)
184         {
185                 filter[i] = 0;
186         }
187
188
189
190         for (i = 0; i < bz; i++)
191         {
192                 detect_buffer[i] = 0.0;
193         }
194
195
196
197         inv_max =1 / max;
198         inv_fs = 1/ fs;
199         ZC_init(0);
200         ZC_init(1);
```

```
201         start = 0;
202         detector_index = 0;
203         detector_sum = 0;
204         counter = 0;
205         cont = 0;
206         acme = 0;
207
208
209         // Globally disables interrupts
210         IRQ_globalDisable();
211         // Enables the NMI interrupt
212         IRQ_nmiEnable();
213         // Maps an event to a physical interrupt
214         IRQ_map(IRQ_EVT_RINT1, 15);
215         // Enables the event
216         IRQ_enable(IRQ_EVT_RINT1);
217         // Globally enables interrupts
218         IRQ_globalEnable();
219
220         //decide the node number
221         if (DSK6713_DIP_get(1) == 0)
222         {
223                 node = 1;
224         }
225         else if (DSK6713_DIP_get(2) == 0)
226         {
227                 node = 2;
228         }
229         else node = 0;
230         DSK6713_LED_on(node);
231
232         // Infinite Loop
233         while(1)
234         {
235
236         }
237
```

```
238 }
239
240 interrupt void serialPortRcvISR()
241 {
242         union {Uint32 combo; short channel[2];} data;
243         short i = 0, j;
244         double input = 0;
245         double output = 0;
246         double reading;
247 /***********************************Read input sample*************************/
248         // Read L+R channels
249         data.combo = MCBSP_read(DSK6713_AIC23_DATAHANDLE);
250         reading = (double)data.channel[0]*inv_max;
251         if (acme < reading)
252                 acme = reading;
253
254
255
256 /*******************************FIR Filter************************/
257
258         if (cont == BL)
259                   cont= 0;
260
261         //writes data in right channel into buffer
262     filter[cont] = reading;
263
264         j = 0;
265         input = 0;
266
267         for (i = cont; i>= 0; i--)
268         {
269                 input += B[j]*filter[i];
270                 j++;
271         }
272
273         for (i = BL-1;i>cont;i--)
274         {
```

```
275                 input += B[j]*filter[i];
276                 j++;
277         }
278
279         cont++;
280
281 /*******************************FIR Filter END**********************/
282
283
284 if (start == 0)
285 {
286
287 /*********************************DETECTOR*********************/
288
289         detect_buffer[detector_index] = input*input;
290
291
292
293
294         if (detector_index < 79)
295         {
296                 detector_sum = detector_sum +
297                         detect_buffer[detector_index] -
298                         detect_buffer[detector_index+1];
299                 detector_index++;
300         }
301         else
302         {
303                 detector_sum = detector_sum +
304                         detect_buffer[detector_index] -
305                         detect_buffer[0];
306                 detector_index = 0;
307         }
308
309
310
311                 if (detector_sum > det_sum)
```

```
312                 {
313                         counter = 0;
314                         start = 1;
315                         ZC_init(0);
316                         ZC_init(1);
317                         TS = 0;
318                 }
319
320 }
321
322 /*********************************DETECTOR END*********************/
323
324 /*****************************State Machine************************/
325 else
326 {
327 if(TS == 0)     //time slot 0
328 {
329         if ((counter <start_listen)&&(counter>=start_slot))  //0.1s wait
330         {
331                 if (counter < 80)
332                 {
333                         detect_buffer[counter] = 0;
334                 }
335                 if (input > peak[0])
336                 {
337                         peak[0] = input;
338                         scale[0] = 1/peak[0];
339                 }
340                 counter++;
341         }
342         else if ((counter >= start_listen) && (counter < stop_listen))
343         //0.5s track
344         {
345                 ZC_track(input,0);
346                 counter++;
347                 //ind[0]++;
348
```

```
349         }
350         else if (counter == stop_listen)
351         {
352                 ZC_estimation(0);
353                 counter++;
354                 ind[0]++;
355         }
356         else if ((counter > stop_listen) && (counter < end_slot))
357         {
358                 ind[0]++;
359                 counter ++;
360         }
361         else if (counter == end_slot)
362         {
363                 ind[0]++;
364                 TS++;
365                 counter = 0;
366         }
367 }
368 else if (TS == 1) //time slot 1
369 {
370         if (node == 1)
371         {
372                 if ((counter>=start_play) && (counter <= stop_play))
373                 {
374
375                         output = ZC_play(0);
376                         data.channel[0] = (short)(output*32000);
377                         data.channel[1] = data.channel[0];
378                         counter++;
379                         ind[0]++;
380                         phase_updater(0);
381
382                         // Write L+R channels
383                         MCBSP_write(DSK6713_AIC23_DATAHANDLE, data.combo);
384                 }
385                 else if ((counter > stop_play) && (counter < end_slot))
```

```
386                 {
387                         counter++;
388                 }
389                 else if (counter == end_slot)
390                 {
391                         counter = 0;
392                         TS++;
393                 }
394         }
395         else if (node == 2)             //time slot 2
396         {
397                 if ((counter <start_listen)&&(counter>=start_slot))
398                 //0.1s wait
399                 {
400                         if (input > peak[1])
401                         {
402                                 peak[1] = input;
403                                 scale[1] = 1/peak[1];
404                         }
405                         counter++;
406                         ind[0]++;
407                         phase_updater(0);
408                 }
409                 else if ((counter >= start_listen) && (counter < stop_listen))
410                 //0.5s track
411                 {
412                         ZC_track(input,1);
413                         counter++;
414                         ind[0]++;
415                         phase_updater(0);
416
417                 }
418                 else if (counter == stop_listen)
419                 {
420                         ZC_estimation(1);
421                         counter++;
422                         phase_updater(0);
```

```
423                         phase[1] = (double)ind[1]*inv_fs*fhat[1] + phhat[1];
424                         phase[1] = phase[1] - 2*pi*510;
425                         while (phase[1] > 2*pi)
426                                 phase[1] = phase[1] - 2*pi;
427                         while (phase[1] < 0)
428                                 phase[1] = phase[1] + 2*pi;
429                         phase_updater(1);
430                         ind[0]++;
431                         ind[1]++;
432                 }
433                 else if ((counter > stop_listen) && (counter < end_slot))
434                 {
435                         ind[0]++;
436                         ind[1]++;
437                         counter ++;
438                         phase_updater(0);
439                         phase_updater(1);
440
441                 }
442                 else if (counter == end_slot)
443                 {
444                         phase_updater(0);
445                         phase_updater(1);
446                         ind[0]++;
447                         ind[1]++;
448                         TS++;
449                         counter = 0;
450                 }
451         }
452
453 }
454
455 else if (TS == 2)       //time slot 2
456 {
457         if(node == 1)
458         {
459                 if ((counter <start_listen)&&(counter>=start_slot))
```

```
460                 //0.1s wait
461                 {
462                         if (input > peak[1])
463                         {
464                                 peak[1] = input;
465                                 scale[1] = 1/peak[1];
466                         }
467                         counter++;
468                 }
469                 else if ((counter >= start_listen) && (counter < stop_listen))
470                 //0.5s track
471                 {
472                         ZC_track(input,1);
473                         counter++;
474
475                 }
476                 else if (counter == stop_listen)
477                 {
478                         ZC_estimation(1);
479                         counter++;
480                         ind[1]++;
481
482                         phase[1] = (double)ind[1]*inv_fs*fhat[1] + phhat[1];
483                         phase[1] = phase[1] - 2*pi*510;
484                         while (phase[1] > 2*pi)
485                                 phase[1] = phase[1] - 2*pi;
486                         while (phase[1] < 0)
487                                 phase[1] = phase[1] + 2*pi;
488                         phase_updater(1);
489                 }
490                 else if ((counter > stop_listen) && (counter < end_slot))
491                 {
492                         ind[1]++;
493                         counter ++;
494                         phase_updater(1);
495                 }
496                 else if (counter == end_slot)
```

```
497                 {
498                         ind[1]++;
499                         phase_updater(1);
500                         TS++;
501                         counter = 0;
502                 }
503         }
504         else if (node == 2)
505         {
506                 if ((counter>=start_play) && (counter <= stop_play))
507                 {
508                         output = ZC_play(0);
509
510                         data.channel[0] = (short)(output*32000);
511                         data.channel[1] = data.channel[0];
512                         counter++;
513                         ind[0]++;
514                         ind[1]++;
515                         phase_updater(0);
516                         phase_updater(1);
517
518                         // Write L+R channels
519                         MCBSP_write(DSK6713_AIC23_DATAHANDLE, data.combo);
520                 }
521                 else if ((counter > stop_play) && (counter < end_slot))
522                 {
523                         counter++;
524                         ind[1]++;
525                         phase_updater(1);
526                 }
527                 else if (counter == end_slot)
528                 {
529                         counter = 0;
530                         phase_updater(1);
531                         ind[1]++;
532                         TS++;
533                 }
```

```
534         }
535 }
536 else if (TS == 3)               //time slot 3
537 {
538         if ((counter >= start_slot_last) && (counter < start_play_last))
539         {
540                 ind[1]++;
541                 phase_updater(1);
542                 counter++;
543         }
544         if ((counter>=start_play_last) && (counter <= stop_play_last))
545                 {
546
547                         output = ZC_play(1);
548                         phase_updater(1);
549
550                         data.channel[0] = (short)(output*32000);
551                         data.channel[1] = data.channel[0];
552                         counter++;
553                         ind[1]++;
554
555                         MCBSP_write(DSK6713_AIC23_DATAHANDLE, data.combo);
556                 }
557                 else if ((counter > stop_play_last) && (counter < end_slot_last))
558                 {
559                         counter++;
560                 }
561                 else if (counter == end_slot_last)
562                 {
563                         counter = 0;
564                         TS = 0;
565                         start = 0;
566                         detector_index = 0;
567                         detector_sum = 0;
568                 }
569 }
570 }
```

```
571 }
572
573
574 void ZC_track (double input, unsigned short ZCnum)
575 /*
576 This function is used to update the tracking variables
577 of the linear regression for each estimator.
578
579 input:  the read sample from ADC
580                 The sample is compared with the hysteresis
581                 parameter and used to decide if the state
582                 is changed.
583
584 ZCnum:  index of the estimator
585 */
586 {
587 double slope = 0;
588 int temp = 0;
589
590 input = input * scale[ZCnum];
591
592 if (ind[ZCnum] == 0)
593 {
594         if (input > A)
595           {
596                 i1[ZCnum] = 0;
597                 i1_value[ZCnum] = input;
598                 state[ZCnum] = 1;
599           }
600           else if (input< A_i)
601           {
602                 i2[ZCnum] = 0;
603                 i2_value[ZCnum] = input;
604                 state[ZCnum] = 2;
605           }
606           else state[ZCnum] = 0;
607 }
```

```
608 else
609 {
610         if(state[ZCnum] == 0)
611         {
612                 if(input > A)
613                 {
614                         i1[ZCnum] = ind[ZCnum];
615                         i1_value[ZCnum] = input;
616                         state[ZCnum] = 1;
617                 }
618                 else if(input<A_i)
619                 {
620                         i2[ZCnum] = ind[ZCnum];
621                         i2_value[ZCnum] = input;
622                         state[ZCnum] = 2;
623                 }
624         }
625         else if (state[ZCnum] == 1)
626         {
627                 if(input > A)
628                 {
629                         i1[ZCnum] = ind[ZCnum];
630                         i1_value[ZCnum] = input;
631                 }
632                 else if ((input > A_i) && (input < A))
633                 {
634                         state[ZCnum] = 3;
635                 }
636                 else if (input < A_i)
637                 {
638                         i2[ZCnum] = ind[ZCnum];
639                         i2_value[ZCnum] = input;
640                         slope = (i2_value[ZCnum] - i1_value[ZCnum])*fs;
641                         temp = i2[ZCnum] - i1[ZCnum];
642                         slope = slope / (double)(temp);
643                         zt[ZCnum] = (double)i1[ZCnum]*inv_fs;
644                         zt[ZCnum] = zt[ZCnum] - i1_value[ZCnum]/slope;
```

```
645                         n[ZCnum]++;
646                         if (n[ZCnum] >= 2)
647                         {
648                                 ph[ZCnum] = ph[ZCnum]+pi;
649                         }
650                         else ph[ZCnum] = pi;
651                         state[ZCnum] = 2;
652
653                         ZC_updatepar(ZCnum);
654                 }
655         }
656         else if (state[ZCnum] == 2)
657         {
658                 if (input < A_i)
659                 {
660                         i2[ZCnum] = ind[ZCnum];
661                         i2_value[ZCnum] = input;
662                 }
663                 else if ((input > A_i) && (input < A))
664                 {
665                         state[ZCnum] = 4;
666                 }
667                 else if (input > A)
668                 {
669                         i1[ZCnum] = ind[ZCnum];
670                         i1_value[ZCnum] = input;
671                         slope = (i2_value[ZCnum] - i1_value[ZCnum])*fs;
672                         temp = i2[ZCnum] - i1[ZCnum];
673                         slope = slope / (double)(temp);
674                         zt[ZCnum] = (double)i2[ZCnum]*inv_fs;
675                         zt[ZCnum] = zt[ZCnum] - i2_value[ZCnum]/slope;
676                         n[ZCnum]++;
677                         if (n[ZCnum]>=2)
678                                 ph[ZCnum] += pi;
679                         else ph[ZCnum] = 0;
680
681                         state[ZCnum] = 1;
```

```
682
683                     ZC_updatepar(ZCnum);
684             }
685     }
686     else if (state[ZCnum] == 3)
687     {
688             if (input > A)
689             {
690                     i1[ZCnum] = ind[ZCnum];
691                     i1_value[ZCnum] = input;
692                     state[ZCnum] = 1;
693             }
694             else if ((input > A_i) && (input < A))
695             {
696                     state[ZCnum] = 3;
697             }
698             else if (input < A_i)
699             {
700                     i2[ZCnum] = ind[ZCnum];
701                     slope = (i2_value[ZCnum] - i1_value[ZCnum])*fs;
702                     temp = i2[ZCnum] - i1[ZCnum];
703                     slope = slope / (double)(temp);
704                     zt[ZCnum] = (double)i1[ZCnum]*inv_fs;
705                     zt[ZCnum] = zt[ZCnum] - i1_value[ZCnum]/slope;
706                     n[ZCnum]++;
707                     if (n[ZCnum]>=2)
708                     {
709                             ph[ZCnum] = ph[ZCnum] + pi;
710                     }
711                     else ph[ZCnum] = pi;
712
713                     state [ZCnum]= 2;
714                     ZC_updatepar(ZCnum);
715             }
716     }
717     else if (state[ZCnum] == 4)
718     {
```

```
719                     if(input < A_i)
720                     {
721                             i2[ZCnum] = ind[ZCnum];
722                             i2_value[ZCnum] = input;
723                             state[ZCnum] = 2;
724                     }
725                     else if ((input > A_i) && (input < A))
726                     {
727                             state[ZCnum] = 4;
728                     }
729                     else if (input > A)
730                     {
731                             i1[ZCnum] = ind[ZCnum];
732                             i1_value[ZCnum] = input;
733                             slope = (i2_value[ZCnum] - i1_value[ZCnum]) *fs;
734                             temp = i2[ZCnum] - i1[ZCnum];
735                             slope = slope / (double)(temp);
736                             zt[ZCnum] = (double)i2[ZCnum]*inv_fs;
737                             zt[ZCnum] = zt[ZCnum] - i2_value[ZCnum]/slope;
738                             n[ZCnum]++;
739                             if (n[ZCnum]>= 2)
740                             {
741                                     ph[ZCnum] = ph[ZCnum]+pi;
742                             }
743                             else ph[ZCnum] = 0;
744
745                             state[ZCnum] = 1;
746                             ZC_updatepar(ZCnum);
747                     }
748
749             }
750     }
751
752     ind[ZCnum]++;
753     }
754
755
```

```
756 void ZC_updatepar(unsigned short ZCnum)
757 /*
758 This function is used to update the tracking variables
759
760 ZCnum: index of the estimator
761 */
762 {
763         AA[ZCnum] = AA [ZCnum]+ zt[ZCnum]*zt[ZCnum];
764         BB[ZCnum]++;
765         CC[ZCnum] = CC[ZCnum] + zt[ZCnum];
766         DD[ZCnum] = DD[ZCnum] - zt[ZCnum]*ph[ZCnum];
767         EE[ZCnum] = EE[ZCnum] - ph[ZCnum];
768 }
769
770
771 void ZC_estimation(unsigned short ZCnum)
772 /*
773 This function is used to estimate the unknown parameters based on
774 the tracking variables.
775
776 ZCnum: index of the estimator
777 */
778 {
779         delta[ZCnum] = BB[ZCnum]  - CC[ZCnum] *CC[ZCnum] /AA[ZCnum];
780         phhat[ZCnum] = -(EE[ZCnum]-CC[ZCnum]*DD[ZCnum]/AA[ZCnum])/delta[ZCnum];
781         fhat[ZCnum]  = -CC[ZCnum] /AA[ZCnum] *phhat[ZCnum]-DD[ZCnum]/AA[ZCnum];
782 }
783
784 void ZC_init(unsigned short ZCnum)
785 /*
786 This function is used to initalize the parameters.
787
788 ZCnum: index of the estimator
789 */
790 {
791
792         i1[ZCnum] = 0;
```

```
793          i2[ZCnum] = 0;
794          i1_value[ZCnum] = 0;
795          i2_value[ZCnum] = 0;
796          AA[ZCnum] = 0;
797          BB[ZCnum] = 0;
798          CC[ZCnum] = 0;
799          DD[ZCnum] = 0;
800          EE[ZCnum] = 0;
801          delta[ZCnum] = 0;
802          zt[ZCnum] = 0;
803          ph[ZCnum] = 0;
804          n[ZCnum] = 0;
805          ind[ZCnum] = 0;
806          state[ZCnum] = 0;
807          fhat[ZCnum] = 0;
808          phhat[ZCnum] = 0;
809          peak[ZCnum]= -1.0;   //max of input beacon
810          scale[ZCnum] = 1.0; //1/peak
811          acme = 0;
812  }
813
814  double ZC_play(unsigned short ZCnum)
815  /*
816  This function is used to generate the sinosoid wave
817  based on the estimates.
818
819  ZCnum: index of the estimator
820  */
821  {
822          double output;
823          output = sindp(phase[ZCnum]);
824
825          return output;
826  }
827
828
829  void phase_updater(unsigned short ZCnum)
```

```
830 /*
831 This function is used to update the phase under
832 holdover mode.
833
834 ZCnum: index of the estimator
835 */
836 {
837         phase[ZCnum] = phase[ZCnum] + fhat[ZCnum]*inv_fs;
838         while (phase[ZCnum] > 2*pi)
839                 phase[ZCnum] = phase[ZCnum] - 2*pi;
840         while (phase[ZCnum] < 0)
841                 phase[ZCnum] = phase[ZCnum] + 2*pi;
842 }
```

Appendix C

Approximate Maximum Likelihood Estimation using Secant Method Matlab Source Code

The following MATLAB code is used to analyze the recording in *.WAV format by using the FFT-Secant MLE described in Section 3.3. By using this code, the amplitude estimate in each timeslot can be calculated and thus the power ratio ρ of the beamformer is computed.

C.1 Main Function

```
% LYZ Sept-21-201
% Use Secant ML estimator to estiamte
% for phase, frequency, and amplitude

clear all;
clc;
% ----------------------------------------
% USER PARAMETERS
% ----------------------------------------
M = 2;              % number of nodes
```

```
11 nfft = 2^16;         % number of points in estimator FFT
12 N = 100;             % number of tests
13 T = 0.6;             % duration of signal used for estimates
14 j = sqrt(-1);
15 delay = 0.13;            % time before the beacon started
16 period = 7;                      % length of each test
17 tstart = 0.2;            % start time to cut signal
18 tend = tstart+T;         % end time to cut signal
19 D = 1.25;            % time difference between successive signals
20 D_last = 2;                      % time different between the last beacon
21                                          % and the 2nd last beacon
22 el = 1e-7;                       % error bound for iteration
23 slot = [0 D 2*D 2*D+D_last];
24 slot = slot + delay;
25
26
27
28 % ----------------------------------------
29
30 format compact
31
32 % read datafile (variables y and fs)
33 % load .mat
34 [y,fs] = wavread('cut4.wav');
35 ahat = zeros(2*M,N);
36 omegahat = zeros(2*M,N);
37 thetahat = zeros(2*M,N);
38 counter = zeros(2*M,N);
39
40
41
42 tic
43
44 for i = 1:N
45         for k = 1:2*M
46                 t0 = slot(k)+(i-1)*period;
47                 sigstartindex = round((t0+tstart)*fs);
```

```
48                  sigendindex = round((t0+tend)*fs)-1;
49                  sig = y(sigstartindex:sigendindex);
50          NN = length(sigstartindex:sigendindex);
51          n0 = sigstartindex;
52                  startindex = round(((k-1)*period+tstart)*fs);
53                  endindex = round(((k-1)*period+tend)*fs)-1;
54                  NN = length(sig);
55                  [omegahat(k,i),thetahat(k,i),ahat(k,i),ifconv] = ...
56          ML_est(sig',nfft,NN,el,1/fs,n0);
57
58          if ifconv == 0,
59              break;
60          end
61
62      end
63
64      if ifconv == 0,
65          continue;
66      end
67          [i toc 4]
68  end
69  ampratio = ahat(end,:)./(ahat(2,:)+ahat(3,:));
70  powratio = ampratio.^2;
71  minimum = min(powratio)
72  maximum = max(powratio)
73  average = mean(powratio)
74  sd = (var(powratio)^0.5)
75  save ahat_cut14.mat ahat ampratio powratio omegahat thetahat counter
```

C.2 Function of Implementing the FFT-Secant MLE

This function is used to implement the FFT-Secant MLE.

```
1  function [omega,theta,ahat,flag_conv ] = ML_est(z,nfft,N,e,T,n0 )
2  %UNTITLED3 Summary of this function goes here
3  %   Detailed explanation goes here
4
5  A = fft(z,nfft)/N;
```

```
[peak,k] = max(abs(A(1:nfft/2)));

omega1 = k*2*pi/nfft/T;
omega2 = (k-2)*2*pi/nfft/T;
y1 = dA_fun(N,z,omega1,1/T);
y2 = dA_fun(N,z,omega2,1/T);

delta = 1;
jj= 2;
error = 0;
while abs(delta)>e
    jj = jj+1;

    delta = y1 * (omega1 - omega2)/(y1-y2);
    omega2 = omega1;
    y2 = y1;
    omega1 = omega1 - delta;

    y1 = dA_fun(N,z,omega1,1/T);
    if (jj > 2000)
        error = 1;
        break;
    end

end

if error == 1
    flag_conv = 0;
    printf('Error\n')
else
    flag_conv = 1;
end

omega = omega1;
theta = angle(exp(-1j*(omega1*n0*T))*...
                A_fun(N,z,omega1,1/T));
ahat = abs(A_fun(N,z,omega1,1/T));
```

```
43
44 end
```

C.3 Function of Computing $A(\omega)$

This function is used to compute (2.10).

```
1  function value = A_fun(N,xx,w,fs)
2  % Compute the likelihood function A(omega) from Rife's paper
3
4
5  % Input Paramter:
6  % N: length of signal
7  % xx: input signal
8  % w: omega
9  % fs: sampling frequency
10
11 % Output Parameter:
12 % value: A(k)
13
14 %w = 2*pi*k*fs/nfft;
15
16 T = 1/fs;
17 j = sqrt(-1);
18 nn = 0:N-1;
19
20 value = (1/N)*sum(xx.*exp(-j*nn*w*T));
21
22
23 end
```

C.4 Function of Computing $F^{'}(\omega)$

This function is used to compute (3.18).

```
1  function da = dA_fun( N,xx,w,fs)
```

```
2  % Input Paramter:
3  % N: length of signal
4  % xx: input signal
5  % w: omega
6  % fs: sampling frequency
7
8  % Output Parameter:
9  % da: A'(k)
10
11 %w = 2*pi*k*fs/nfft;
12
13 T = 1/fs;
14
15 n = 0:N-1;
16 sine = sin(T*n*w);
17 cose = cos(T*n*w);
18
19 x = real(xx);
20 y = imag(xx);
21
22 B = sum(x.*cose+y.*sine);
23 B = B/N;
24 C = sum(x.*sine - y.*cose);
25 C = -C/N;
26
27 dB = sum(n.*(y.*cose - x.*sine));
28 dB = dB*T/N;
29 dC = sum(n.*(x.*cose + y.*sine));
30 dC = -T/N*dC;
31
32 da = 2*B*dB+2*C*dC;
33
34
35 end
```

C.5 Function of Phase Unwrapping

This function is used unwrap the phase to the range $(-\pi, \pi]$.

```
1 function y = dounwrap(x)
2 % doing phase unwrapping
3
4 y = x;
5 I = find(x>pi);
6 y(I) = y(I)-2*pi;
7 I = find(x<-pi);
8 y(I) = y(I)+2*pi;
```

Bibliography

[1] C. Johnson, W. Sethares, and A. Klein, *Software Receiver Design: Build Your Own Digital Communications System in Five Easy Steps*. 2011.

[2] M. Skolnik, "Introduction to radar systems," 1980.

[3] W. Knight, R. Pridham, and S. Kay, "Digital signal processing for sonar," *Proceedings of the IEEE*, vol. 69, no. 11, pp. 1451–1506, 1981.

[4] B. Widrow and S. Stearns, "Adaptive signal processing," *Englewood Cliffs, NJ, Prentice-Hall, Inc., 1985, 491 p.*, vol. 1, 1985.

[5] R. Mudumbai, D. Brown, U. Madhow, and H. Poor, "Distributed transmit beamforming: challenges and recent progress," *Communications Magazine, IEEE*, vol. 47, no. 2, pp. 102–110, 2009.

[6] J. DiBiase, H. Silverman, and M. Brandstein, "8 Robust Localization in Reverberant Rooms," *Microphone arrays: signal processing techniques and applications*, p. 157, 2001.

[7] K. Varma, *Time-delay-estimate based direction-of-arrival estimation for speech in reverberant environments*. PhD thesis, Citeseer, 2002.

[8] D. Rife and R. Boorstyn, "Single-tone parameter estimation from discrete-time observations," *IEEE Transactions on Information Theory*, vol. 20, no. 5, pp. 591–598, 1974.

[9] D. Rife, *Digital tone parameter estimation in the presence of Gaussian noise.* PhD thesis, Polytechnic Institute of Brooklyn, 1973.

[10] T. Abatzoglou, "A fast maximum likelihood algorithm for frequency estimation of a sinusoid based on Newton's method," *IEEE transactions on acoustics, speech, and signal processing*, vol. 33, no. 1, pp. 77–89, 1985.

[11] M. Macleod, "Fast nearly ML estimation of the parameters of real or complex single tones or resolved multiple tones," *Signal Processing, IEEE Transactions on*, vol. 46, no. 1, pp. 141–148, 1998.

[12] D. Brown III, Y. Liao, and N. Fox, "Low-Complexity Real-Time Single-Tone Phase and Frequency Estimation," *IEEE Military Communication*, 2010.

[13] S. Tretter, "Estimating the frequency of a noisy sinusoid by linear regression (Corresp.)," *Information Theory, IEEE Transactions on*, vol. 31, no. 6, pp. 832–835, 2002.

[14] S. Kay, "A fast and accurate single frequency estimator," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 12, pp. 1987–1990, 2002.

[15] U. Mengali and M. Morelli, "Data-aided frequency estimation for burst digital transmission," *Communications, IEEE Transactions on*, vol. 45, no. 1, pp. 23–25, 1997.

[16] D. Kim, M. Narasimha, and D. Cox, "An improved single frequency estimator," *Signal Processing Letters, IEEE*, vol. 3, no. 7, pp. 212–214, 1996.

[17] M. Ni, *An Optimized Software-Defined-Radio Implementation of Time-Slotted Carrier Synchronization for Distributed Beamforming.* PhD thesis, WORCESTER POLYTECHNIC INSTITUTE, 2011.

[18] S. Kay, *Fundamentals of statistical signal processing: estimation theory.* 1993.

[19] M. Fowler, "Phase-based frequency estimation: A review," *Digital Signal Processing*, vol. 12, no. 4, pp. 590–615, 2002.

[20] D. Brown, B. Zhang, B. Svirchuk, and M. Ni, "An experimental study of acoustic distributed beamforming using round-trip carrier synchronization," in *Phased Array*

Systems and Technology (ARRAY), 2010 IEEE International Symposium on, pp. 316–323, IEEE.

[21] K. Atkinson, "An introduction to numerical analysis. 1989," *New York*, p. 528.

[22] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*. Springer Verlag, 2007.

[23] F. Hildebrand, *Introduction to numerical analysis*. Dover Pubns, 1987.

[24] G. Allaire and S. Kaber, *Numerical linear algebra*. Springer Verlag, 2008.

[25] S. Rajan, S. Wang, R. Inkol, and A. Joyal, "Efficient approximations for the arctangent function," *Signal Processing Magazine, IEEE*, vol. 23, no. 3, pp. 108–111, 2006.

[26] V. Friedman, "A zero crossing algorithm for the estimation of the frequency of a single sinusoid in white noise," *Signal Processing, IEEE Transactions on*, vol. 42, no. 6, pp. 1565–1569, 2002.

[27] S. Tretter, *Communication System Design Using DSP Algorithms: With Laboratory Experiments for the TMS320C6713 DSK*. Springer Verlag, 2008.

[28] D. Brown and H. Poor, "Time-slotted round-trip carrier synchronization for distributed beamforming," *Signal Processing, IEEE Transactions on*, vol. 56, no. 11, pp. 5630–5643, 2008.

[29] N. Kehtarnavaz, *Real-time digital signal processing based on the TMS320C6000*. Newnes, 2005.