School of Engineering

Faculty of Science

Theory and Realization of Novel Algorithms for Random Sampling in Digital Signal Processing

by

King Chuen LO

A thesis submission in fulfilment of the degree of Doctor of Philosophy

University of Durham

June 1996

Supervisor : Professor A. Purvis

Theory and Realization of Novel Algorithms for Random Sampling in Digital Signal Processing

King Chuen LO

Abstract

Random sampling is a technique which overcomes the alias problem in regular sampling. The randomization, however, destroys the symmetry property of the transform kernel of the discrete Fourier transform. Hence, when transforming a randomly sampled sequence to its frequency spectrum, the Fast Fourier transform cannot be applied and the computational complexity is N^2.

The objectives of this research project are (1) To devise sampling methods for random sampling such that computation may be reduced while the anti-alias property of random sampling is maintained : Two methods of inserting limited regularities into the randomized sampling grids are proposed. They are parallel additive random sampling and hybrid additive random sampling, both of which can save at least 75% of the multiplications required. The algorithms also lend themselves to the implementation by a multiprocessor system, which will further enhance the speed of the evaluation. (2) To study the auto-correlation sequence of a randomly sampled sequence as an alternative means to confirm its anti-alias property : The anti-alias property of the two proposed methods can be confirmed by using convolution in the frequency domain. However, the same conclusion is also reached by analysing in the spatial domain the auto-correlation of such sample sequences. A technique to evaluate the auto-correlation sequence of a randomly sampled sequence with a regular step size is proposed. The technique may also serve as an algorithm to convert a randomly sampled sequence to a regularly spaced sequence having a desired Nyquist frequency. (3) To provide a rapid spectral estimation using a coarse kernel : The approximate method proposed by Mason in 1980, which trades the accuracy for the speed of the computation, is introduced for making random sampling more attractive. (4) To suggest possible applications for random and pseudo-random sampling : To fully exploit its advantages, random sampling has been adopted in measurement

instruments where computing a spectrum is either minimal or not required. Such applications in instrumentation are easily found in the literature. In this thesis, two applications in digital signal processing are introduced. (5) To suggest an inverse transformation for random sampling so as to complete a two-way process and to broaden its scope of application. Apart from the above, a case study of realizing in a transputer network the prime factor algorithm with regular sampling is given in Chapter 2 and a rough estimation of the signal-to-noise ratio for a spectrum obtained from random sampling is found in Chapter 3.

Although random sampling is alias-free, problems in computational complexity and noise prevent it from being adopted widely in engineering applications. In the conclusions, the criteria for adopting random sampling are put forward and the directions for its development are discussed.

Acknowledgements

First of all, I would like to express my gratitude to my supervisor, Professor A. Purvis, who has shared with me his insight into the topic of random sampling and provided me invaluable guidance to conduct the research.

I am grateful to the Staff Development Committee of The Hong Kong Polytechnic University for the support of funding related to this programme.

Last but not least, I would like to thank Professor P. Mars and Professor J.S.L. Wong, without whom this programme would never have been realized.

Declaration

I hereby declare that the work reported in this thesis was undertaken by myself, that the research has not been previously submitted for a degree, and that all sources of information have been duly acknowledged.

Copyright

Contents

CHAPTER 1

INTRODUCTION

When a continuous-time, analog signal is to be analysed for its frequency spectrum by a digital system, the signal is usually sampled at regular time intervals and the corresponding signal level is quantized. After the completion of the sampling process, a sequence of discrete- valued samples of the original signal taken within a finite period of time is obtained. On this sequence of sample data, the Discrete Fourier Transform (DFT), or the Fast Fourier Transform (FFT) can be applied to evaluate the frequency components accordingly. The FFT, of which the complexity is N log N [1], is in general the most efficient algorithm for computing the frequency components.

Although regular sampling is widely adopted, it is not without drawback. Shannon's (or Nyquist's) theorem states that for regular sampling, the sampling frequency must be at least twice the value of the highest frequency of the sampled signal; otherwise aliases will occur. Aliases are in fact a by-product of regular sampling. Their occurrences generate ambiguities in the spectral analysis of a signal. To avoid such confusion, the signal to be sampled is usually treated by an anti-alias low-pass filter before the sampling process is performed. However, practical difficulties may arise in using the anti-alias filter. Suppose a wide-band signal is to be sampled, an anti-alias filter which has a wide pass-band is required. It follows that a high sampling frequency of at least twice the cut-off frequency of the filter is needed and fast hardware must be used. Another approach is that the wide-band signal is separated

into different non-overlapping frequency bands by a bank of band-pass filters so that each band of the signal can be sampled by a lower sampling frequency; hence the advantages of sub-Nyquist sampling can be acquired. When designing the filters, one has to compromise between the ripple within the pass-band, the attenuation in the stop band, the slope of the roll-off at the cut-off frequency, etc. To implement a satisfactory banks of such filters, especially to fulfill the requirement of non-overlapping frequency bands, is not a simple job at all.

The objective of sub-Nyquist sampling is to sample an input signal by a frequency lower than the Nyquist limit so that the sampling rate may be lowered and inexpensive hardware may be used in the process. Measurement instruments have actually been built based on this method [2,3]. Sub-Nyquist sampling can be achieved by two different approaches: (i) using regular sampling intervals and (ii) using randomized sampling intervals. In principle, the former approach adopts a multi-sampling system having two or three sampling frequencies to treat an incoming signal having a single frequency. Then, judging from the locations of the aliases created by the respective samplers, the frequency of the input signal can be determined. For irregular or random sampling, a suitable random variable is added to a regular sampling grid, e.g. $t_i = i.T + \tau_i$, where t is the sampling time, T a regular sampling interval, τ a random variable and i an integer. In theory random sampling is "continuous" sampling, hence no Nyquist limit exists when evaluating the frequency components. Instead of standing as sharp spikes in the spectrum, the aliases are turned into a broadband noise, which can be distinguished from a signal. In practice, however, the word-length of the digital system or computer processing the signal sets a bound

to the spectrum to be analysed. These two approaches will be discussed in detail in Chapter 3.

Compared to the conventional regular sampling method, random sampling overcomes the aliasing problem and enjoys the advantages of sub-Nyquist sampling. There are certainly drawbacks for this method, among which is the heavy computational effort in calculating the spectrum. For regular sampling, there are trigonometrical symmetries found in the kernel of the transform. These trigonometrical symmetries can be utilized to devise efficient computational algorithms such as the Fast Fourier Transform (FFT), Prime Factor Algorithm, etc. To illustrate this point, a review of some conventional algorithms for DFT is included in chapter 2. Because the symmetry in timing is obviously destroyed as random sampling is adopted, no regularities occur in the kernel. In computing the spectrum, the direct Fourier calculation, in which a different 'random' exponential term is multiplied to each data point, must be performed. The complexity of the computation is thus N^2. If a higher speed of computation is desired, some sort of regularity in timing must be inserted into the sampling process, but to such an extent that the anti-alias property of random sampling is still maintained. To achieve this objective, two novel algorithms are introduced. They are to interlace and to concatenate several suitable random sampling sequences to form a resultant sequence for taking sample points.

The first approach, i.e. the interlacing method, is called the parallel random sampling. Its computational algorithm exploits the trigonometrical symmetry to reduce up to 87% of the multiplications required in computing the first band of frequency components. For subsequent bands, the saving increases with the number of sampling blocks used. The whole process, from sampling to computation, can be

implemented by a multiprocessor system. With this sampling method, bursts of noise, which are the residues of the aliases, appear in the spectrum. Although these bursts can be used as another means to identify the input frequency, one may want to eliminate them. The second approach, which is the hybrid additive random sampling, is devised to give a background noise nearly as "clean" as the genuine random sampling method. The computational algorithm derived from this method saves at least 75% of the multiplications required and it can be implemented in a modular form. The reconstructed spectrum offers a signal-to-noise ratio close to that of genuine random sampling. Chapter 4 and chapter 5 will elaborate on these two methods respectively.

The Wiener -Khinchine relation states that for a stationary random signal x(n):

$$S_{xx}(\omega) = \sum_{k=-\infty}^{\infty} R_{xx}(k) \exp(-j\omega k)$$

where $S_{xx}(\omega)$is the power spectrum and Rxx(k) is the auto-correlation of the signal. In words, the power spectrum of a sequence is the Fourier transform of its auto-correlation. Hence studying the auto-correlation of a random sampling sequence helps to explain its anti-alias property. Chapter 6 will show that the power spectrum of a sequence of data obtained from random sampling can be estimated from the circular auto-correlation of the sequence at *regular* time intervals.

In general, it takes more time for a computer to perform a multiplication than an addition; therefore many fast computational algorithms for the DFT aim at reducing the number of multiplications. Gaster and Roberts [4] proposed an approximate method for estimating the DFT which uses only two levels (namely, -1 and +1) to represent the kernel of the transform. Mason [5] also suggested to round off the trigonometric terms to three levels (namely, -1,0 and +1). By doing so, all

multiplications are changed into additions or subtractions, so that the speed of computation is enhanced at the expense of the accuracy of the results. These round-off methods are justifiable to be applied to the data obtained by random sampling since the frequency components, even if computed by the exact DFT, in practice will not equal exactly to those reconstructed from regular sampling. There will be more information and discussion about this topic in chapter 7.

As the FFT is in general the most efficient and readily available computational algorithm and it is based on a regular sampling grid, we expect to see that many engineering applications adopt the regular sampling approach. Because the advantage of random sampling is its anti-alias property, any applications which can benefit from this property are suitable to adopt the random sampling scheme. Examples in the areas of instrumentation and digital signal processing will be included in chapter 3 and chapter 8 respectively.

Although one may perform a spectral evaluation by transforming a randomly sampled sequence from the spatial domain to the frequency domain, the direct inverse transform is made impossible by the spectral noise generated by random sampling. Chapter 9 will provide a different approach to realize such an inverse operation. When random sampling is adopted, the designer has to compromise between the bandwidth of the spectrum, the accuracy of the amplitude of the signal recovered, the background noise level, the computational effort, etc. For example, if the computational effort is reduced by using either the parallel random sampling or the hybrid additive random sampling, the background noise level tends to increase. In the conclusions contained in chapter 10, the properties of random sampling will be summarized. The

performance of random sampling and regular sampling will be compared. Finally, some possible directions for the development of random sampling will be suggested.

CHAPTER 2

REVIEW OF SOME CONVENTIONAL ALGORITHMS FOR FREQUENCY ANALYSIS

The study of computational algorithms for the DFT is a mature subject. Cooley and Tukey published the well-known FFT as early as 1965 [6]. Their approach is in fact a radix-2 algorithm, which requires the length of the transform to be a power of two, and the calculation is done in the field of complex number. There are many transforms closely related to the DFT; to name a few, there are the Discrete Cosine Transform (DCT), the Discrete Sine Transform (DST) and the Discrete Hartley Transform (DHT), which can be obtained from the DFT by suitable algebraic manipulations. Different directions of research in this area were also pursued, e.g. using prime numbers for the sequence length of the transform [7,8], or using mathematical structures in a finite field [9]. There is, however, a common ground for all the above algorithms. They assume that the sample data are recorded at regular intervals so that in the kernel of the transform, symmetry property exists. This property can be exploited to reduce the complexity of the computation.

Since there are so many computational algorithms for the DFT based on regular sampling, it is not the purpose of this chapter to list them exhaustively. Only a few representative algorithms will be included here to give a flavour to this topic. Emphasis will be put on the computational complexity and the use of trigonometric symmetry to form a computational algorithm so as to set the scene for the discussion of random sampling.

2.1 The Discrete Fourier Transform

Let us consider the sampling in the frequency domain of a discrete-time signal. Recall that an aperiodic signal with finite energy has a continuous spectrum. For such an aperiodic discrete-time signal x(n), its Fourier transform is given by :

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)\, e^{-j\omega n} \qquad \textbf{(2-1)}$$

Suppose $X(\omega)$ is sampled periodically in frequency at a spacing of $\delta\omega$ radians between successive samples. Since $X(\omega)$ is periodic with period 2π, only samples in the fundamental frequency range are necessary. For convenience, let N equidistant samples be taken in the interval $0 \le \omega < 2\pi$ with spacing $\delta\omega = 2\pi / N$. If we evaluate (2-1) at $\omega = 2\pi k / N$, we obtain:

$$X\left(\frac{2\pi}{N}k\right) = \sum_{n=-\infty}^{\infty} x(n)\, e^{-j2\pi nk/N} \qquad k = 0,1,...,N-1 \qquad \textbf{(2.-2)}$$

The summation in (2-2) can be written as an infinite number of summations :

$$X\left(\frac{2\pi}{N}k\right) = \ldots + \sum_{n=-N}^{-1} x(n)\, e^{-j2\pi nk/N} + \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi nk/N} + \sum_{n=N}^{2N-1} x(n)\, e^{-j2\pi nk/N} + \ldots$$

If we change the index in the inner summation from n to n - lN and interchange the order of the summation, we obtain :

$$X\left(\frac{2\pi}{N}k\right) = \sum_{n=0}^{N-1} \left[\sum_{l=-\infty}^{\infty} x(n-lN)\right] e^{-j2\pi nk/N} \qquad \textbf{(2-3)}$$

for k = 0,1,2, ..., N-1. The signal $x_p = \sum_{l=-\infty}^{\infty} x(n-lN)$ obtained by a periodic repetition of x(n) every N samples, is obviously periodic with fundamental period N. Consequently, it can be expanded in a Fourier series as

$$x_p(n) = \sum_{k=0}^{N-1} c_k e^{j2\pi nk/N} \qquad n = 0,1,...,N-1 \qquad \textbf{(2-4)}$$

with Fourier coefficients

$$c_k = \frac{1}{N} \sum_{k=0}^{N-1} x_p(n) e^{-j2\pi nk/N} \qquad k = 0,1,...,N-1 \qquad \textbf{(2-5)}$$

Comparing (2-5) with (2-3), we conclude that

$$c_k = \frac{1}{N} X\left(\frac{2\pi}{N}k\right) \qquad k = 0,1,...,N-1 \qquad (2\text{-}6)$$

Therefore, $$x_p(n) = \frac{1}{N} \sum_{k=0}^{N-1} X\left(\frac{2\pi}{N}k\right) e^{j2\pi nk/N} \qquad n = 0,1,...,N-1 \qquad (2\text{-}7)$$

Equation (2-7) provides the reconstruction of the periodic signal $x_p(n)$ from the samples of the spectrum $X(\omega)$. Since $x_p(n)$ is a periodic extension of x(n), the

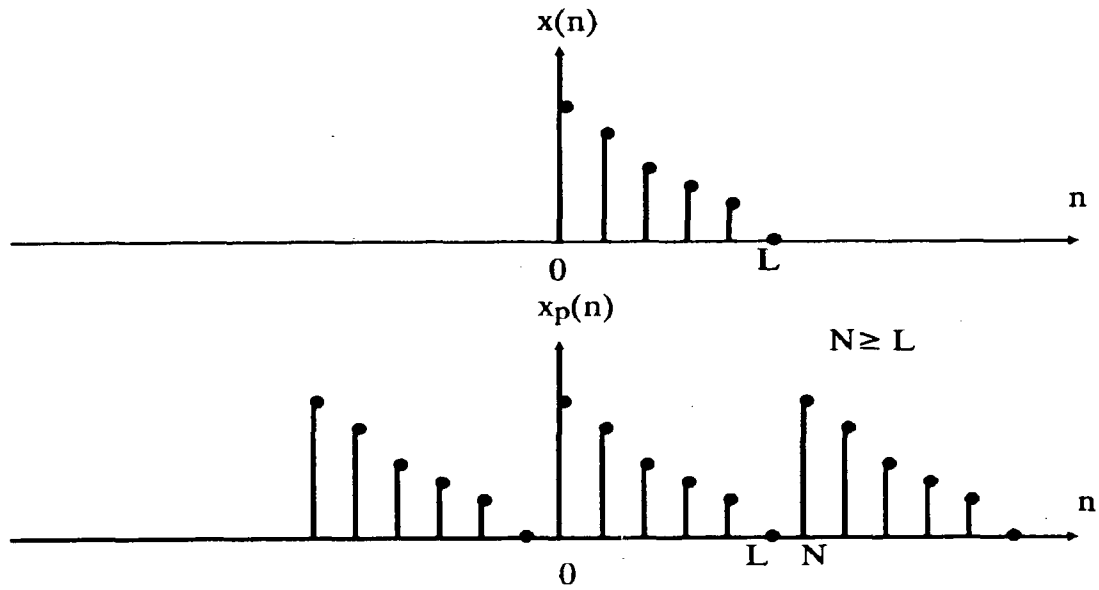


Fig. 2-1 Aperiodic sequence x(n) of length L and its periodic extension for N≥L

sequence x(n) can be recovered from $x_p(n)$ if there is no aliasing in the time domain, that is, if x(n) is time-limited to less than the period N of $x_p(n)$.

In summary, when a sequence x(n) has a finite duration of length $L \leq N$, then $x_p(n)$ is a periodic repetition of x(n), where $x_p(n)$ over a single period is given by :

$$x_p(n) = \begin{cases} x(n) & 0 \leq n \leq L-1 \\ 0 & L \leq n \leq N-1 \end{cases}$$

Consequently, the frequency samples $X(2\pi k/N)$, $k = 0,1,...,N-1$ uniquely represent the finite-duration sequence x(n). Hence the *discrete Fourier transform* of x(n):

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi nk/N} \qquad k=0,1,2,...,N-1 \qquad \textbf{(2-8)}$$

In turn, the sequence x(n) can be recovered from the frequency samples by the inverse discrete Fourier transform (IDFT) :

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k)\, e^{j2\pi nk/N} \qquad n=0,1,2,...,N-1 \qquad \textbf{(2-9)}$$

2.1.1 Symmetry property : The kernel of the DFT, which is $e^{-j2\pi i/N}$, consists of the roots of unity in the complex plane. If N is an even number, $e^{-j2\pi i/N}$ is the complex conjugate of $e^{-j2\pi(N-i)/N}$, and $e^{-j2\pi(N/2+i)/N} = -e^{-j2\pi i/N}$. Similar symmetry also exists when N is odd. This simple relationship is the principle from which many fast computational algorithms, including the FFT, are derived.

To illustrate the use of symmetry, let us consider a very simple algorithm. The frequency spectrum of x(n) can be evaluated from eqn (2-8) by varying the parameter k. It is obvious that the kernel is periodic and x(n) is also extended periodically, both with a period of N points. Hence there are only N distinct values for X(k) to be

calculated. Even for these N distinct X(k), we need to compute only half of them because of the symmetry mentioned above. If N is even, the first N/2 frequency components are the complex conjugates of the remaining components :

$$X\left(\frac{N}{2}\right) = -X(0)$$
$$X(i) = X^{*}(N-i) \qquad \text{for } i=1,2,...,N/2-1 \tag{2-10}$$

where * denotes the complex conjugate. The case for an odd N is similar and obvious.

2.2 Divide-and-Conquer Approach to Compute the DFT

This approach is based on the decomposition of an N-point DFT into smaller DFTs. Let N be factorized as a product of two integers, that is, $N = N_1N_2$. The sequence x(n), $0 \le n \le N-1$, which is a one-dimensional array, can now be stored as a two-dimensional array indexed by n_1 and n_2, where $0 \le n_1 \le N_1-1$ and $0 \le n_2 \le N_2-1$. Suppose we select the mapping $n = N_2n_1 + n_2$, we obtain an arrangement in which the first row consists of the first N_2 elements of x(n) and the second row consists of

n_1 \ n_2	0	1	2	...	N_2 - 1
0	x(0)	x(1)	x(2)	...	x(N_2-1)
1	x(N_2)	x(N_2+1)	x(N_2+2)	...	x(2N_2-1)
2	x(2N_2)	x(2N_2+1)	x(2N_2+2)	...	x(3N_2-1)
	⋮	⋮	⋮	...	⋮
N_1-1	x([L-1]N_2)	x([L−1]N_2+1)	x([L−1]N_2+2)	...	x(N_1N_2-1)

Fig 2-2 Two dimensional data array for storing x(n) with $n = N_2n_1 + n_2$

the next N_2 elements of x(n) and so on, as illustrated in Fig. 2-2. Another possible mapping is, of course, $n = n_2 + N_1 n_2$, which is a column-wise mapping.

A similar arrangement can be used to store the computed DFT values. Let the mapping be from the index k to a pair of indices k_1 and k_2 , where $0 \le k_1 \le N_1 - 1$ and $0 \le k_2 \le N_2 - 1$. The row-wise mapping is given by $k = N_2 k_1 + k_2$.

Now that x(n) is mapped into a rectangular array $x(n_1,n_2)$ and X(k) is mapped into a corresponding array $X(k_1,k_2)$. Then the DFT can be expressed as a double sum over the elements of the rectangular array multiplied by suitable twiddle factors. To be specific, let x(n) be mapped row-wise and the DFT mapped column-wise; then

$$X(k_1,k_2) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x(n_1,n_2) W_N^{(N_2k_1+k_2)(n_1+N_1n_2)} \qquad \textbf{(2-11)}$$

where $W_N = e^{-j2\pi/N}$. But $W_N^{(N_2k_1+k_2)(n_1+N_1n_2)} = W_N^{N_2N_1k_1n_2}\ W_N^{N_1n_2k_2}\ W_N^{N_2k_1n_1}\ W_N^{n_1k_2}$, $W_N^{N_2N_1k_1n_2} = 1$ as $N_2N_1 = N$, $W_N^{N_2k_1n_1} = W_{N/N_2}^{k_1n_1} = W_{N_1}^{k_1n_1}$, and $W_N^{N_1k_2n_2} = W_{N/N_1}^{k_2n_2} = W_{N_2}^{k_2n_2}$. With these simplifications, eqn (2-11) may be expressed as :

$$X(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \left\{ W_N^{n_1k_2} \left[\sum_{n_2=0}^{N_2-1} x(n_1,n_2) W_{N_2}^{n_2k_2} \right] \right\} W_{N_1}^{n_1k_1} \qquad \textbf{(2-12)}$$

The computation of eqn(2-11) can be carried out in three steps :

First, compute the N_2-point DFTs

$$F(n_1,k_2) = \sum_{n_2=0}^{N_2-1} x(n_1,n_2) W_{N_2}^{n_2k_2} \qquad 0 \le k_2 \le N_2 - 1 \qquad \textbf{(2-13)}$$

for each of the rows $n_1 = 0,1, ..., N_1\text{-}1$.

Second, compute a rectangular array $G(n_1,k_2)$

$$G(n_1,k_2) = W_N^{n_1k_2} F(n_1,k_2) \qquad \begin{matrix} 0 \le n_1 \le N_1-1 \\ 0 \le k_2 \le N_2-1 \end{matrix} \qquad \textbf{(2-14)}$$

Finally, compute the N_1-point DFTs

$$X(k_1,k_2) = \sum_{n_1=0}^{N_1-1} G(n_1,k_2) W_{N_1}^{n_1k_1} \qquad \text{for } k_2 = 0,1,\ldots,N_2-1 \qquad \textbf{(2-15)}$$

Apparently the above procedure looks more complex than directly computing the DFT. However, let us evaluate its computational complexity. The first step requires $N_1N_2^{\,2}$ complex multiplications and $N_1N_2(N_2\text{-}1)$ complex additions. The second step requires N_1N_2 complex multiplications. Finally the third step requires $N_2N_1^{\,2}$ complex multiplications and $N_2N_1(N_1\text{-}1)$ complex additions. Recalling that $N = N_1N_2$, the computational complexity of the whole process is therefore $N(N_1+N_2+1)$ complex multiplications and $N(N_1+N_2-2)$ complex additions. If

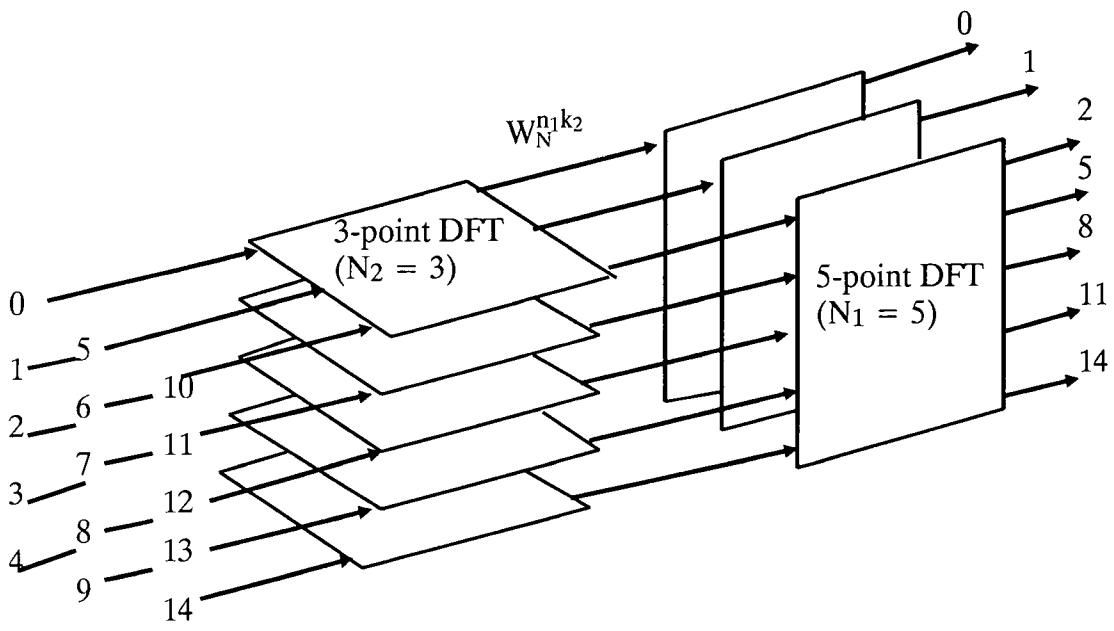


Fig. 2-3 Computation of DFT with N = 15 by 3-point and 5-point DFTs

$N_1 \approx N_2$, then N_1 or $N_2 \approx \sqrt{N}$, and the complexity for both multiplications and additions is approximately $2N\sqrt{N}$. Comparing to the direct computation which requires N^2 complex multiplications and N(N-1) complex additions, the divide-and-conquer approach can reduce the complexity.

When N is a highly composite number, i.e. $N = r_1 r_2 \ldots r_u$, then the decomposition can be repeated u-1 times, which means that smaller DFTs are formed and a more efficient algorithm is available. If the factors (r's) are mutually prime to each other, we have the prime factor algorithm. When all the r's are equal, we have $N = r^u$; then all the DFTs are of size r. This number r is called the radix of the computational algorithm. In particular when r = 2, we obtain the Fast Fourier Transform. Proakis and Manolakis [10] provide a detailed discussion on this topic.

2.3 The Fast Fourier Transform

By far, the radix-2 algorithms are the most widely used FFT algorithms. When the sequence length N is a power of 2, i.e. $N = 2^u$, we can apply the divide-and-conquer approach described above successively to form finally DFTs of length 2. At first, let M = N/2 and L = 2. This selection splits the whole data sequence into two N/2-point data sequences, $f_1(n)$ and $f_2(n)$, corresponding to the even-numbered and odd-numbered samples of x(n) respectively :

$$f_1(n) = x(2n)$$
$$f_2(n) = x(2n+1) \qquad n = 0,1,\ldots,\frac{N}{2}-1$$

As $f_1(n)$ and $f_{2(n)}$ are obtained by decimating x(n) by a factor of 2, the resulting FFT algorithm is called the *decimation-in-time* algorithm. Now the N-point DFT can be expressed as follows :

$$X(k) = \sum_{m=0}^{N/2-1} x(2m)\, W_N^{2mk} + \sum_{m=0}^{N/2-1} x(2m+1)\, W_N^{k(2m+1)} \qquad \textbf{(2-16)}$$

But $W_N^2 = W_{N/2}$, eqn (2-16) can be expressed as

$$X(k) = \sum_{m=0}^{N/2-1} f_1(m)\, W_{N/2}^{km} + W_N^k \sum_{m=0}^{N/2-1} f_2(m)\, W_{N/2}^{km}$$

$$= F_1(k) + W_N^k F_2(k) \qquad k = 0,1,\ldots,\frac{N}{2}-1 \qquad \textbf{(2-17)}$$

where $F_1(k)$ and $F_2(k)$ are the N/2 -point DFTs of the sequence $f_1(m)$ and $f_2(m)$ respectively. In addition, $F_1(k)$ and $F_2(k)$ are periodic with period N/2 and $W_N^{k+N/2} = -W_N^k$, which is the symmetry property discussed in section 2.1.1. Hence eqn (2-17) can be written as

$$X(k) = F_1(k) + W_N^k F_2(k) \qquad k = 0,1,\ldots,\frac{N}{2}-1$$

$$X\left(k+\frac{N}{2}\right) = F_1(k) - W_N^k F_2(k) \qquad k = 0,1,\ldots,\frac{N}{2}-1 \qquad \textbf{(2-18)}$$

W_N^k in eqn (2-18) is the twiddle factor.

Note that the direct computation of $F_1(k)$ requires $(N/2)^2$ complex multiplications. The same is true for $F_2(k)$. Furthermore, there are N/2 additional complex multiplications required to compute $W_N^k F_2(k)$. Hence the total number of complex multiplications is $N^2/2 + N/2$. Thus the first step of decimation reduces the number of multiplications from N^2 to $N^2/2 + N/2$, which is nearly half the original number for a large N.

Having performed the decimation-in-time once, we can repeat the process for each of the sequences $f_1(n)$ and $f_2(n)$, which will generate in total 4 sequences of N/4 points each. If the decimation process is repeated successively, in the last stage there

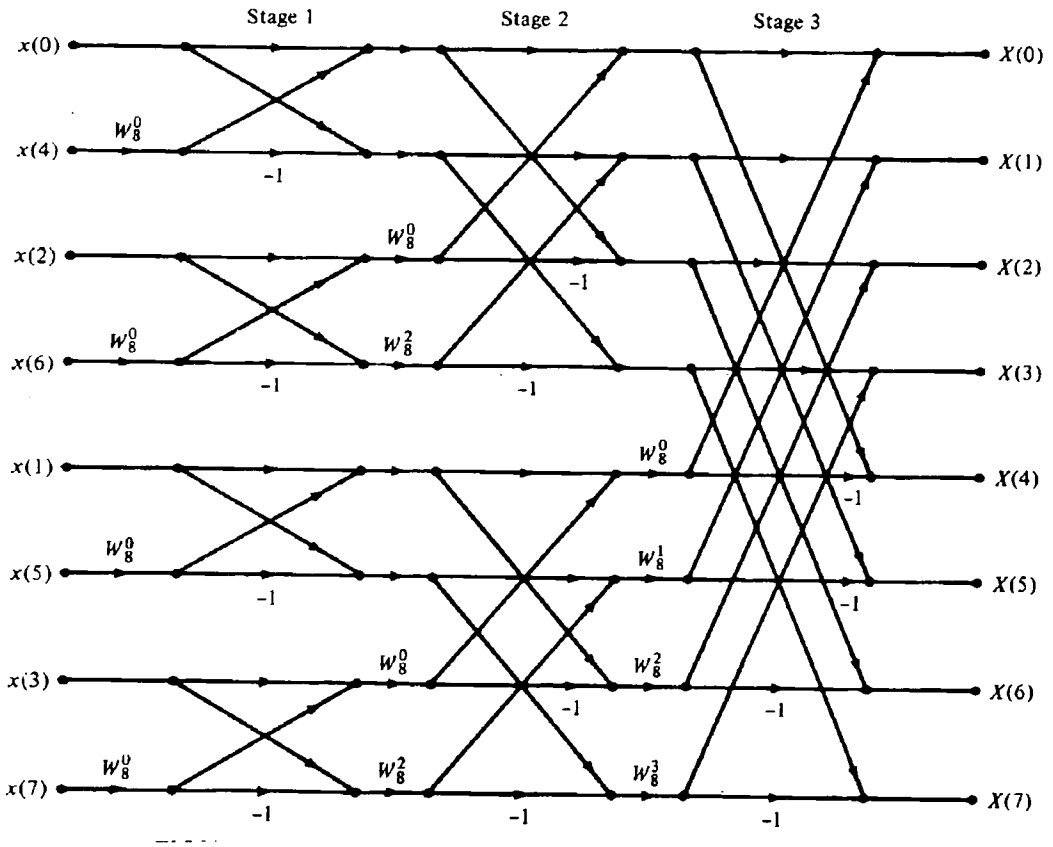


Fig . 2-4 Eight-point decimation-in-time FFT algorithm

will be N/2 2-point DFTs to be computed. Each of these 2-point DFTs is called a "butterfly" in the signal flow diagram of the algorithm. Fig. 2-4 shows the signal flow diagram for an eight- point decimation-in-time FFT algorithm, in which the computation is done *in place*, that is, the same 2N storage locations are used throughout the computation of the N-point DFT, and the output data sequence is *in order*, that is, in the normal order.

From Fig. 2-4, we can estimate the number of complex multiplications required. For a sequence of length $N = 2^u$, the decimation process is repeated $u = \log_2 N$ times until the resulting sequences are reduced to one point each. For each stage there are N/2 multiplications of the twiddle factors to be performed. Hence the

number of complex multiplications is reduced to (N/2) $\log_2 N$. The number of complex additions can be shown to be N $\log_2 N$.

Another radix-2 FFT algorithm, called the decimation-in-frequency algorithm, can be obtained by the divide-and-conquer approach with $N_2 = 2$ and $N_1 = N/2$ in the first step of decimation. Both the decimation-in-time and the decimation-in-frequency algorithms need to shuffle the input data sequence in a bit-reversed order so that the output data sequence can emerge in order if the computation is to be done in place. Details of the decimation-in-frequency algorithm and data shuffling are well documented in many reference books [1,10].

2.4 The Prime Factor Algorithm

The aim of the divide-and-conquer approach described in section 2.2 is to change a one-dimensional transform into a multi-dimensional transform so that the computational complexity is reduced. The mapping leading to the derivation of eqn(2-12) imposes no condition on the two transform lengths N_1 and N_2, but twiddle factors $W_N^{n_1 k_2}$, which requires complex multiplications, are created between the two different dimensions. Good [11] and Winograd [12], however, used a mapping that requires N_1 and N_2 be relatively prime to each other. By algebraic manipulations according to the number theory, each dimension is uncoupled to the others and the undesirable twiddle factors can thus be eliminated.

2.4.1 Address Mapping : Let $N = N_1 N_2$ where N_1 and N_2 are relatively prime to each other. The indices may be mapped as :

$$n = An_1 + Bn_2 \qquad 0 \le n_1 \le N_1 - 1 \text{ and } 0 \le n_2 \le N_2 - 1$$

$$k = Ck_1 + Dk_2 \qquad 0 \le k_1 \le N_1 - 1 \text{ and } 0 \le k_2 \le N_2 - 1$$

then the transform becomes :

$$X(k_1,k_2) = \sum_{n2=0}^{N_2-1}\sum_{n1=0}^{N_1-1} x(n_1,n_2) W_N^{(An_1+Bn_2)(Ck_1+Dk_2)}$$
$$= \sum_{n2=0}^{N_2-1}\sum_{n1=0}^{N_1-1} x(n_1,n_2) W_N^{ACn_1k_1} W_N^{ADn_1k_2} W_N^{BCn_2k_1} W_N^{BDn_2k_2} \qquad \textbf{(2-19)}$$

Note that $W_N^{mN} \equiv 1$ and $W_N^{mN+i} = W_N^i$, where m and i are integers. The index i forms an arithmetic structure modulo N . To make eqn (2-19) a real two-dimensional transform, we want

$$AC \equiv 1,\ AD \equiv 0,\ BC \equiv 0 \text{ and } BD \equiv 1 \qquad \textbf{(2-20)}$$

There are, of course, many possible solutions to satisfy the above conditions. One possible mapping for n is that $n = N_2 n_1 + N_1 n_2$, giving $A = N_2$ and $B = N_1$. Because N_1 and N_2 are relatively prime to each other, this mapping must be one-to-one. Substituting $A = N_2$ and $B = N_1$ into eqn(2-19):

$$X(k_1,k_2) = \sum_{n2=0}^{N_2-1}\sum_{n1=0}^{N_1-1} x(n_1,n_2) W_N^{N_2Cn_1k_1} W_N^{N_2Dn_1k_2} W_N^{N_1Cn_2k_1} W_N^{N_1Dn_2k_2}$$
$$= \sum_{n2=0}^{N_2-1}\sum_{n1=0}^{N_1-1} x(n_1,n_2) W_{N_1}^{Cn_1k_1} W_N^{N_2Dn_1k_2} W_N^{N_1Cn_2k_1} W_{N_2}^{Dn_2k_2} \qquad \textbf{(2-21)}$$

Now the index of the first exponential term, W_{N1}, is modulo N_1, and the last, W_{N2} is modulo N_2. In order to satisfy eqn (2-20), we can deduce, from the exponential terms of eqn (2-21), that $C = N_2 \langle N_2^{-1} \rangle_{N_1}$ and $D = N_1 \langle N_1^{-1} \rangle_{N_2}$, where $\langle x \rangle_{Nr}$ denotes the residue of x modulo Nr and x^{-1} its multiplicative inverse . Substituting these choices into eqn (2-21) and omitting the $\langle\ \rangle_{Nr}$ for simplicity :

$$X(k_1,k_2) = \sum_{n2=0}^{N_2-1}\sum_{n1=0}^{N_1-1} x(n_1,n_2) W_{N_1}^{N_2N_2^{-1}n_1k_1} W_N^{N_2N_1N_1^{-1}n_1k_2} W_N^{N_1N_2N_2^{-1}n_2k_1} W_{N_2}^{N_1N_1^{-1}n_2k_2}$$

$$= \sum_{n2=0}^{N_2-1} \sum_{n1=0}^{N_1-1} x(n_1,n_2) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2} \qquad \textbf{(2-22)}$$

as $<N_2 N_2^{-1}>_{N_1} \equiv 1$, $<N_1 N_1^{-1}>_{N_2} \equiv 1$ and $N_1 N_2 = N$ making the second and the third exponential terms equal to 1. Hence the mapping for the indices :

$$n = N_2 n_1 + N_1 n_2 \qquad \textbf{(2-23)}$$

$$k = N_2 <N_2^{-1}>_{N_1} k_1 + N_1 <N_1^{-1}>_{N_2} k_2 \qquad \textbf{(2-24)}$$

where $0 \le n_1 \le N_1 - 1$, $0 \le n_2 \le N_2 - 1$, $0 \le k_1 \le N_1 - 1$ and $0 \le k_2 \le N_2 - 1$

The order to evaluate the two summations in the transform expressed by eqn (2-22) is immaterial, i.e. we can evaluate either n_1 or n_2 first. Conceptually the transform is performed over a two-dimensional array, but in a single-processor system, the elements belonging to a row or column are usually taken from the memory when required for computation, after which the results are put back into the respective memory locations. Fig. 2-5 illustrates a particular column being *loaded* into the processor for computation and *retrieved* after computation. Suppose we choose to evaluate n_1 first, then the procedure becomes :

(1) Take x(n) according to eqn (2-23) one column at a time, i.e. for each n_2, $n_1 = 0,1,...,N\text{-}1$.

(2) For each of the columns $n_2 = 0,1,..., N_2\text{-}1$, compute the N_1-point DFTs

$$F(k_1,n_2) = \sum_{n1=0}^{N_1-1} x(n_1,n_2) W_{N_1}^{n_1 k_1} \qquad k_1 = 0,1,...N_1-1$$

The results are stored in-place, i.e. in the corresponding locations of $x(n_1,n_2)$.
(3) Take $F(k_1,n_2)$ row by row; for each k_1 compute

$$X(k_1,k_2) = \sum_{n2=0}^{N_2-1} F(k_1,n_2)W_{N_2}^{n_2k_2} \qquad k_2 = 0,1,...,N_2-1$$

(4) Re-order or unscramble the results according to eqn (2-24).

To give a simple numerical example, let $N = 15 = 3x5$. Then $N_1 = 3$, $N_2 = 5$ and $n = 5\,n_1 + 3\,n_2$, where $n_1 = 0,1,2$ and $n_2 = 0,1,2,3,4$. With this mapping for n, we obtain conceptually a 3 x 5 data array similar to the one shown in Fig. 2-2. After computing the inverses, we obtain $<5^{-1}>_3 = 2$ and $<3^{-1}>_5 = 2$; so the mapping for $k = 10k_1 + 6\,k_2$, which specifies the indices for unscrambling the resulting sequence.

2.4.2 Computational complexity : When a one-dimensional array of length N is arranged as a two-dimensional array of N_1xN_2, there are N_1 DFTs of length N_2 and N_2 DFTs of length N_1 to be computed. Assuming direct DFT calculation, there should be $N_1N_2^{\,2} + N_2N_1^{\,2} = N(N_1+N_2)$ complex multiplications and $N(N_1+N_2-2)$ complex additions. Referring to the 3x5 example, the original number of multiplications is $(15)^2 = 225$, but with the prime factor algorithm, the number becomes $15(3+5) = 120$, which is a reduction by nearly a factor of 2. In fact, efficient algorithms to compute short DFTs of length 2,3,4,5, 7,8,9 and 16 are available [13]. Moreover, when the length of the DFT is an odd prime, the DFT can also be evaluated in high speed by a convolution [8]. One may even utilize fast hardware in the form of a recursive filter to compute the short DFTs [14]. If $\mu(N_r)$ is the complexity of a particular algorithm chosen for a short DFT of length-N_r, then the computational complexity for multiplication for two stages can be written as $N_1\mu(N_2) + N_2\mu(N_1)$.

2.4.3 In-place, In-order Algorithm : The mapping given by eqn (2-23) and (2-24) leads only to an in-place algorithm since unscrambling is required after the last stage of

computation. By modifying slightly the mapping given by eqn (2-24), an in-place, in-order algorithm can be achieved [15]. The procedure is as follows :

(1) Load from the memory array column by column according to $n = N_2n_1 + N_1n_2$.

(2) Perform the length-N_1 DFTs along each column. Place the intermediate results $F(k_1,n_2)$ into the memory according to the mapping $k' = N_2N_2^{-1}k_1 + N_1n_2$.

(3) Load from the memory row by row, also according to $n = N_2n_1 + N_1n_2$.

(4) Perform the length-N_2 DFTs along each row . Place the results into the memory according to the mapping $k = N_2k_1 + N_1N_1^{-1}k_2$.

By examining the above procedure, we see that the mapping specified by eqn (2-24) is implemented in two phases; first for k_1 then for k_2. Results are unscrambled after every short DFT instead of at the end of the whole transform, but the final

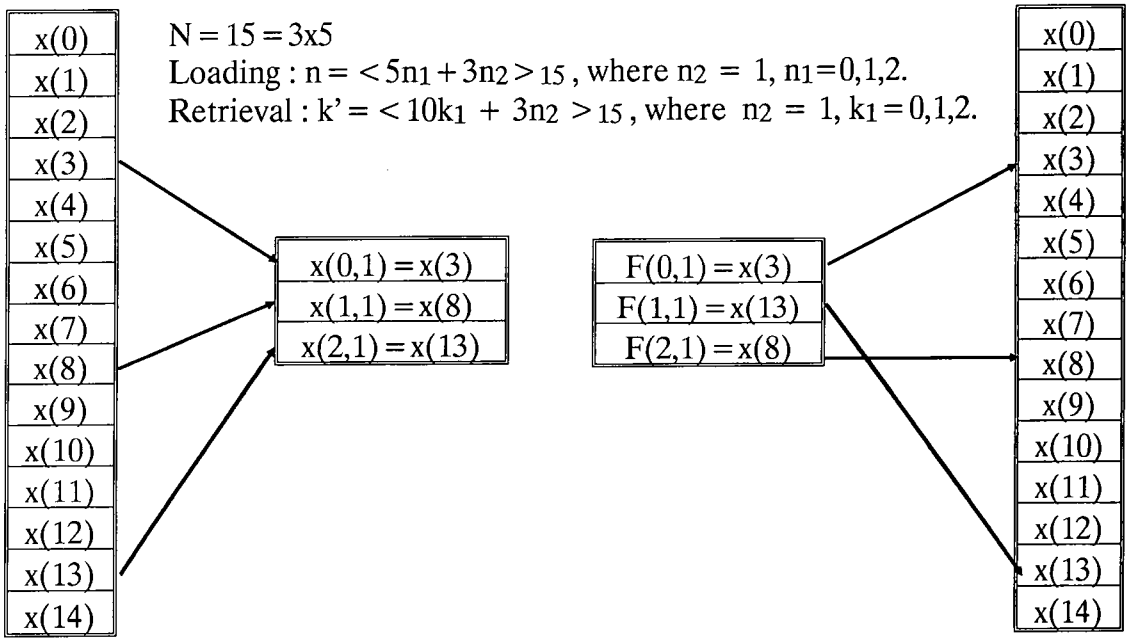


Fig. 2-5 Memory loading and retrieval for an in-place, in-order prime factor algorithm

outcomes are the same for both cases. Each time when an unscrambling operation is done, only a permutation of data within the same row or column is effected, which is an in-place process. Fig. 2-5 shows the loading and retrieval of the second column ($n_2 = 1$) of a 3x5 example. The permutation is such that the elements of a particular row or column will always stay together in the same row or column. Thus the permutation does not affect the results of the short DFTs but only their addresses. For a particular algorithm, all the addresses required for the transform can be calculated in advance; therefore the generation of addresses should not affect the computation time at all.

To find the multiplicative inverse of an integer in a finite field, the solution of a diophantine equation is involved. One may refer to books about number theory [16].

2.4.4 Multi-dimensional transform : To gain the full advantage of the prime factor algorithm, small integers are desired to be the factors making up the product of the sequence length, which enables the use of efficient short DFT algorithms as discussed in section 2.4.2. If only a two-dimensional algorithm is used, i.e. $N = N_1.N_2$, the resulting sequence length could be too small to be useful when both N_1 and N_2 are small integers. One solution is that we select a highly composite sequence length, i.e. $N = N_1.N_2....N_i$, where the N_i's are mutually prime to each others. Then more choices are available for the sequence length and the two-dimensional prime factor algorithm can be applied recursively as well.

Let $N = N_1.N_2....N_i$, where the N_i's are mutually prime to each others. Define a mapping $n = <N_r n_1 + N_1 n_r >_N$, where $N_r = N_2.N_3...N_i = N/N_1$, $n = 0,1,...,N-1$, $n_1 = 0,1,...,N_1-1$ and $n_r = 0,1,2,...,N_r-1$. As N_1 is prime to N_r by definition, the above

mapping forms a two-dimensional prime factor algorithm. We may first compute the length-N_1 DFTs , then followed by the length-N_r DFTs. When computing the length-N_r DFTs, we can apply the prime factor algorithm here again by defining another mapping $n' = <N_r'n_2 + N_2n_r' >_N$, where $Nr' = N_3.N_4...N_i = Nr/N_2$, $n' = 0,1,...,Nr-1$, $n_2 = 0,1,...,N_2-1$ and $n_r' = 0,1,2,...,N_r'-1$. It is obvious that this decomposition of data can be continued till the very last factor of N_i is reached. A multi-dimensional form of prime factor algorithm can thus be written as [17]:

$$X(k_i,...,k_2,k_1) = \sum_{n_i=0}^{N_i-1} ... \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x(n_i,...,n_2,n_1)\ W_1^{n_1k_1} W_2^{n_2k_2} ... W_i^{n_ik_i}$$

and in the r-th dimension, the addresses for loading and retrieval of data can be given by [18]:

$$n = <Nrn' + Rn_r>_N$$

$$k = <Nrn' + RR^{-1}n_r>_N \qquad \textbf{(2-25)}$$

for $n' = 0,1,...,R-1$, $n_r = 0,1,...,N_r-1$ where $R = N/N_r$ for the r-th dimension, and $<RR^{-1}>_{N_r} = 1$. In the retrieval equation, n_r is used instead of k_r for simplicity.

Let us take the sequence length $N = 3x5x7 = 105$ as an example of a three-dimensional algorithm. Assuming we evaluate length-3 DFTs first, the addresses for loading and retrieval are given respectively by :

$$n = <3n' + 35n_r>_{105}, \qquad n'=0,1,...,34, \quad n_r=0,1,2$$

$$k = <3n' + 70n_r>_{105}, \qquad n'=0,1,...,34, \quad n_r=0,1,2$$

Following the concept of a two-dimensional array, after each of the 35 length-3 DFTs is computed, the results are permuted and placed in the respective columns of the array. To complete the computation, we have to perform 3 length-35 DFTs along the rows of the array, the indices of which are given by

$$n = <35n' + 3n_r>_{105} \qquad n'=0,1,2,\ n_r=0,1,...,34$$

Each of these rows can also be treated as a 5 by 7 array by the same method. Since each row is handled independently, its indices may be mapped again from 0 to 34 in order to simplify the addressing scheme in the calculation to follow. Performing the length-5 DFTs along the sub-columns, the mapping is :

$$n = <5n' + 7n_r>_{35}, \qquad n'=0,1,...,6, \quad n_r=0,1,...,4$$

$$k = <5n' + 21n_r>_{35}, \qquad n'=0,1,...,6, \quad n_r=0,1,...,4 \qquad \textbf{(2-26)}$$

and followed by the length-7 DFTs along the sub-rows with

$$n = <7n' + 5n_r>_{35}, \qquad n'=0,1,...,4, \quad n_r=0,1,...,6$$

$$k = <7n' + 15n_r>_{35}, \qquad n'=0,1,...,4, \quad n_r=0,1,...,6 \qquad \textbf{(2-27)}$$

In eqn (2-26) and (2-27), $n = 0$ or $k = 0$ refers to the first element of the row. Finally, the results in each row must be permuted according to

$$k = <35n' + 36n_r>_{105}, \qquad n'=0,1,2, \quad n_r=0,1,...,34$$

before they are in order.

2.5 Realization of the PFA in a Transputer Network

The prime factor algorithm converts a one-dimensional DFT into a multi-dimensional DFT. When the algorithm is implemented in a general purpose computer with a single processor, different dimensions must be computed one after another. If special hardware or supercomputers are available, it is possible to share the computation by several processors performing simultaneously in a suitable configuration. Hypercube and pipeline architectures were suggested and studied by G. Aloisio et al [19]. Basically the hypercube is a structure that maps naturally into the Cooley-Tukey algorithm rather than the PFA; hence it is not surprising to see that the Cooley-Tukey algorithm excels the PFA in several aspects when implemented in

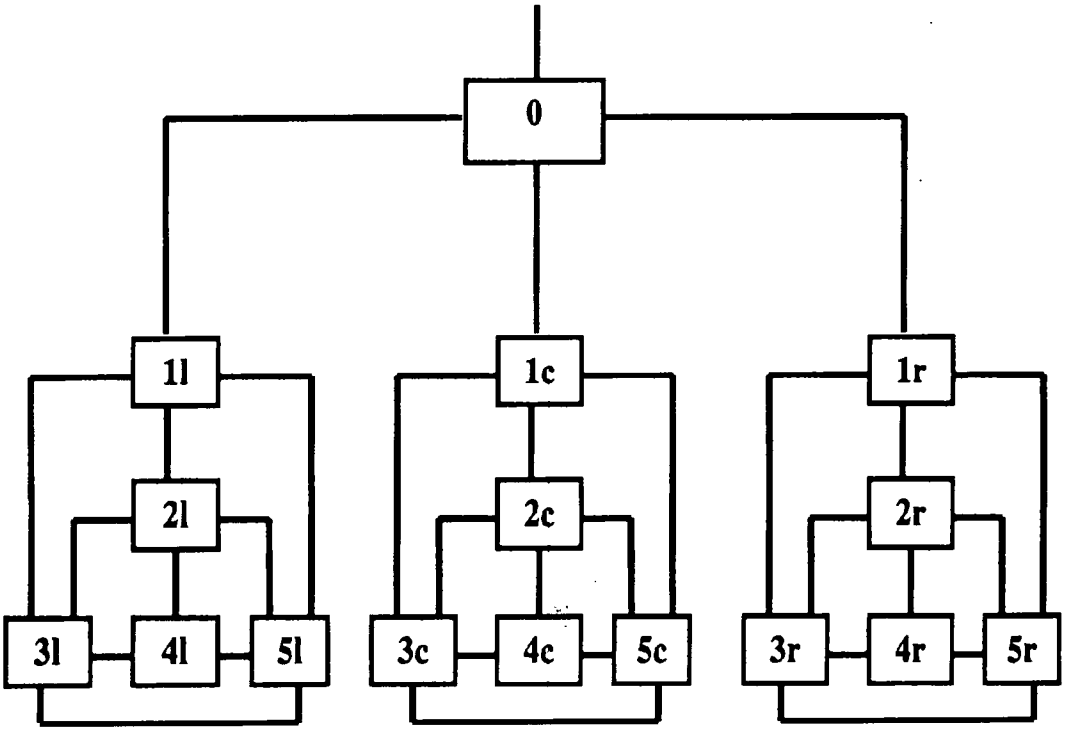


Fig. 2-6 The tree network of transputers (adopted from the configuration of a transputer system in the University of Durham)

a hypercube, especially the communication time required. There are, however, some other networks that fit the PFA better. A particular example is the tree network shown in Fig. 2-6 [20], which is suitable for calculating DFT of a sequence length of N = 3x5x7x11 = 1155 with a minimum communication time for exchanging partial results between processors.

2.5.1 The Overall Scheme In Fig. 2-6, each box represents a transputer and each line represents a two-way communication channel. The tree network is formed by connecting the communication channels of the transputers together. Processor 0 is the root transputer which is responsible for communicating with the host computer system. Processors 1l, 1c and 1r are called the branches with l,c and r denoting left, central and right respectively. Processors 2 to 5 are called the leaves. When the data are fed into the root processor, they are rearranged into sets of three according to the

addresses of dimension 3. These sets are sent in parallel to the three branch processors, each of which calculates one of the three transform components, i.e. processor 1l produces X(0) while processors 1c and 1r calculates X(2) and X(1) respectively. With this systolic-like operation, the partial results are naturally divided into three groups coming out from the processors and the subsequent operations for these three groups are identical, which is to perform the three-dimensional DFT of 5x7x11. This arrangement requires no exchange of partial results between the branches.

Underneath each of the branch processors, there are another 4 leaf processors linked together by communication channels. Hence there are three groups of 5 processor each. Since the operations in these three groups are identical, we may concentrate our discussion on the left group. There are 385 data points to be calculated by processor 1 for dimension 3. While performing the calculation, the partial results can be distributed to the leaf processor according to the requirement of dimension 5. Since there are 77 sets of data of 5 elements each to be distributed among 5 processors, the load will be uneven. Following the loading is the procedure for performing length-5 DFTs. From this point onwards, each set of data points will be contained and handled by the same processor, so efficient algorithms, like the short DFT algorithms, can be applied.

2.5.2 Addresses in multi-processor system If the PFA is realized by a single-processor system, the addresses of the data expressed as their order in a linear array are sufficient to keep track of all data loading and retrieval. In case that several processors are computing concurrently different parts of the transform, the processors must know the order of the data in that particular dimension and their addresses in the following

dimension where the partial products are to be sent after this stage of computation. The order of the data in the form of a linear array is essential only at the first loading and the final retrieval process. The addressing problem of the PFA has been a research topic for years. When the PFA is implemented in a multi-processor system, the generation of addresses becomes more complicated. In our approach, the indices of data at all dimensions are kept in an appropriate order as their addresses [20].

Since exchanges of partial products among the processors should be kept as few as possible, the in-place in-order algorithm expressed by eqn(2-25), where a length-N sequence is converted to an N_r x R two-dimensional array, will be adopted. As discussed previously, the algorithm can be extended to a multi-dimensional transform by applying the method recursively. For example, if $N = N_1.N_2.N_3$, we may compute dimension N_1 first followed by N_2 and N_3. The loading equation will be :

$$n = < N_1 n_{23} + N_{23} n_1 >_N , \qquad \textbf{(2-28)}$$

where $n_1 = 0,1,2,..., N_1-1$ and $n_{23} = 0,1,2,...,N_2N_3-1$. Having finished with dimension N_1, we can computed dimension N_2 and N_3. Keeping the original sequence order, the loading equations for dimension N_2 and N_3 are given respectively by :

$$n = < N_{23} n_1 + N_1 < N_2 n_3 + N_3 n_2 >_{N_2} >_N \qquad \textbf{(2-29)}$$

and

$$n = < N_{23} n_1 + N_1 < N_3 n_2 + N_2 n_3 >_{N_3} >_N \qquad \textbf{(2-30)}$$

where $n_2 = 0,1,2,..., N_2 -1$ and $n_3 = 0,1,2,..., N_3 -1$. Hence the address of a data point in a three-dimensional PFA can be determined by three indices, i.e. n_1,n_2 and n_3. Let us call these indices the class number, group number and sequel number of a data point. Eqn (2-28) divides the data of length N into N_1 classes of N_2 groups and in each group, there are N_3 data points. Rewriting eqn (2-28) and (2-29), we obtain :

$$n = < N_{23} c + N_1 < N_2 g_2 + N_3 s_2 >_{N_2} >_N \qquad \textbf{(2-31)}$$

and $$n = < N_{23}\, c + N_1 < N_3\, g_3 + N_2\, s_3 >_{N_3} >_N \quad \textbf{(2-32)}$$

where c is the class number, and g_x and s_x are the group and sequel number respectively in dimension x. After the computation in dimension 1, we must determine the sets of numbers associated with the data in the next dimension. Hence the relationship between these numbers must be studied. The following theorems about their relationship are useful in determining the routing of the partial products from one processor to another.

Theorem 1 : In a two-dimensional PFA, the group number and sequel number of a data in the first dimension become its sequel number and group number respectively in the second dimension, i.e. $g_1 \equiv s_2$ and $s_1 \equiv g_2$.

Proof : Refer to $n = < N_1\, n_2 + N_2\, n_1 >_N$, a two-dimensional loading equation. For dimension N_1, n_2 is g_1 and n_1 is s_1 by definition. When dealing with the second dimension N_2, n_1 becomes g_2 and n_2 becomes s_2. Hence $g_1 \equiv s_2$ and $s_1 \equiv g_2$.

Theorem 2 : In a three-dimensional PFA, the sequel number in the first dimension is the class number in the second and third dimensions.

Proof : In eqn (2-28) n_1 is the sequel number of dimension 1. In eqn (2-29) and (2-30), the same index n_1 becomes the class number of dimensions 2 and 3.

Theorem 3 : The starting address of a class differs from its neighbouring class by $< N_2 N_3 >_N$ if loading is performed according to eqn (2-31) or (2-32).

Proof : In eqn (2-31) or (2-32), the starting address of each class is given by $< c N_{23} >_N$ since $g = 0$ and $s = 0$. As $c = 0,1,2,...,N_1\text{-}1$, the first elements of the classes are found at locations $< 0 >_N$, $< N_{23} >_N$, $< 2\, N_{23} >_N$, ... , $< (N_1 - 1)\, N_{23} >_N$. Therefore

the starting addresses between two neighbouring classes differ by $<N_{23}>_N$ $= <N_2N_3>_N$.

Theorem 4 : The group and sequel numbers in each class of the second and third dimensions can be determined from the group numbers of the first dimension. For dimension 2, $g_2 = <N_2^{-1} g_1>_{N_3}$, and $s_2 = <N_3^{-1} g_1>_{N_2}$. It follows from Theorem 1 that $g_3 \equiv s_2$ and $s_3 \equiv g_2$.

Proof : From eqn (2-28) and (2-31), $n_{23} \equiv <g_1>_{N_{23}} \equiv < N_2 g_2 + N_3 s_1>_{N_{23}}$. Taking modulo N_3 on the above equation, we obtain $g_2 = <N_2^{-1} g_1>_{N_3}$ where N_2^{-1} is the inverse of $<N_2>_{N_3}$. Similarly, taking modulo N_2, we obtain $s_2 = <N_3^{-1} g_1>_{N_2}$.

Theorem 5 : If $n = < N_1 n_2 + N_2 n_1>_N$ is the loading equation, the retrieval addresses of a group can be obtained by updating the sequel numbers of the elements as follows : $s_{1(i)}=s_{1(i-1)} + <N_2>_{N_1}$, where $s_{1(i)}$ is the i-th sequel number in dimension 1 and $i = 0,1,...,N_1-1$, i.e. modulo N_1.

Proof : The retrieval addresses are given by $k = < N_1 n_2 + N_2^{-1}N_2 n_1>_N$. It is obvious that the changes in addresses are effected by the second term of the equation. To express the retrieval addresses in terms of the loading addresses, we make $k = n$ and obtain

$$< N_2^{-1}N_2 n_1>_N \equiv <N_2 n'_1>_N , \qquad \textbf{(2-33)}$$

where n'_1 represents the loading indices. Since both n_1 and $n'_1 = 0,1,...N_1-1$, eqn (2-23) is modulo N_1. Hence $n_1 \equiv <N_2 n'_1>_{N_1}$ and adopting the i notation defined above, we obtain $n_{1(i)} \equiv n_{1(i-1)} + <N_2>_{N_1}$ as n'_1 increments by 1. Since n_1 is the sequel number in dimension 1, we have proved the theorem.

2.5.2.1 Example :The use of the above theorems in calculating the addresses of the data is illustrated here by a numerical example. Let N = 3 x 5 x 7 = 105. The first dimension is the length-3 DFTs, the second is length-5 and the third is length-7. The loading equation is $<3g_1 + 35s_1>_{105}$, where g_1 and s_1 are the group and sequel number respectively in the first dimension. Since processor 1c computes X(2) and processor 1r computes X(1), the retrieval equation $<3g_1 + 70s_1>_{105}$ is already realized if we keep $s_1 = 1$ to processor 1c and $s_1 = 2$ to processor 1r. After computing length-3 DFTs, the partial products are stored as a 3 x 35 array. The following table shows the arrangement conceptually.

	g=0	g=1	. . .	g=33	g=34	g=35
s= 0	0	3		96	99	102
s = 1	35	38		26	29	32
s = 2	70	73		61	64	67

When we load the transputer network with the partial results, we do so according to the order of dimension 35, i.e. the data are stored in three columns of 35 rows each as shown in the following table :

	c=0	c=1	c=2
s = 0	0	35	70
1	3	38	73
2	6	41	76
	•	•	•
	•	•	•
	•	•	•
32	96	26	61
33	99	29	64
34	102	32	67

Elements of column 1 are distributed in the left branch while those of columns 2 and 3 are in the central and right branches respectively. As stated in theorem 1, sequence numbers in this dimension are the group numbers of dimension 3.

In dimension 35, we can predict from the *known* sequel numbers the indices of data points in dimension 5 or 7 according to Theorem 4. In dimension 5, $N_2 = 5$ and $N_3 = 7$, $<5^{-1}>_7 \equiv <7^{-1}>_5 = 3$. Let us take data 96 ($c = 0, s = 32$) as an example. From Theorem 4, $g_2 = <3\times32>_7 = 5$ and $s_2 = <3\times32>_5 = 1$; therefore data point 96 should be routed to ($c = 0, g = 5, s = 1$). We check this by listing the addresses according to $n = <0 + 3<5g_2 + 7s_2>_{35}>_{105}$:

Table 2-1 : Loading addresses according to
$n = <0 + 3<5g_2 + 7s_2>_{35}>_{105}$

c = 0

	g = 0	1	2	3	4	5	6
s = 0	0	15	30	45	60	75	90
1	21	36	51	66	81	96	6
2	42	57	72	87	102	12	27
3	63	78	93	3	18	33	48
4	84	99	9	24	39	54	69

From the above table, we can confirm that data 96 is at ($c = 0, g = 5, s = 1$)

After considering the loading addresses of the data, let us examine their retrieval addresses. The purpose of the retrieval equation is to shuffle the data according to the computed addresses. In fact this shuffling can be realized either by switching the contents of cells in the array or by *modifying the addresses* associated with these cells. Since we are using the in-place algorithm, the modification of addresses is very simple because only the *sequel numbers* of the data need to be updated. A recursive equation, which involves only simple addition, is stated in

Theorem 5 for updating the sequel numbers. Moreover, only one set of new sequel numbers needs to be calculated for all the groups in a certain dimension as the elements having the same sequel numbers are shuffled in exactly the same order within their individual groups. The retrieval equation for dimension 5 is given by $k = <0 + 3<5g_2 + 21s_2>_{35}>_{105}$. Table2-2 lists all the addresses of this 5x7 array.

Table 2-2 : Loading addresses according to
$n = <0 + 3<5g_2 + 21s_2>_{35}>_{105}$

c = 0

	g = 0	1	2	3	4	5	6
s = 0	0	15	30	45	60	75	90
1	63	78	93	3	18	33	48
2	21	36	51	66	81	96	6
3	84	99	9	24	39	54	69
4	42	57	72	87	102	12	27

The scrambling can be easily predicted by $s_{(i)}=s_{(i-1)} + <N_3>_{N_2}$ (Theorem 5), where $i = 0,1,..., N_2-1$. In the above example, $N_2=5$, $N_3 = 7$, $s_{1(0)} = 0$ and $<7>_5 = 2$. Hence $s_{(i)}=s_{(i-1)} + 2$ modulo 5, giving $s = 0,2,4,1,3$. Mapping the sequel numbers in Table 2-1into the new set of sequel numbers generates the addressing scheme shown in Table 2-3, which is equivalent to the scheme in Table 2-2, the retrieval addresses. The sequel numbers are significant because they are related to the power of the kernel

Table 2-3 : Retrieval addresses obtained by re-mapping the sequel numbers of the loading addresses.

c = 0

	g = 0	1	2	3	4	5	6
s = 0	0	15	30	45	60	75	90
2	21	36	51	66	81	96	6
4	42	57	72	87	102	12	27
1	63	78	93	3	18	33	48
3	84	99	9	24	39	54	69

in the DFT; for example, in dimension N_3, $X(k) = \frac{1}{N_3}\sum_{s_3=0}^{N_3-1} x(s_1)W_{N_3}^{k\,s_3}$. Provided that the data points are maintained in the proper order for each computation, it is insignificant whether these data are physically scrambled or not in the memory storage.

2.5.3 Procedure in a three-dimensional case : As a summary to all the details described above, let us go through a complete computational procedure of a transform of $N = 3x5x7 = 105$.

(1) Load the input data group by group from the root processor to all three branch processors 1l,1c and lr according to $<3g_1 + 35\, s_1>$, where $g_1 = 0,1,2,...,34$ and $s_1 = 0,1,2$.

(2) Each processor is to compute one component of each length-3 transform. Processor ll computes the first component : $X(0) = x(0)+x(1)+x(2)$, processor lc computes the third component : $X(2) = x(0)+x(1)W_3^2 + x(1)W_3^1$, and processor 1r computes the second : $X(1)=x(0)+x(1)W_3^1+ x(2)W_3^2$.

(3) The sequel numbers s_1 become the class number of dimension 5x7. By labelling the partial products from 1l as class 0, from 1c as class 1 and from 1r as class 2, the retrieval equation for dimension 3 has been implemented.

(4) The group numbers $g_1 = 0,1,...,34$ are used for computing the addresses for the next dimensions in all classes according to $g_2 = <3g_1>_7$ and $s_2 = <3g_1>_5$ (Theorem 4).

(5) Each branch processor keeps a loading assignment for itself and its leaves according to g_2 so that the partial products can be routed to a suitable processor in the following dimension. One possible assignment is :

Processor number	Group number g_2
1	0,1
2	2,3
3	4
4	5
5	6

(6) All branch and leaf processors compute concurrently the length-5 transforms of the groups according to their sequel numbers. After the computation, re-map all sequel numbers according to $s_{2(i)} = s_{2(i-1)} + 2$ (Theorem 5).

(7) For dimension 7, $g_3 = s_2$ and $s_3 = g_2$ (Theorem 1). Route the partial products with their indices to the appropriate processors according to the assignment for g_3.

(8) All branch and leaf processors compute concurrently the length-7 transforms of the groups according to their sequel numbers. After the computation, re-map all sequel numbers according to $s_{3(i)} = s_{3(i-1)} + 5$ (or -2) (Theorem 5).

(9) For all three classes, restore g_1 (which is the sequel numbers of dimension 35) by $g_1 = <7g_3 + 5\, s_3 >_{35}$. Then re-map $g_{1(i)} = g_{1(i-1)} + 3$ (Theorem 5).

(10) Restore all the one-dimensional addresses by $k = <35c + 3g_1>_{105}$.

(11) Route the results to the host computer.

Although the above example is set for a three-dimensional case, the above procedure can be extended into any number of dimension. Because the PFA is applied

recursively, at every stage we can choose to deal with at most three dimensions. Supposing $N = N_1N_2N_3N_4$, at the first stage we handle dimensions N_1, N_2 and N_3xN_4. Then for the next stage we handle dimension N_2, N_3 and N_4. So in the second stage there are 4 indices to represent each address. Although the addresses of the data can be expressed implicitly by the addresses of their locations in the memory, it is safer to include the addresses in form of the indices as a header to the data points. For a four-dimensional case, only the indices of the last three dimensions are sufficient since the first indices are implied by the location of the three branches. As the address generation (by integer calculation) takes a negligible duration in comparison to the complex calculation of DFTs, whether the addresses are pre-calculated or not is not significant.

2.5.4 Analysis in timing : In a multiprocessor network, communication between processors is usually time consuming and should be kept to a minimal. The tree-network is so connected that it is most suitable for computing a transform of sequence length N = 3x5x... , where exchanges of partial products are required only after dimension 5. If N = 3x5x7x11, after the computation of dimension 3, the data are neatly distributed among the three branches which do no communication with each other. After dimension 5, partial products must be exchanged between each branch processor and its own leaves. Having completed the exchanges, each processor holds all the 77 data required for the remaining computation, which means that no further communication is needed. This is valid even if the sequence length is extended to more than 4 dimensions.

A program was written in OCCAM for implementing a transform of N = 3x5x7x11 = 1155 and loaded into the tree-network simulated by our 16 -node

transputer network called SUPERLINK [21]. For each node, a T800 transputer is used with 32 Kwords local memory. The clock rate of the system is 20 MHz and the speed of the communication links is at 10 Mbits/sec. The process begins by passing the 1155 points of data (in form of complex numbers, 2x32 bits) in groups of 3 and in triplicate to all the branch processors simultaneously. The duration takes about 52 ms. The DFTs are performed by direct calculation. In dimension 3, 385 points of length-3 DFTs are evaluated by each processor, with a maximum duration of 12 ms. The loading assignment for the partial products is :

Processor number	Group number
1	0 to 15
2	16 to 31
3	32 to 46
4	47 to 61
5	62 to 76

The partial products are then distributed to the 4 leaf processors for the length-5 calculation. The passage of data from the root, the length-3 DFT collocation and the distribution of partial products after dimension 3 are performed in a pipeline. Hence the duration of the passage of data from the root (52 ms) covers the total processing time for these three operations. In the dimension 5, the evaluation time for at most 80 data points (16x5) by 1 processor is about 4.2 ms. The communication time for passing for the same number of data points from one processor to its neighbour is about 3.6ms. The longest path, which involves routing through an intermediate processor, takes 7.2 ms. The Transputer Technical Notes [22] suggest that the computation and the communication process can be decoupled, but our simulation reveals that these two processes cannot be totally independent of each other as synchronous communication is adopted. Since there is also a set-up time for each

communication, to transfer the partial products in blocks are more efficient. In dimension 7 and 11, no exchanges of data between processors are required. The address generation time is very small as compared to that for evaluating the DFT - it takes about 0.97 μs to produce one set of group number and sequel number. After the final stage of computation, the results are distributed among all the processors and must be returned to the host system through the root processor. It takes another 52 ms to complete this passage. A summary of the timing is tabulated in Table 2-4.

2.5.5 Concluding remarks : From Table 2-4, it is obvious that the communication time, which actually is associated with the hardware, dominates the total duration of the process. Given that there are available faster communication channels or alternative arrangements in memory storage like dual port RAM or direct memory access, the communication time can be greatly reduced. The address generation takes only a negligible amount of time. Furthermore these addresses can be generated beforehand. To make a fairer comparison, let us consider only the computational time. The total computational time in the tree-network is 34.5 ms. Had the whole process be implemented by a single transputer, the computational time would be 370 ms. Hence the speed-up factor :

$$\frac{\text{computational time by 1 processor}}{\text{computational time by network}} = \frac{370}{34.5} = 10.7$$

Table 2-4 : Timing of the tree-network

	max. data points /processor	communication time /processor	computational time/processor	address gen. time
from root	1155	52 ms	0	0
length 3	365		12 ms	0.37 ms
length 5	80	3.6 ms*	4.2 ms	74 μs
length 7	77	0	6.6 ms	74 μs
length 11	77	0	11.7 ms	74 μs

* The longest route takes twice the duration

Excluding the root processor, there are 15 processors in the network carrying out the evaluation. The efficiency is :

$$\frac{\text{speed-up factor}}{\text{total number of processors}} = \frac{10.7}{15} = 71\%$$

When a multi-processor system is used for evaluating DFT by PFA, the data transfer between processors must be kept to a minimal level in order to achieve a high turnover of results. The tree-network is a suitable topology for a PFA of 3x5x ... because data transfer is done only after dimension 5 and among the processors within the same branch. The addresses in the form of indices of different dimensions and the address prediction scheme introduced are very convenient in determining the addresses of data in the next dimension. In most practical cases, a sequence length of 3x5x7x11 = 1155 for DFT is quite sufficient. Therefore, a four-dimensional PFA should be most common.

CHAPTER 3

SUB-NYQUIST SAMPLING

When an analog signal is processed by a digital system, sampling of the input signal must be done. To choose a suitable sampling period T or equivalently a sampling rate $F_S = 1/T$, we must have some information concerning the frequency content of the signal. In general, all the signals related to engineering, such as speech or television, can be represented over a short time segment as a sum of sinusoids of different amplitudes, frequencies and phases :

$$x_a(t) = \sum_{i=1}^{N} A_i \cos(2\pi f_i t + \theta_i)$$

For each of these signals, however, the maximum frequency does not exceed a known frequency F_{max} . For example, F_{max} = 3 kHz for speech and F_{max} = 6 MHz for television. Based on the knowledge of this F_{max} , an appropriate sampling frequency F_S can be determined. We know that the highest frequency in an analog signal which can be unambiguously reconstructed is $F_S/2$ when the signal is sampled at F_S. Any frequency above $F_S/2$ will yield sample values identical to a corresponding frequency

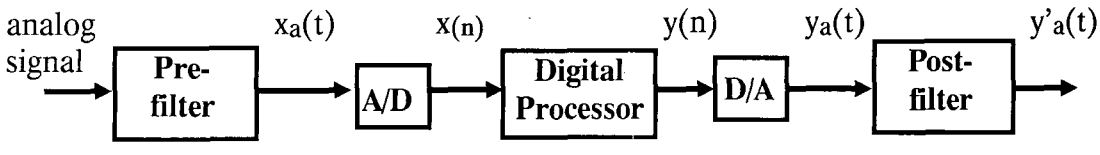


Fig. 3-1 Block diagram of a digital system processing an analog signal

in the range $-F_s/2 \leq f \leq Fs/2$. These ambiguities arising from aliasing can be avoided by selecting a sufficiently high sampling rate which is $F_s \geq 2F_{max}$.

3.1 Shannon Sampling Theorem

A band-limited continuous-time signal, with highest frequency (bandwidth) B hertz, can be uniquely recovered from its samples provided that the sampling rate $F_s \geq 2B$ samples per second.

According to this theorem, if the highest frequency contained in an analog signal $x_a(t)$ is $F_{max} = B$ and the signal is sampled at a rate $F_s > 2F_{max} = 2B$, then $x_a(t)$ can be exactly recovered from its sample values using the interpolation function

$$g(t) = \frac{\sin 2\pi Bt}{2\pi Bt}$$

Thus the signal is reconstructed by :

$$x_a(t) = \sum_{n=-\infty}^{\infty} x_a\left(\frac{n}{F_s}\right) g\left(t - \frac{n}{F_s}\right),$$

where $x_a\left(\frac{n}{F_s}\right) = x_a(nT) = x(n)$ are the samples of $x_a(t)$. This is called the ideal interpolation formula, of which the proof can be found in many reference books [1,10].

When the sampling of $x_a(t)$ is done at a minimum sampling rate $F_s = 2B$, the reconstruction becomes :

$$x_a(t) = \sum_{n=-\infty}^{\infty} x_a\left(\frac{n}{2B}\right) \frac{\sin 2\pi B(t-n/2B)}{2B(t-n/2B)}.$$

The above series is known as the Cardinal Series of Shannon Sampling Theorem[23]. The sampling rate $F_s = 2B = 2F_{max}$ is called the *Nyquist rate*. When aliasing occurs due to a sampling frequency lower than the Nyquist rate, the effect can be understood

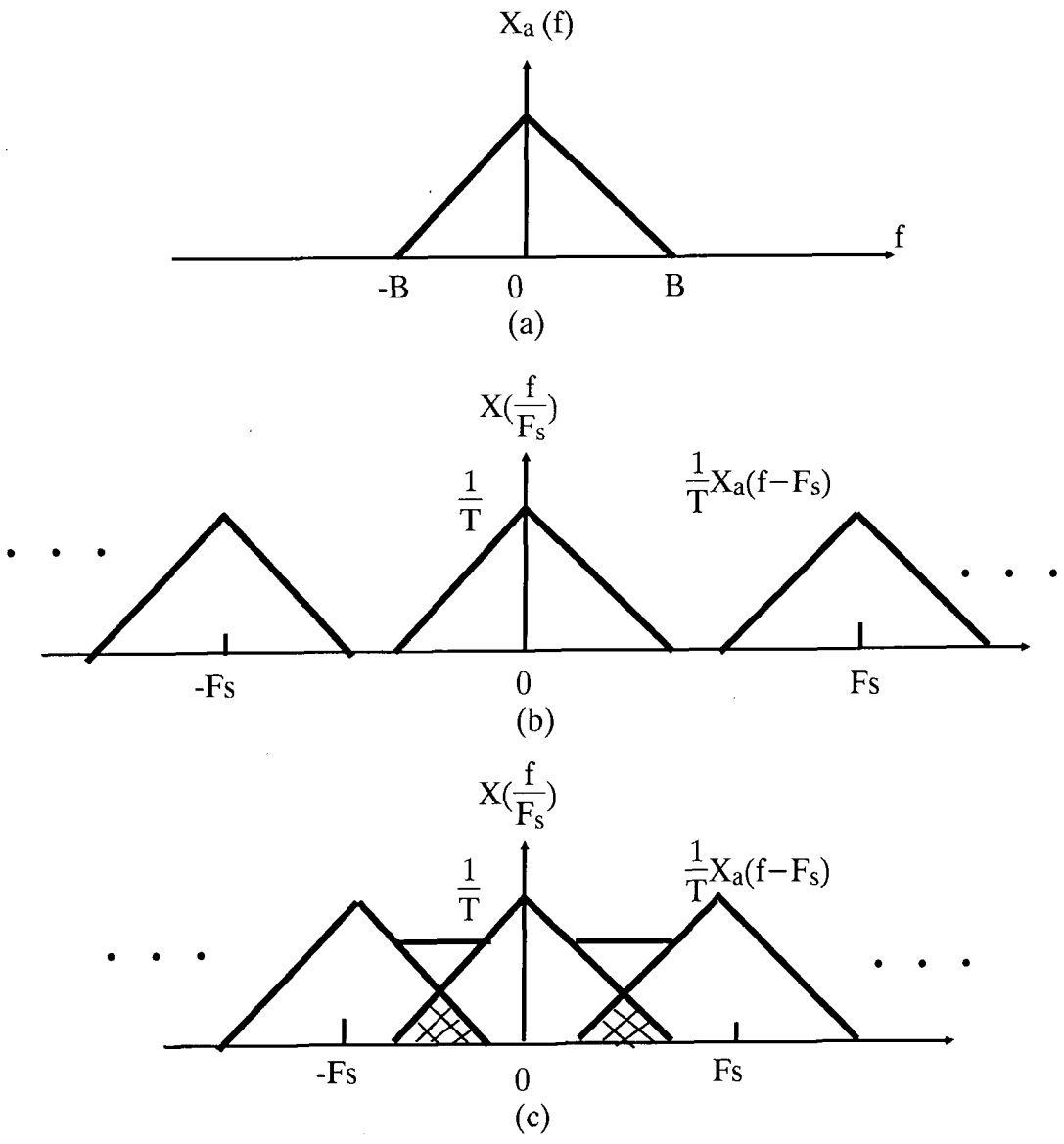


Fig . 3 -2 Aliasing around the folding frequency (a) original spectrum (b) reconstructed spectrum with no aliasing (c) reconstructed spectrum with aliasing.

as a multiple folding of the frequency axis of the frequency variable f for the analog signal. Any sampling process of which the average sampling rate is below the Nyquist rate can be called a sub-Nyquist sampling process.

3.2 Randomized Sampling

A regular sampling sequence can be described mathematically by

$$u(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{3-1}$$

where T is the sampling interval and δ is the Dirac function. To make this description applicable to irregular sampling, the above equation can be modified to a more general form as

$$u(t) = \sum_{n=-\infty}^{\infty} \delta(t - t_n) \tag{3-2}$$

where t_n is the sampling time at interval n. For regular sampling, $t_n = n\,T$. If sampling is performed at random instants, we can write

$$t_n = nT + \tau_n \qquad n = 0,1,2,... \tag{3-3}$$

or

$$t_n = t_{n-1} + \tau_n \qquad n = 0,1,2,... \tag{3-4}$$

where τ is a random variable. The sampling intervals chosen according to eqn (3-3) adds a jitter to the regular sampling interval. Eqn (3-4) describes a process which is known as additive random point process. It was proposed by Shapiro and Silverman [24] as a convenient tool for performing randomized sampling. Hence we may call the sampling derived from the above two process as the *jittered sampling method* and the *additive random sampling method.* When performing the additive random sampling, the randomness introduced into the sampling process can be controlled by one parameter, the ratio σ/μ, where σ and μ are the standard deviation and the mean value respectively of the sampling periods defined as t_n - t_{n-1} at interval n. If $\sigma/\mu = 0$, the sampling is periodic and the time intervals between the points are equal to T. By increasing this ratio, it is possible to obtain an extremely randomized sampling.

As discussed in section 3-1, if the sampling rate of a regular sampling sequence is below the Nyquist rate, aliasing will occur. In section 2.1, the periodic extension of

a regularly sampled signal is briefly mentioned. It can be shown that aliasing is in fact associated with this extension on a regular sampling grid. Using a suitable randomized sampling scheme, an alias-free[1], sub-Nyquist sampling can be achieved. Shapiro and Silverman [24], Beutler [25] and Masry [26] published various theorems and criteria on alias-free sampling.

3.2.1 Theory of Shapiro and Silverman : Shapiro and Silverman argue that jittered sampling is not alias-free since the sampling time t_n are still "attracted" to the equi-space value *nh*, where h is the average sampling period. What is needed to break up this regularity is some sort of "floating point" sampling scheme. Thus the additive random sampling, in which each sampling time is derived from the preceding one by the addition of an independent random variable, is introduced. Keeping their notations, $t_n = t_{n-1} + \gamma_n$, where γ_n, n= ...,-2,-1,0,1,2,..., is a family of identically distributed, independent random variables, with E $[\gamma_n]$ =h $< \infty$ and a common probability density $p(\tau)$. Of course $p(\tau) \geq 0$ and $\int_{-\infty}^{\infty} p(\tau)\, d\tau = 1$. The Fourier transform of $p(\tau)$

$$\varphi(\omega) = \int_{-\infty}^{\infty} \exp(i\omega\tau)\, p(\tau)\, d\tau = \int_{0}^{\infty} \exp(i\omega\tau)\, p(\tau)\, d\tau \qquad \textbf{(3-5)}$$

is the characteristic function (in the sense of probability theory) of the distribution $p(\tau)$.

Shapiro and Silverman study the correlation function of a random sequence and subsequently propose the following theorem : *Additive random sampling is alias free if the characteristic function $\varphi(\omega)$ takes no value more than once on the real axis.*

[1] Sampling is alias free iff the translates by $2\pi n$ (n any integer) of the support of the spectrum of x(t) are all disjoint.

First they show that the correlation sequence $c_h(n)$ and its correlation function $C(\tau)$ of a random sequence $x_n(t)$ is given by :

$$c_h(n) = \mathrm{E}\,[x(t_{m+n})\,x(t_m)] = \int_{-\infty}^{\infty} C(\tau)\,p_n(\tau)\,d\tau$$

where $p_n(\tau) = \int_0^{\tau} p_{n-1}(\tau-u)\,p(u)du$, $n \geq 2$, and $p_1(\tau) = p(\tau)$. For an additive random sampling sequence :

$$c_h(n) = \int_0^{\infty} C(\tau)\,p_n(\tau)\,d\tau\,, \qquad n \geq 1$$

$$c_h(n) = C(0)\,.$$

By Parseval's theorem and the convolution theorem for Fourier transform, $c_h(n)$ is related to the power spectrum $F(\omega)$:

$$c_h(n) = \int_{-\infty}^{\infty} F(\omega)\,\varphi^n(\omega)\,d\omega\,, \qquad n \geq 0 \tag{3-6}$$

The question to be answered is : For which $p(\tau)$ is there only one correlation function which leads via eqn (3-6) to a given correlation sequence $c_h(n)$? if no aliasing occurs, eqn (3-6) is satisfied for only one distinct real (non-negative) function in $L^1 \cap L^2$. By conformal mapping, it is found that the required $p(\tau)$ has its characteristic function $\varphi(\omega)$ which is one-to-one on the real axis. This is the sufficient condition for additive random sampling to be alias-free.

Finally, the "Poisson sampling" with an average rate ρ and $p(\tau) = \rho\exp(-\rho\tau)$ for $\tau \geq 0$ and $p(\tau) = 0$ for $\tau < 0$ is verified to be alias free.

3.2.2 Beutler's Theory : Beutler defines that *a sampling sequence* $\{t_n\}$ *is alias free relative to S (a family of spectra) if no two random processes with different spectra belonging to S yield the same correlation sequence* $\{r(n)\}$.

The correlation sequence of the discrete process $\{x(t_n)\}$ is denoted by $r(n) = E[x(t_{m+n})x^*(t_m)]$ where * indicates complex conjugacy. It is also supposed that the probability distributions of $(t_{m+n} - t_m)$ do not depend on m. Under the assumption that $\{x(t)\}$ is wide-sense stationary, r(n) depends only on n. Two random processes with respective spectra G_1 and G_2 are said to have different spectra if there exists a continuous function f such that

$$\int_{-\infty}^{\infty} f(\omega)\, dH(\omega) \neq 0$$

where $H = G_1 - G_2$. Thus spectra are regarded as identical iff they differ by at most a constant at all their points of continuity.

Whether a given $\{t_n\}$ is alias free relative to S depends on the relation between G and $\{r(n)\}$. The expectation on x(t) is most conveniently written in terms of its spectral representation :

$$E[x(t_{m+n})x^*(t_m)] = \frac{1}{2\pi}\int_{-\infty}^{\infty} \left\{\exp[i\omega(t_{m+n} - t_m)]\right\} dG(\omega) \qquad \textbf{(3-7)}$$

Consequently :

$$r(n) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f_n^*(i\omega)\, dG(\omega).$$

By definition $f_n^*(n) = E\left\{\exp[-i\omega(t_{m+n} - t_m)]\right\} = \int_{-\infty}^{\infty} e^{i\omega u}\, dF_n(u)$, **(3-8)**

in which F_n is the probability distribution function for n successive sampling intervals. Note that the difference in sign of the exponential in (3-7) and (3-8) is irrelevant because of the character of G for a real stochastic process. For each possible set of $\{t_n\}$ statistics, the mapping $G \rightarrow \{r(n)\}$ is a linear bounded transformation. An inverse exists in the sense that G can be inferred from $\{r(n)\}$ iff this transformation is

one-to-one. Thus the definition identifies the spectral recovery capability with the alias-free property.

Let β_S^* be the family of measures induced by H of the form $H = G_1 - G_2$, where G_1 and G_2 are any spectra belonging to S. In particular the null measure is in β_S^* and we have the following theorem : *The sampling sequence* $\{t_n\}$ *is alias free relative to S iff with* $H \in \beta_S^*$

$$\int_{-\infty}^{\infty} f_n^* \, dH(\omega) \qquad \text{for all } n \text{ implies } H = 0. \tag{3-9}$$

From the above theorem, the following corollaries are deduced.

Corollary 1 : Let β_S be a Banach space[2] with dual β_S^*, and assume that each $f_n^* \in \beta_S^*$. Then $\{t_n\}$ is alias free relative to S iff $\{f_n^*\}$ is closed in β_S.

Since closure and completeness are equivalent in a Banach space, Corollary 1 can be rephrased as :

Corollary 2 : Under the hypotheses of Corollary 1, $\{t_n\}$ is alias free with respect to S iff every $g \in \beta_S^*$ can be approximated as closely as desired (in β_S^* norm) by finite linear combinations of the f_n^*.

Assume that $(t_{m+n} - t_m)$ has a probability density f_n. A substitution of (3-8) in (3-9) followed by an application of Fubini's theorem yields the following.

Corollary 3 : Let D be the difference of two correlation functions corresponding to spectra belonging to S. Then $\{t_n\}$ is alias free relative to S iff

$$\int_0^{\infty} f_n(u) \, D(u) du \qquad \text{for all n implies } D = 0.$$

[2] A Banach space is a normed space which is complete in the metric defined by its norm; this means that every Cauchy sequence is required to converge [27].

After setting the above criteria, three examples of $\{t_n\}$ are examined to see whether they are alias free. The first example is the Poisson point process. The second example is a sequence sampled on a finite interval by periodic sampling, in which samples are randomly and independently skipped such that the average sampling rate is an arbitrary small fraction of the Nyquist rate. The third example is the randomly jittered sampling at Nyquist rate. All the three examples are found to be alias free.

3.2.3 Expectation value of Spectrum : The spectrum of a signal x(t) sampled in a duration T_o can be given by the Fourier Series

$$S(f_k) = \frac{2}{T_o}\int_o^{T_o} x(t)\, exp(-j2\pi f_k t)\, dt \qquad \textbf{(3-10)}$$

where $f_k = k/T_o$ and k is an integer. When x(t) is sampled by a randomized timing sequence $\{t_n\}$, according to Bilinskis and Mikelsons [28], the spectrum of the sampled signal $\{x(t_n)\}$:

$$S_s(f_k) = \lim_{\Theta\to\infty}\frac{2}{\Theta}\int_0^{\Theta} x(t)\, u(t)\, \exp(-j2\pi f_k t)\, dt \qquad \textbf{(3-11)}$$

where $\Theta = E[t_N]$, N is the number of samples processed, $f_k = k/\Theta$ and k is an integer. Making use of eqn (3-2), $\{x(t_n)\} = \int_{-\infty}^{\infty} x(t)\,\delta(t - t_n)\, dt$, it follows that

$$S_s(f_k) = \lim_{N\to\infty}\frac{2}{N}\sum_{n=0}^{N-1} x(t_n)\, \exp(-j2\pi f_k t_n) \qquad \textbf{(3-12)}$$

Let us consider the case when sampling is performed periodically. Then the function $u(t) = \delta(t - t_n)$ is also periodic and its Fourier series is

$$u(t) = \frac{1}{T_o} + \frac{1}{T_o}\sum_{r=1}^{\infty}[\exp(-2\pi rt/T_o) + \exp(2\pi rt/T_o)] \qquad \textbf{(3-13)}$$

Substituting (3-13) into (3-11) and neglecting the scaling factor $1/T_o$ yields

$$S_s(f_k) = \lim_{\Theta\to\infty}\frac{2}{\Theta}\int_0^{\Theta} x(t)\exp(-j2\pi f_k t)\left\{1+\sum_{r=1}^{\infty}[\exp(-j2\pi rt/T_o)+\exp(j2\pi rt/T_o)]\right\}dt$$

$$= \lim_{\Theta\to\infty}\frac{2}{\Theta}\int_0^{\Theta} x(t)\exp(-j2\pi f_k t)\,dt + \sum_{r=1}^{\infty}\lim_{\Theta\to\infty}\frac{2}{\Theta}\int_0^{\Theta} x(t)$$

$$\times[\exp(-j2\pi rt/T_o)+\exp(j2\pi rt/T_o)]\,dt$$

$$= S_x(f_k) + \sum_{r=1}^{\infty}\left[S_x\left(\frac{r}{T_0}+f_k\right) + S_x^*\left(\frac{r}{T_0}-f_k\right)\right] \qquad \textbf{(3-14)}$$

where $S_x(f_k) = \lim_{\Theta\to\infty}\frac{2}{\Theta}\int_0^{\Theta} x(t)\exp(-j2\pi f_k t)\,dt$ is the spectrum of the original signal and $S_x^*(f_k)$ is the complex conjugate of $S_x(f_k)$. Eqn (3-14) shows that replicas of the original spectrum are found periodically in the frequency domain as depicted in Fig. 3-2 (b). When the sampling frequency is lower than the Nyquist rate, aliasing will occur because of insufficient separation as depicted in Fig. 3-2 (c).

Now assume that $\{t_n\}$ is a random sampling sequence, and $p_n(t)$ be the probability density function (p.d.f.) of the nth random variable in u(t). From eqn (3-11), the *expectation value* of the estimated spectrum :

$$E\left[\lim_{\Theta\to\infty}S_s(f_k)\right] = \lim_{\Theta\to\infty}\frac{2}{\Theta}\int_0^{\Theta}\sum_{n=1}^{N}p_n(t)\,[x(t)\ \exp(-j2\pi f_k t)]\,dt \qquad \textbf{(3-15)}$$

If the "total" p.d.f. at time t, $p(t) = \sum_{n=1}^{N}p_n(t) = \frac{1}{\mu}$ = a constant, then **(3-16)**

$$E\left[\lim_{\Theta\to\infty}S_s(f_k)\right] = \lim_{\Theta\to\infty}\frac{2}{\Theta\mu}\int_0^{\Theta} x(t)\exp(-j2\pi f_k t)\,dt$$

Hence the expectation of the estimated spectrum approaches the original spectrum if the condition specified by eqn (3-16) is satisfied. Note that no replicas of the original spectrum nor any aliasing is produced.

3.2.4 Jittered Random Sampling : Bilinskis and Mikelsons discuss the alias-free property of the jittered random sampling and the additive random sampling by looking at the probability density functions (p.d.f.) of the random variable τ inserted into the sampling intervals. For the *jittered random sampling*, the timing model is given by

$$t_n = nT + \tau_n \qquad n = 0,1,2,...$$

where $\{\tau_n\}$ is a family of independent identically distributed random variables with zero mean. Let $t_0 = 0$, the "total" p.d.f. of the time intervals at time t from t_0 is

$$p(t) = \sum_{n=1}^{N} p_n(t), \qquad N \to \infty$$

where $p_n(t)$ is the p.d.f. of τ_n. To understand the meaning of the function p(t), imagine that a narrow time window Δt is moved along the time axis. Under the condition that $\Delta t \to 0$, p(t) at any time t is equal to the probability that one of the sampling points t_s will fall within this window Δt. Fig. 3-3 shows the p.d.f. of a set of random variables belonging to a stream of jittered sampling points for n = 1 to 7. As can be seen from the figure, this particular function has multiple maxima and minima and the peaks do not change as t increases. Obviously, this p(t) does not satisfy eqn (3-16).

If the time intervals are distributed uniformly in the intervals $nT \pm 0.5T$, then the resulting p(t) of the sampling points is constant for $t > 0.5T$, which can be seen in Fig. 3-4. When this sampling scheme is applied, all instantaneous signal values are sampled with an equal probability and eqn(3-16) is satisfied. It seems, therefore, that this method of generating random sampling points is acceptable. However, there are a number of substantial disadvantages which prevents the wide application of this method. These drawbacks are :

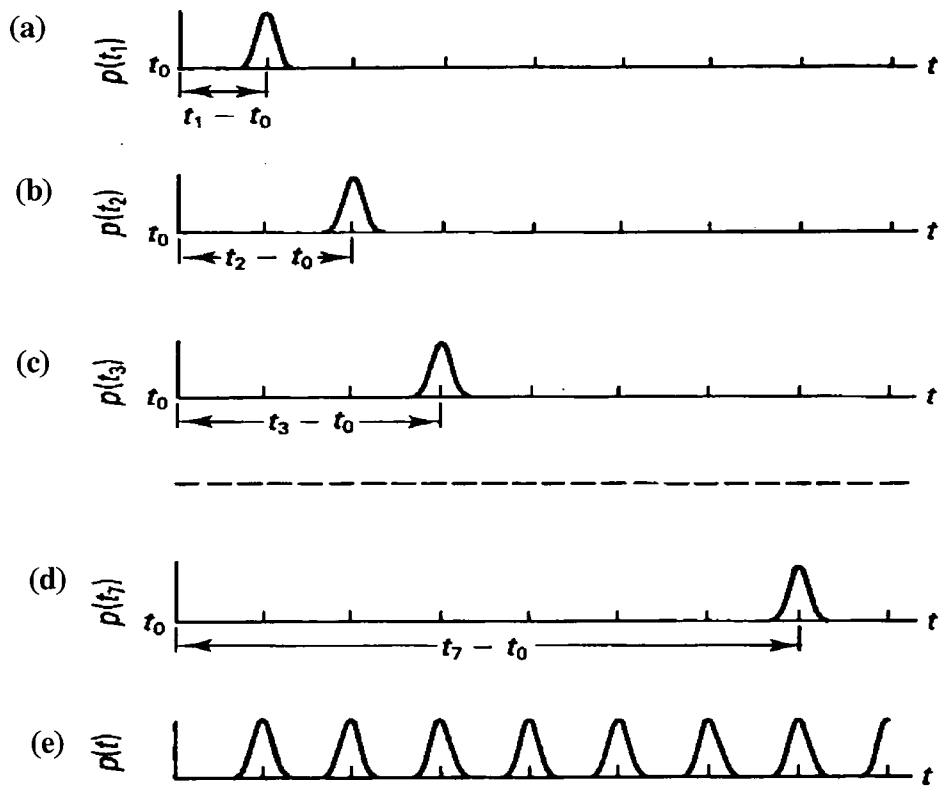


Fig. 3-3 Probability density functions characterizing the jittered sampling. (a) ,(b),(c) and (d) : Probability density functions of time intervals at t_1, t_2,t_3 and t_4. (e) Resulting sampling point density function. (From [28])

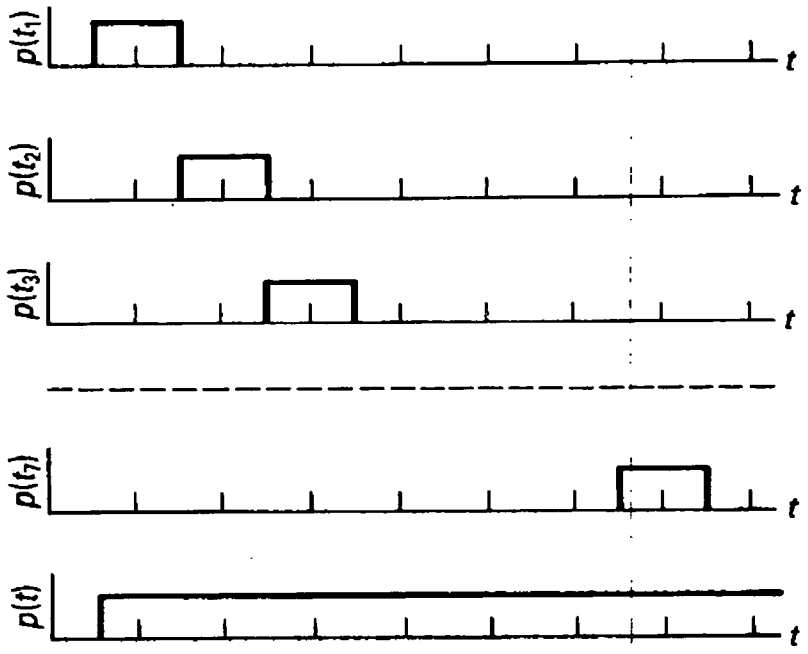


Fig . 3-4 The probability density function of a jittered sampling that τ_n are distributed uniformly in the range $T \pm 0.5T$. (From [28])

1. The random variables $\{\tau_n\}$should be distributed strictly uniform within the given intervals.

2. Time intervals between any two successive sampling instants t_n and t_{n+1} may be very short. The implementation of this scheme should be wide-band even at a relatively low mean sampling rate.

3. The randomness introduced is considerably large.

4. Statistical errors resulting from the relatively powerful randomness introduced at sampling are significant.

3.2.5 **Additive Random Sampling** : In case of *additive random sampling*, samples are taken at instants

$$t_n = t_{n+1} + \tau_n, \qquad n = 0,1,2,...$$

where τ_n is a realization of a random variable. Consider the time interval $[0,t_n] = t_1 + t_2 + ... + t_n$. The random variables are characterized by their respective p.d.f. $\{p_n(t)\}$. We can write

$$\begin{aligned} p_1(t) &= p_\tau(t) \\ p_2(t) &= p_1 * p_\tau(t) \\ &\cdots\cdots\cdots\cdots \\ p_n(t) &= p_{n-1} * p_\tau(t) \end{aligned}$$

where * denotes the "additive" operation. As the interval $[0,t_n]$ is a sum of n statistically independent random variables, we know from the central limit theorem that whatever p.d.f. these variables may have, the probability distribution of the random variable $[0,t_n]$ will approach the normal distribution as n approaches infinity. Fig. 3-5 shows two different p.d.f.s; one is uniform and the other is exponential. The

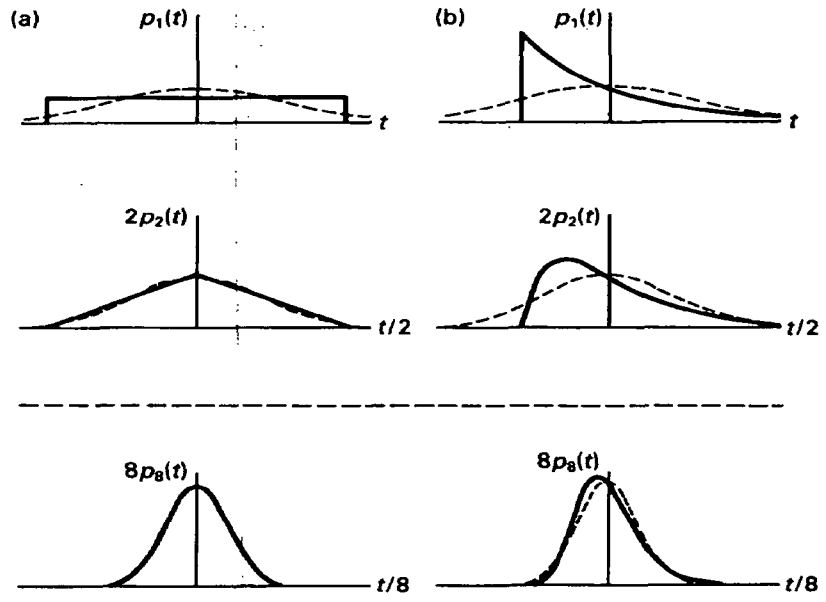


Fig. 3-5 Evolution of probability density function $p_r(t)$ when p1(t) is (a) uniform and (b) exponential. (From [28])

expected value of the time intervals $\{(t_n - t_{n-1})\}$ is μ. In both cases the corresponding $p_r(t)$ change and become more normally distributed as the summing of intervals proceeds from r = 1 to 8.

Consequently, when the additive random sampling is applied, no matter what form of $p_\tau(t)$ a variable τ may take, the sampling point density function p(t) will always tend to the constant level $1/\mu$ when t exceeds a certain time T_a which depends on $p_\tau(t)$. Fig. 3-6 shows the p.d.f. of an additive random sampling scheme from t_0 to t_8. It can be seen that p(t) tends to become flat as t progresses. Hence when t is large, p(t) = $1/\mu$, which means that eqn (3-16) can be satisfied.

Bilinskis and Mikelsons also analyze the case when the sampling intervals $\{t_n, t_n + 1\}$ are correlated. They give an example that the individual $\{t_n\}$ are distributed normally with a mean value μ and a standard deviation σ. The correlated additive random point process can be defined as

$$\begin{cases} t_1 = t_0 + \tau_1 = t_0 + \mu + \sigma\xi_1 \\ t_n = t_{n-1} + \tau_n = \mu + \rho(\tau_{n-1} - \mu) + \sigma\sqrt{1-\rho^2}\,\xi_n \qquad \text{for } n > 1 \end{cases}$$

where $\{\xi_n\}$ are the uncorrelated instantaneous values of a zero mean normal random process with a variance of 1 and ρ is the correlation coefficient defined as

$$\rho_m = \frac{E[\,\tau_n\,\tau_{n+m}\,]}{\sigma^2} = \rho^m$$

with $|\rho| < 1$ and $\lim_{m\to\infty} \rho_m = 0$. Fig. 3-7 shows a positively correlated a.r.s. in (a) and a negatively correlated a.r.s. in (b). It can be seen from the figure that a positive correlation function helps the sampling point density function converge to the constant level $1/\mu$ sooner. If, however, the correlation function is negative, the convergence of the sampling point density function will be slowed down.

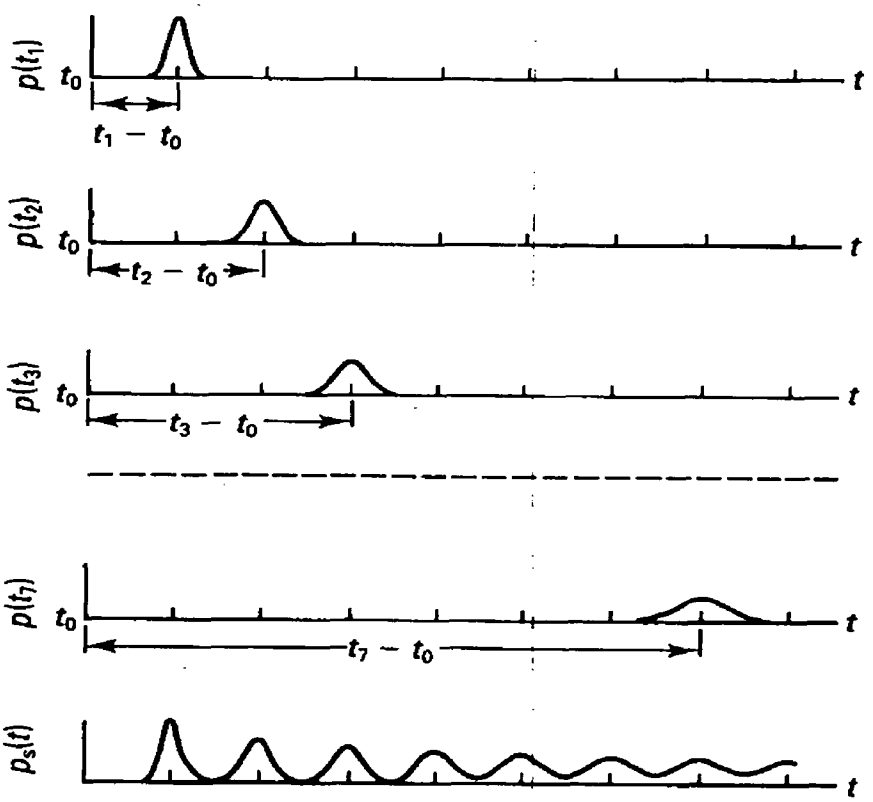


Fig. 3-6 Probability density functions characterizing additive random sampling : (a), (b), (c) , (d) Probability density functions of time intervals at t_1,t_2,t_3 and t_7. (e)Resulting sampling point density function. (From [28])

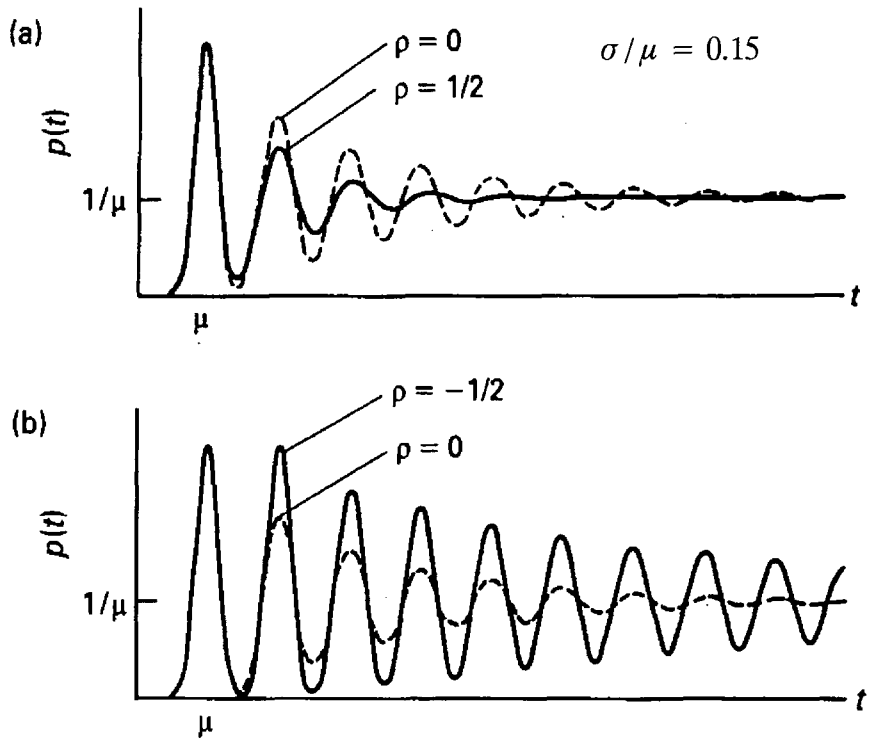


Fig. 3-7 Sampling point density functions for uncorrelated sampling (dashed lines) and correlated sampling of a.r.s. when (a) $\rho = \frac{1}{2}$*; (b)* $\rho = -\frac{1}{2}$ *(From [28]).*

From the analyses of various authors, we can conclude that additive random sampling surpasses jittered sampling in acting as an alias-free sampling process. The magnitude of the estimated spectrum recovered from the sequence sampled by a.r.s. approaches the exact values if the number of sampling points is large enough.

3.3 Realization of Additive Random Sampling

To sample a signal x(t) by a.r.s., the timing is specified by eqn (3-4), which is $t_n = t_{n-1} + \tau_n$, $n = 0,1,2,....$ After a sampled sequence $\{x(t_n)\}$ is obtained, the amplitude spectrum can be reconstructed by

$$X(0) = \frac{1}{N}\sum_{n=0}^{N-1} x(t_n)$$

$$X(k) = \frac{2}{N}\sum_{n=0}^{N-1} x(t_n)\exp(-j2\pi f_k t_n), \qquad k = 1,2,3,\ldots \tag{3-17}$$

where N is the number of samples and should be a large integer. Eqn(3-17) is a slight modification of eqn (3-12). In the normalized case, the total sampling period $T_0 = 1$ second and $f_0 = 1$ Hz, then $f_k = k$. Eqn (3-17) becomes

$$X(k) = \frac{2}{N}\sum_{n=0}^{N-1} x(t_n)\exp(-j2\pi k t_n), \qquad k = 1,2,3,\ldots \tag{3-17a}$$

Amplitude spectra of a signal reconstructed from regular sampling and a.r.s. are shown in Fig. 3-8 for comparison. Note that the Nyquist limit is at k = 512. When a.r.s. is adopted (Fig. 3-8 (b)), there are no longer any replicas of the original spectrum in the frequency domain. The sharp aliases in this case are turned into a broad-band background noise. Provided that the signal-to-noise ratio is high, the signal can be recognized easily by setting a suitable threshold in amplitude to pick out the signal.

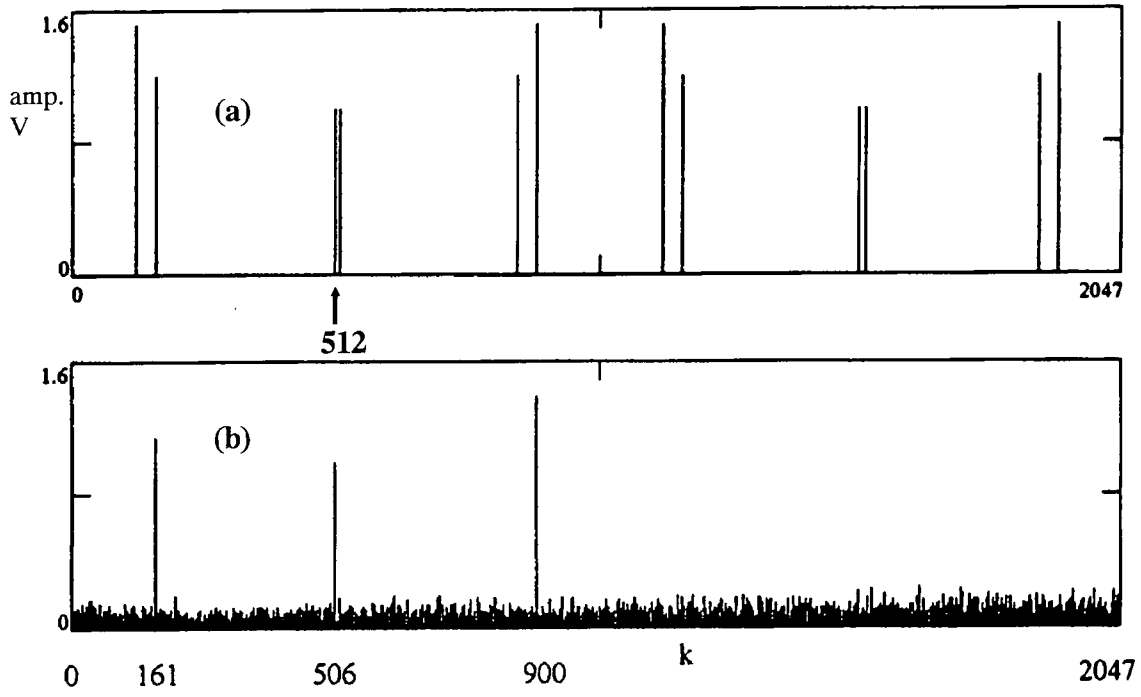


Fig. 3-8 Amplitude spectra of a signal sampled for 1024 points by (a) regular sampling, (b) additive random sampling. Note that the Nyquist limit is at k = 512.

In this example, the signal-to-noise ratio is 17.3 dB and the average accuracy of the recovered signal is 95.3 %.

3.3.1 Computational Complexity : In Chapter 2, the computational complexity of a few fast algorithms, including the best known radix-2 FFT, have been mentioned. The main saving in computation of these fast algorithms, in fact, comes from the trigonometric symmetry of the kernel of the transform, which is $\exp(-j2\pi kn/N)$. By adopting the "divide-and-conquer" approach, the FFT exploits this symmetry to such an extent that the complexity is reduced to $N\log_2 N$.

When random sampling is adopted, the kernel of the transform becomes $\exp(-j2\pi f_k t_n)$, where t_n is a random variable. At once we realize that the roots of unity are no longer lying evenly on the unit circle in the complex plane. Hence no symmetry is available and the complexity is obviously N^2. By nature, when random sampling is used, "fast" algorithms (in the sense of having a linear complexity, such as NlogN) are not available. A heavy computation load is therefore the price that randomized sampling has to pay.

3.3.2 Estimation of Noise Level : Assuming that the input signal is a simple sinusoid of unity amplitude, i.e. $x(t) = 1\exp(2\pi f_i t)$ volt, let us estimate the signal-to-noise ratio. The sampling time of a randomized sampling scheme is $t_n = nT + \beta_n$, where T is the mean sampling period and β_n is the deviation from the mean value at time t_n. Then the sampled signal is given by

$$x(n) = \exp\left[j2\pi f_i T\left(n + \frac{\beta_n}{T}\right)\right] = \exp\left[j2\pi \frac{k}{N}\left(n + \frac{\beta_n}{T}\right)\right]$$

since $f_i = k / NT$. Then the DFT of x(n) is, according to Berkovitz and Rusnak[29]:

$$X(l) = \sum_{n=0}^{N-1} \exp\left[\frac{j2\pi k}{N}\left(n + \frac{\beta_n}{T}\right)\right] \exp\left[\frac{-j2\pi l}{N}\left(n + \frac{\beta_n}{T}\right)\right]$$

$$= \sum_{n=0}^{N-1} \exp\left[\frac{j2\pi n}{N}(k-l)\right] \exp\left[\frac{j2\pi\beta_n}{NT}(k-l)\right] \quad \textbf{(3-18)}$$

To compute the power spectrum, eqn(3-18) is multiplied by its conjugate and then the expected value is taken :

$$E[I_N(l)] = \frac{1}{N}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} \exp\left[\frac{j2\pi}{N}(k-l)(n-m)\right] E\left\{\exp\left[\frac{j2\pi}{NT}(k-l)(\beta_n-\beta_m)\right]\right\}$$

(3-19)

The value of the expected value on the right hand side of eqn (3-19) is of the form :

$$E\beta = E\left\{\exp\left[\frac{j2\pi i}{NT}(\beta_n-\beta_m)\right]\right\} = \begin{cases} 1, & n=m \\ |q|^2, & n\neq m \end{cases}$$

where i is an integer. When l = k, E[I_N(l)] gives the spectrum of the original signal :

$$E[I_N(l)] = \frac{1}{N}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} \exp\left[\frac{j2\pi}{N}(0)(n-m)\right] = N \quad \textbf{(3-20)}$$

When $l \neq k$, eqn (3-19) gives the noise in the background. When n = m, $\exp\left[\frac{j2\pi n}{N}(k-l)(n-m)\right] = 1$ and $E\beta = 1$. There are totally N cases. Hence eqn (3-19) becomes :

$$E[I_N(l)] = \frac{1}{N}\sum_{n\neq m}\sum_{m\neq n} \exp\left[\frac{j2\pi}{N}(k-l)(n-m)\right] E\beta + 1$$

$$= \frac{1}{N}\sum_{n\neq m}\left\{\exp\left[\frac{j2\pi}{N}(k-l)(n-0)\right] E\beta 0 + \exp\left[\frac{j2\pi}{N}(k-l)(n-1)\right] E\beta 1 + \ldots\right\} + 1$$

(3-21)

There are N of these exponential terms and all the $E\beta$'s have the same value. Looking at one of these exponential terms :

$$\frac{1}{N}\left\{\sum_{n\neq m}\exp\left[\frac{j2\pi}{N}(k-l)(n-p)\right] E\beta + E\beta - E\beta\right\}$$

$$= \frac{E\beta}{N}\left\{\sum_{n=0}^{N-1}\exp\left[\frac{j2\pi}{N}(k-l)(n-p)\right] - 1\right\} = \frac{-|q|^2}{N} \qquad \textbf{(3-22)}$$

because (k-l) and (n-p) are integers, so the summation is of the form $\sum_{i=0}^{N-1}\exp\left(\frac{j2\pi i}{N}\right) =$ 0. Substituting eqn(3-22) into (3-21) yields :

$$E[I_N(l)] = 1 - |q|^2 \qquad \textbf{(3-23)}$$

3.3.2.1 Noise and Signal Power : From eqn (3-20) and (3-23), the average signal-to-noise ratio is N : $(1 - |q|^2)$ and the maximum noise power is obtained when $|q|^2 \to 0$. As a rough estimation, within the band from k = 0 to N-1, if the input x(t) = exp ($2\pi f_i t + \theta$), the minimum signal-to-noise ratio is N : 1. Hence in a band of N frequency components, the signal-to-noise ratio for 1 watt of input is :

$$S/N = N : 1 \text{ (or } 10\log_{10}N \text{ dB)} \qquad \textbf{(3-24a)}$$

If amplitude is concerned, for 1 volt of input in the same band, the ratio becomes :

$$S/N = \sqrt{N} : 1 \text{ (or } 20\log_{10}\sqrt{N} \text{ dB)} \qquad \textbf{(3-24b)}$$

From eqn(3-24a), it can be seen that noise power is 1/N watt in a band of N frequency components per watt input. Hence the noise spectral density per watt input is :

$$\eta = \frac{1}{N^2} \text{ (W per frequency resolution)} \qquad \textbf{(3-24c)}$$

The above analysis is based on the timing model of the jittered sampling. In case of the additive random sampling, the expected value of the exponential term of the random variable should be [29] :

$$E\left\{\exp\left[\frac{j2\pi i}{NT}(\beta_n - \beta_m)\right]\right\} = q^{|n-m|}$$

where q_τ is the value of the characteristic function of τ at frequency f_i. Since the minimum signal-to-noise ratio of N : 1 is obtained at $|q|^2 \rightarrow 0$, the result should give a good indication to the additive random sampling as well. Simulation results about the signal-to-noise ratios are tabulated in Table 3-1 and 3-2. From these tables, we can see that the noise power in the band depends on the sequence length N as well as the total input signal power. When N = 480, the signal-to-noise ratios from simulations are close to the estimated value, but when N = 1024, the signal-to-noise ratios drop slightly below the estimated value.

Table 3-1 : Signal-to-noise ratios (as defined by eqn (3-24a))of spectra reconstructed from jittered sampling

Input, V	length N	recovered amp., V	total power, V^2	noise, rms, mV	S/N, dB simulated	S/N, dB estimated
$\exp(2\pi \times 30t_n)$	480	1.41	1.99	65.7	26.6	26.8
	1024	1.42	2.02	46.7	29.6	30
$2\cos(2\pi \times 200\, t_n)$	480	2.00	4.00	97.0	26.3	26.8
$2\cos(2\pi \times 600\, t_n)$	1024	2.05	4.20	67.1	29.7	30
$\cos(2\pi \times 30\, t_n) + \sin(2\pi \times 200t_n) + 2\cos(2\pi \times 400\, t_n)$	480	1.17, 1.18, 2.18	7.51	127.3	26.6	26.8
$\cos(2\pi \times 30\, t_n) + \sin(2\pi \times 400t_n) + 2\cos(2\pi \times 800\, t_n)$	1024	0,89, 0.96, 1.92	5.38	84.9	28.7	30

Table 3-2 : Signal-to-noise ratios (as defined by eqn (3-24a)) of spectra reconstructed from additive random sampling

Input, V	length N	recovered amp., V	total power, V^2	noise, rms, mV	S/N, dB simulated	S/N, dB estimated
$\exp(2\pi \times 30t_n)$	480	1.42	2.02	61.2	27.3	26.8
	1024	1.42	2.02	42.6	30.4	30
$2\cos(2\pi \times 200\, t_n)$	480	1.96	3.84	86.4	27.1	26.8
$2\cos(2\pi \times 600\, t_n)$	1024	2.05	4.20	67.1	29.7	30
$\cos(2\pi \times 30\, t_n) + \sin(2\pi \times 200t_n) + 2\cos(2\pi \times 400\, t_n)$	480	1.07, 0.97, 2.12	6.58	126.8	26.1	26.8
$\cos(2\pi \times 30\, t_n) + \sin(2\pi \times 400t_n) + 2\cos(2\pi \times 800\, t_n)$	1024	0.93, 1.08, 2.03	6.92	84.7	29.8	30

From tables 3-1 and 3-2, we can also see that the total signal power to noise power is a constant at a particular length N. This implies that the individual signal-to-noise ratio will be lower when the input has more than 1 input frequency components. Taking the last row of table 3-1 as an example, the individual signal-to-noise ratios are 20.5dB, 20.5dB and 26.4dB while the total signal-to-noise ratio is 28.2dB. If there are m frequency components of equal power, the individual signal-to-noise ratio will be $P_S/m : P_n$, where Ps and Pn are the total signal power and the r.m.s. noise power respectively. Hence

$$S/N = 10\log_{10}\frac{P_s}{mP_n} = 10\log_{10}\frac{P_s}{P_n} - 10\log_{10} m$$

$$= total\ S/N - 10\log_{10} m \qquad \textbf{(3-25)}$$

If m = 3, the individual S/N should drop by 4.8dB from the total S/N. By simulation, a signal x(t) = cos(2πx30t) + cos(2πx50t) + cos(2πx80t) is sampled by jittered sampling for 480 points. From eqn (3-24), the estimated total signal-to-noise ratio is 26.8 dB. The simulation results in a total signal-to-noise ratio of 26.4 dB and the

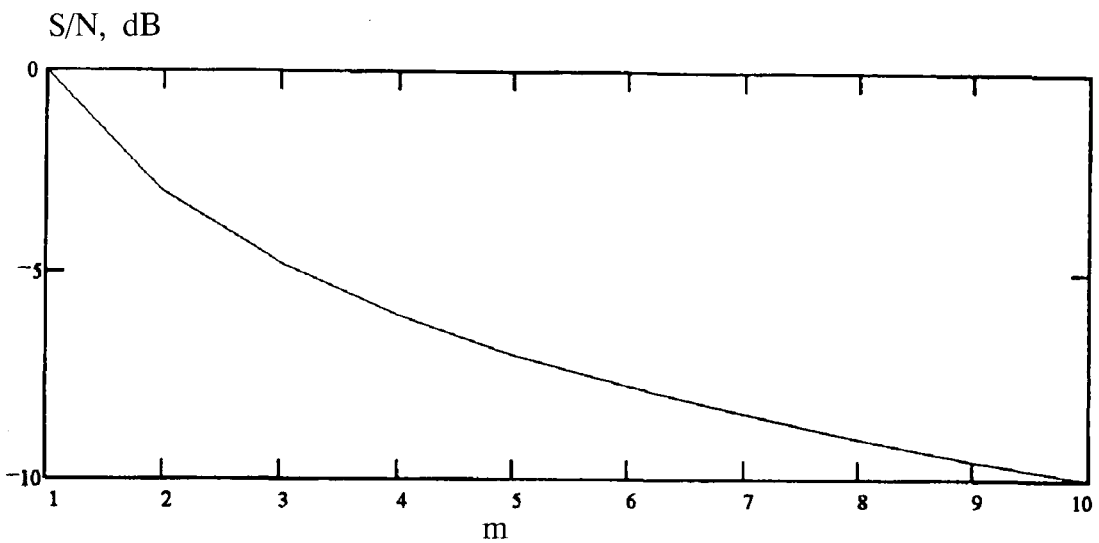


Fig. 3-9 Drop in signal-to-noise ratio (dB) with the increasing number of frequency components (m).

individual signal-to-noise ratios are 21.5, 21.6 and 21.8 dB. Taking the median value, the drop is 26.4 - 21.6 = 4.8 dB, which matches the predicted value. Since the individual signal-to-noise ratio will decrease as the number of frequency components increases, random sampling is suitable for an input signal with only a few frequency components in order to maintain reasonable individual signal-to-noise ratios.

3.3.2.2 Noise and Bandwidth : To compute a band of N frequency components, eqn(3-17) or (3-17a) may be used with k = 0,1,2,...,N-1. When a higher band of another N frequency components are to be evaluated, the same equation will be used, but in this case k = N,N + 1,N + 2,..., 2N-1. If a sinusoidal signal is an input signal located in this band, the noise analysis will be exactly the same as stated in section 3.3.2 and the estimated signal-to-noise ratio will also be given by eqn(3-24). Theoretically, the signal-to-noise ratio in a band of N components remains the same even if the frequency indices increase, which implies that the background noise is independent of the bandwidth. Table 3-3 shows the signal-to-noise ratios of different input signals,

Table 3-3 : Signal-to-noise ratios in different frequency bands

Input signal in V N = 480	recovered power V^2	Noise in mV, (S/N in dB), with k = 0-479	480-959	960-1439	1440-1919
jitter, cos (2π x 30t)	0.98	48.01 (26.2)	61.47 (24.1)	64.64 (23.7)	64.05 (23.8)
jitter, cos (2π x 530t)	0.96	52.27 (25.5)	55.16 (25.0)	62.80 (23.9)	64.57 (23.6)
jitter, cos (2π x 200t) + sin(2π x 800t) + cos (2π x 1200t)	2.95	103.0 (24.4)	101.2 (24.6)	104.2 (24.3)	109.3 (23.9)
a.r.s., cos (2π x 30t)	1	43.26 (27.3)	68.20 (23.3)	61.26 (24.3)	66.06 (23.6)
a.r.s., cos (2π x 530t)	1.05	59.20 (24.8)	51.56 (26.0)	68.04 (23.6)	63.53 (24.2)
a.r.s., cos (2π x 200t) + sin(2π x 800t) + cos (2π x 1200t)	2.40	99.35 (23.9)	87.13 (25.0)	96.08 (24.1)	104.9 (23.4)

all of which have a sequence length N = 480. From this table, noise levels are evaluated up to 4 times of N. We can see that the signal-to-noise ratios do not vary more than 3 dB from band to band.

3.3.3 Word-length and Bandwidth : In theory, there is no fold-over frequency or replicas in the frequency spectrum when random sampling is applied. This is true if the sampling times can be represented with infinite word-length in a system. Unfortunately, no practical digital systems have infinite word-length and a finite word-length set a limit to the useful bandwidth of the reconstructed spectrum. It will be shown that the resolution of the word-length is in effect the sampling frequency of the system.

Assume that the frequency spectrum is evaluated in the normalized case as in eqn (3-17a), which is :

$$X(k) = \frac{2}{N}\sum_{n=0}^{N-1} x(t_n)\exp(-j2\pi k t_n), \qquad k = 1,2,3,\ldots \tag{3-17a}$$

and t_n is represented by d digits past the decimal point, i.e. t_n is of the format 0.yyyy ... to d places. When k = 10^d , we have in the kernel $\exp(-j2\pi\times 10^d \times 0.yyyy...) = \exp(-j2\pi\times yyyy...) = 1$ because the term inside the bracket becomes a whole multiple of 2π. Since exp (0) = exp(-j2nπ) = 1, this is the 'zero' of the function and we have made a complete revolution along the unit circle in the complex plane at k = 10^d . It can be checked by letting k = $10^d + 1$, then

$$\exp(-j2\pi\times(10^d+1)\times 0.yyyy...) = \exp(-j2\pi\times yyyy...)\exp(-j2\pi\times 0.yyyy...)$$
$$= \exp(-j2\pi\times 0.yyyy...)$$

which is the same as at k = 1. The case is exactly the same as in a regularly sampled sequence of length N when the frequency index k reaches N + 1. Hence it is obvious

that the fold-over frequency is at $k = 10^d/2$. Consider the following example. Let the sequence length $N = 480$ and the sampling times are recorded to 3 decimal places. Then the fold-over frequency should be at $k = 1000/2 = 500$. Fig. 3-10 shows the reconstructed spectrum sampled by jittered sampling and the corresponding fold-over frequency. In all digital systems, numbers are expressed in binary digits. Suppose the timing is expressed in a normalized format (the value of the mantissa is between 0 and 1) by floating-point number with a b-bit mantissa and a positive x-bit exponent, then the "period" of the spectrum will be 2^{b-x} Hz and the fold-over frequency is at 2^{b-x-1} Hz. Fig. 3-11 shows the useful bandwidth of a representation of timing by fixed-point number when the fractional part varies from 8 bits to 20 bits.

3.3.4 Practical Considerations : As a summary, let us consider a numerical example. Suppose a bandwidth of 4 GHz is to be covered by a random sampling scheme with a frequency resolution of 0.5 MHz, then the total sampling period $= 10^{-6}/0.5 = 2$ μs. If we divide the whole band into 4 bands of 1 GHz each, then in each band we need $10^9/(0.5 \times 10^{-6}) = 2{,}000$ frequency indices. We may choose to take 2,000 samples in 2 μs, then the mean sampling period $= 2\,\mu s / 2000 = 1$ ns, or the mean sampling frequency is 1 GHz (as compared to 8 GHz by regular sampling). The expected value of the signal-to-noise ratio = 33 dB. For the whole band we need 8,000 points; hence the sampling time should be accurate up to 5 decimal places or 14 bits after the decimal point. The computational load for each band is $2{,}000^2 = 4 \times 10^6$, and for the whole band 1.6×10^7. (Suppose regular sampling is performed, $2^{14} = 16{,}384$ points of data will be sampled. By FFT, the computational complexity is $2^{14} \times 14 = 229{,}376 \approx 2.29 \times 10^5$.) If a signal -to-noise ratio having an expected value of 30 dB and a frequency resolution of 1 MHz are acceptable, we may sample the signal for 1,000

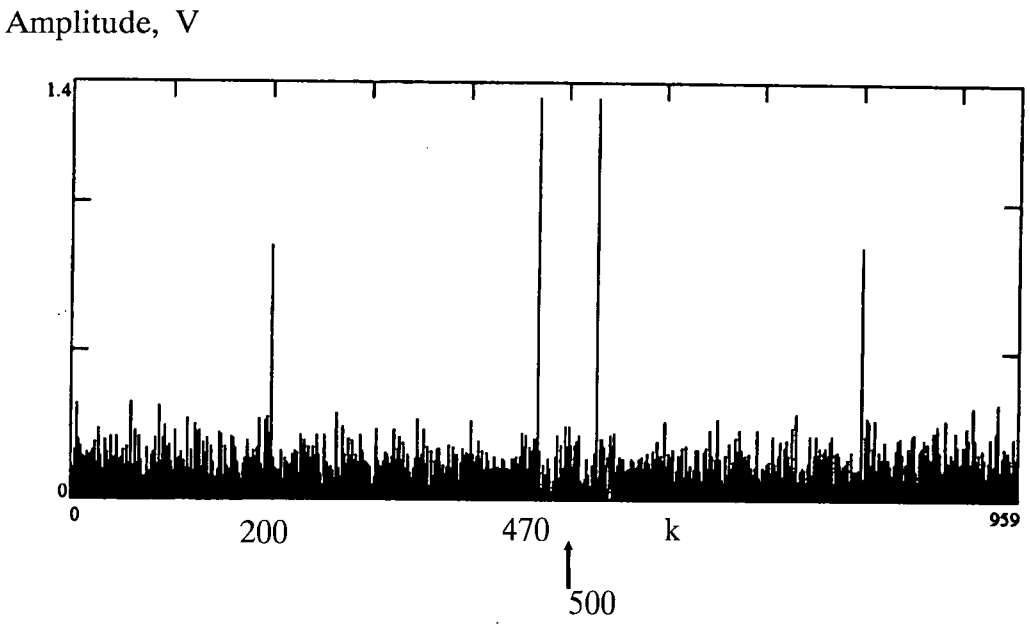


Fig. 3-10 An amplitude spectrum from a randomly sampled sequence of N = 480 and sampling times expressed to 3 decimal places. Note that the fold-over frequency occurs at k = 500.

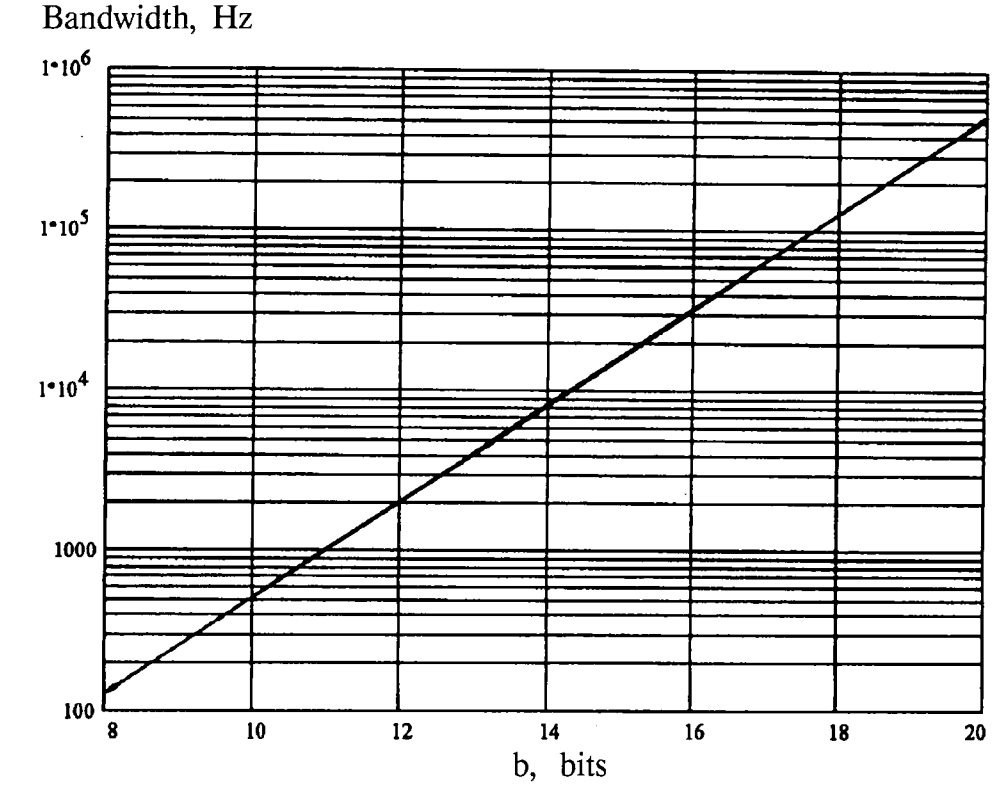


Fig. 3-11 Useful bandwidth (Hz) and the number of bits representing the fractional part of the sampling times.

points in 2μs which gives a mean sampling frequency of 500 MHz. The computational load becomes $4 \times 1{,}000^2 = 4 \times 10^6$. The sampling time needs to be accurate up to 4 decimal places or 13 bits.

Random sampling is an alias-free, sub-Nyquist sampling; therefore, it is most suitable to be adopted to any application that can benefit from this characteristic. Since the background noise level increases with the number of frequency components, the input signal to be sampled randomly should contain only a few sinusoids. Taking all these into consideration, this sampling scheme has been adopted to build measurement equipment, such as oscilloscope or frequency meter, where sub-Nyquist sampling enables the use of slower, thus less expensive, hardware[2,28] and the input signals are usually not too complex. Section 3.3.5 provides several typical examples.

In a random sampling scheme, the randomness of the random variable τ can be expressed by σ/μ, where σ is the variance of τ and μ is the mean sampling period. If this ratio is too small, the sampling approaches a regular sampling so that alias may occur. If, however, this ratio is too large, the error in the amplitude recovered will also be large. Usually, when σ/μ is around 30%, the result is satisfactory.

From section 3.3.2, we see that the signal-to-noise ratio depends on the input power and the sequence length. It is found that the longer the sequence, the better the signal-to-noise ratio. Unfortunately, a longer sequence length means much more computation since the complexity of the computation is N^2. When the sampling is implemented by a digital system, the finite word-length of the system will set an upper bound to the bandwidth of the spectrum.

3.3.5 Typical Applications in Instrumentation

3.3.5.1 Spectrum Analyzer : A spectrum analyzer displays in the frequency domain the components of a stationary signal. When the input signal is to be processed digitally, it has to be sampled and quantized before its spectrum can be computed. If the signal is sampled regularly, an anti-alias filter must be inserted before the sampling process. Since the bandwidth of this filter should be sufficiently wide to accommodate a broad spectrum, the order of the filter would be high so as to achieve a reasonably narrow transition band. A wide bandwidth also leads to a stringent requirement in the speed of the hardware since the sampling frequency of the analyzer must be at least twice the bandwidth of the anti-alias filter.

Random sampling is appropriate to be applied in spectral estimation. First of all the anti-alias filter can be eliminated. The operation speed of the sampling hardware can be reduced as the average sampling frequency can be made lower than the Nyquist frequency. However, as discussed in section 3.3.3, the resolution in the word-length of the sampling time must be high enough to support a wide spectrum. Apart from gaining advantages in hardware, random sampling also provides a flexible bandwidth. For a regularly sampled sequence of N points, only N/2 useful frequency components can be recovered. With random sampling, the spectrum can be computed to any index larger than N provided that the resolution in the word-length of the sampling time is high enough and the background noise level is acceptable. The appearance of a reconstructed spectrum would look similar to the spectrum shown in Fig. 3-8 (b). Another numerical example can be found in section 3.3.4, which is equivalent to an analysis of a spectrum of 4-GHz bandwidth with 0.5-MHz frequency

resolution. If a power spectrum is required, the auto-correlation method described in section 6.4 may be adopted.

3.3.5.2 Meters : For detecting a simple signal comprising only a few sinusoids which are located within a narrow frequency band, random sampling, especially hybrid additive random sampling (section 5.5), is a suitable choice since the amount of computation incurred is small. One such example is the frequency meter, which is usually for measuring monotonic input. There are instruments whose operation is based on non-linear signal conversion, e.g. true r.m.s. voltmeters and wattmeters. When a non-linear converter is involved, it can be quite troublesome if high precision and broad bandwidth are simultaneously required. For this type of instruments, after the conversion process and sampling, the quantity to be measured is made proportional to the d.c. term of the resulting spectrum [2]. If the converted signal is regularly sampled, aliases produced could be very close to the d.c. value, making subsequent filtering very difficult. When the signal is sampled randomly, aliases will become broadband background noise, which can be filtered easily. In this scheme, only the d.c. value of the spectrum is to be computed; thus averaging, not multiplication, is required for calculating the desired quantity. The computation is therefore very simple.

3.3.5.3 Oscilloscope and Filter : Instruments may be so designed that its output signal is reconstructed in the time domain from a periodic input sampled randomly or irregularly. One example is the sampling oscilloscope which utilizes sub-Nyquist sampling to improve its high-frequency performance [30]. Fig. 3-12 illustrates the working principle of the oscilloscope. From the timing diagram, it can be seen that each sampling pulse turns on the sampling circuit for a very short interval. With

reference to the input waveform, the positions of the sampling pulses shift horizontally such that different parts of the input cycle are sampled in each period. In this way,the oscilloscope plots the output waveform point by point, using as many as 1,000 samples to reconstruct the original input. The sampling frequency may be as low as one-hundredth of the input signal frequency. If the input frequency is 1,000 MHz, the required bandwidth of the amplifier would be only 10 MHz, which is much easier to implement.

Non-recursive filtering in the spatial domain can also be realized with a randomly sampled signal [28] . Recalling that with regular sampling, a filtered signal $y(n)$ can be expressed as $y(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k)$, where $x(k)$ and $h(k)$ are the input sequence and the impulse response of the filter respectively, a filtered output $y(t_k)$ can similarly be calculated by weighted summation of a randomized input sequence $x(t_k)$ except that the coefficients $h(t_n\text{-}t_k)$ vary with both n and k, which implies that the values of the filter coefficients are different for computing each output point. Fig. 3-13 shows the timing diagram for computing y_1 to y_5 by the equation $y_n = h_{n1}x_1 + h_{n2}x_2 + \ldots + h_{nk}x_k$, where $y_k = y(t_k)$, $x_k = x(t_k)$ and $h_{nk} = h(t_n\text{-}t_k)$ are the filter coefficients which are the sample values of the impulse response h(t). Note that the peak of h(t) is positioned to coincide in time with the corresponding output signal value. If the two sets of timing $\{t_n\}$ and $\{t_k\}$are fixed for all input signals and h(t) can be explicitly expressed in time t, h_{nk} can be calculated beforehand and stored in memory. The filter can then be implemented by switching the input sequence to a suitable set of filter coefficients for evaluating each output value.

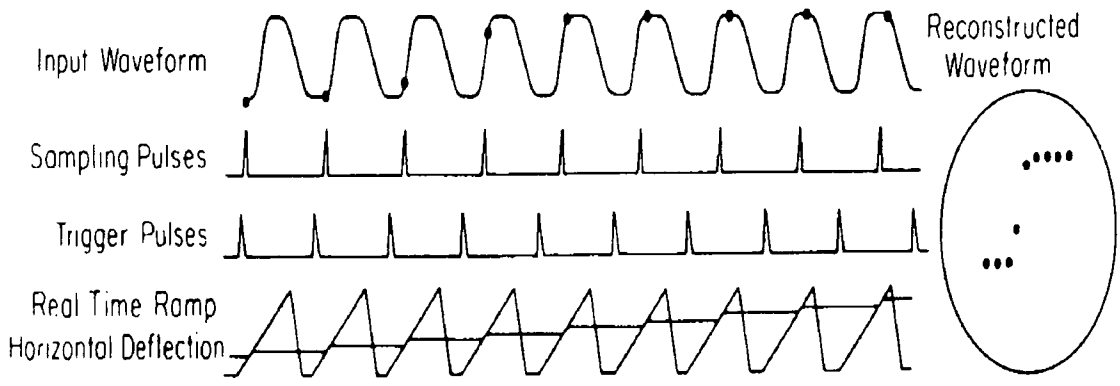


Fig. 3-12 Waveforms illustrating the operation of a sampling oscilloscope (From Helfrick and Cooper [49]).

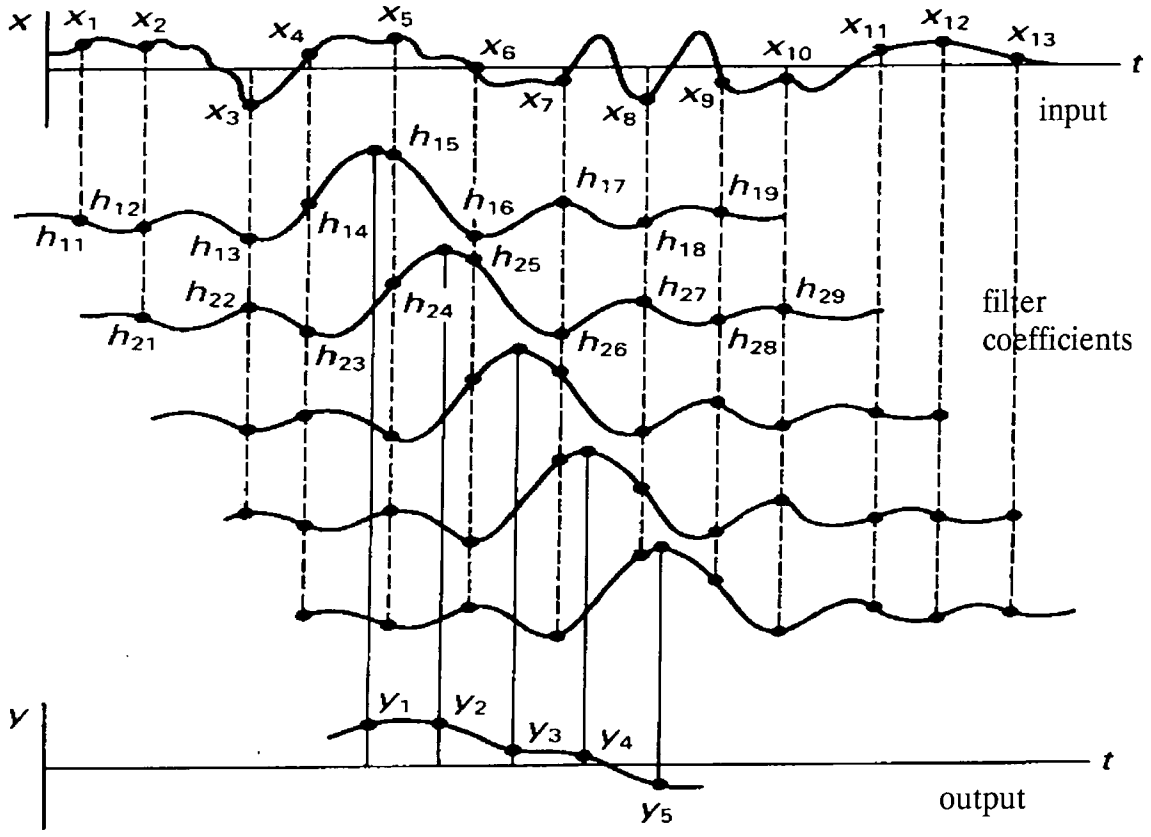


Fig. 3-13 Filtering of a signal x_k by a filter of impulse response h_{nk}. Five output data, y_1 to y_5, are shown (From Bilinskis and Mikelsons[28]).

3.4 Three-sampler System

Apart from random sampling, there is another sampling method that can achieve sub-Nyquist sampling by adopting a totally different approach. The input signal is regularly sampled by three sampling frequencies which are below the Nyquist rate[3,31]. Aliases occur in the three output spectra but the ambiguity can be resolved by the mathematical relationship between the outputs and the sampling frequencies. This method is suitable for sparsely populated spectra over an extremely wide bandwidth.

3.4.1 Principle of the System : To explain the principle of this method, it is assumed that the input signal contains only one frequency component. The results are then readily generalized for a signal containing multiple components. Consider that a signal f_x is input to a sampler and a low-pass filter, which yields an output f_{o1} as shown in Fig. 3-14. If $f_x < 0.5\ f_{s1}$, $f_{o1} = f_x$. If f_x is between $0.5\ f_{s1}$ and f_{s1}, the active element is filtered out but its image $f_{s1} - f_x$ is below $0.5\ f_x$ and thus appears in the output. When $f_x = f_s$, the output $f_{o1} = 0$. This relationship between f_{o1} and f_x is plotted in Fig. 3-15. It is clear from this figure that the relationship is ambiguous and cannot yield a unique value of f_x for a given value of f_{o1}. It is possible, however, to describe their relationship by the following pair of expressions :

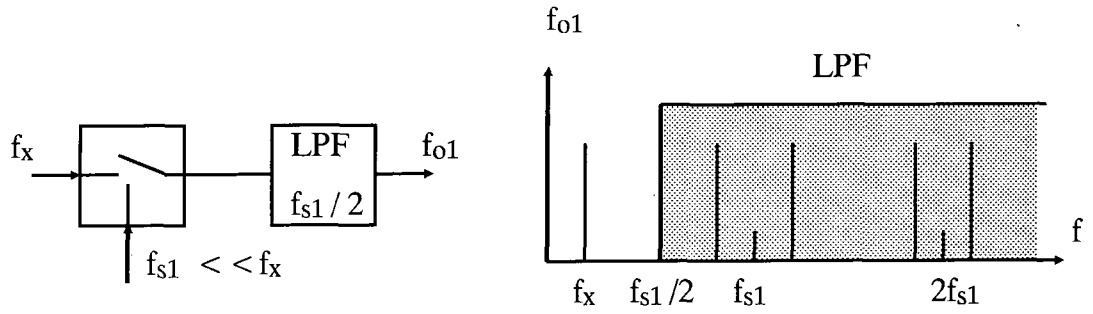


Fig.3-14 The frequency spectrum from a sampler and a low-pass filter. The aliases under the shaded area will appear in the output according to the relationship shown in Fig. 3-15.

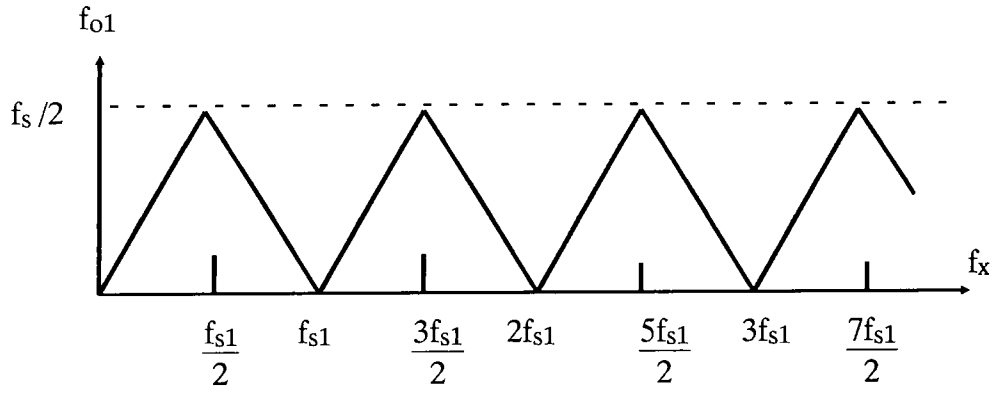


Fig. 3-15 Relationship in frequency between the input and output signals

$$f_{o1} = f_x - Kf_{s1}, \qquad Kf_{s1} \le f_x \le (K+0.5)f_{s1}$$
$$f_{o1} = Kf_{s1} - f_x, \qquad (K+0.5)f_{s1} \le f_x \le (K+1)f_{s1} \qquad \textbf{(3-26)}$$

where K is a positive integer. A possible means of removing the ambiguous relationship between f_x and f_{o1} is to use a second sampling with a different sampling frequency $f_{s2} << f_x$ and $f_{s2} - f_{s1} = \Delta f_s$. The cut-off frequency of the corresponding LPF is at 0.5 f_{s2}. This results in a different output frequency f_{o2} which can also be expressed as :

$$f_{o2} = f_x - Lf_{s2}, \qquad Lf_{s2} \le f_x \le (L+0.5)f_{s2}$$
$$f_{o2} = Lf_{s2} - f_x, \qquad (L+0.5)f_{s2} \le f_x \le (L+1)f_{s2} \qquad \textbf{(3-27)}$$

where L is a positive integer. We can choose either K=L or K=L+1 to relate eqn(3-26) and eqn(3-27). Fig. 3-16 is the frequency characteristic of this joint sub-Nyquist sampler. Close examination of Fig. 3-16 reveals that a point of symmetry exists in the frequency patterns generated by the two graphs where

$$FR = nf_{s1} = (n-0.5)f_{s2}$$

After this point, the frequency pattern is a mirror image of the pattern before FR. Therefore, the highest frequency of the components must be less than the value of FR. This operational frequency range is given by :

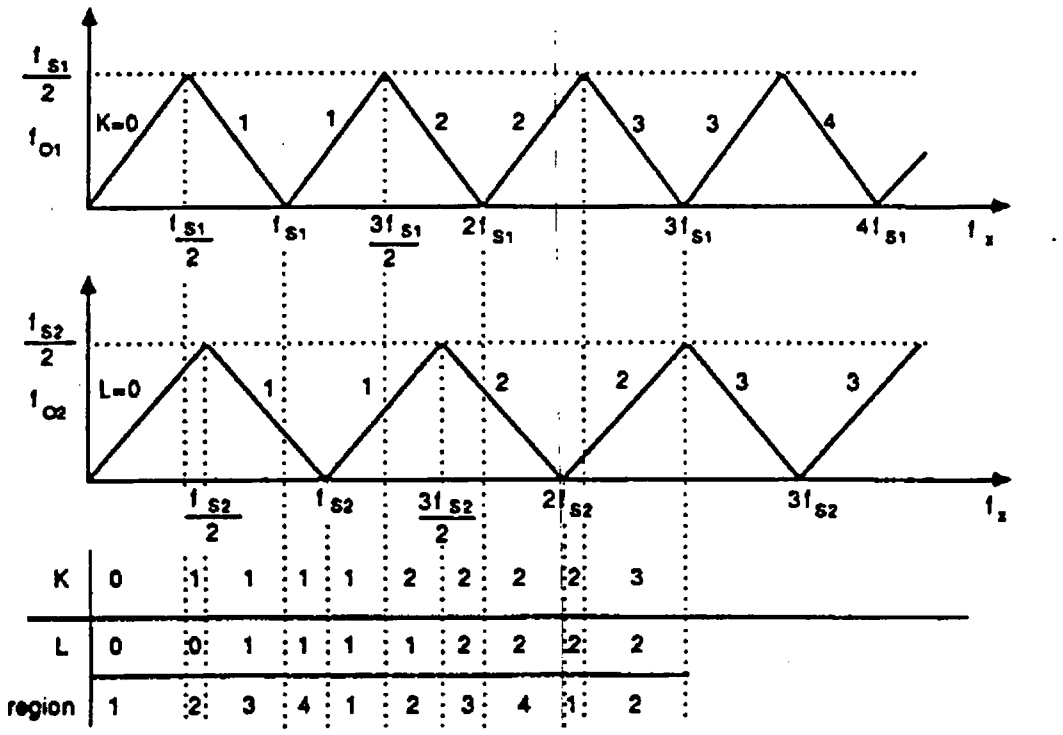


Fig. 3-16 Frequency characteristic of the joint sub-Nyquist sampler

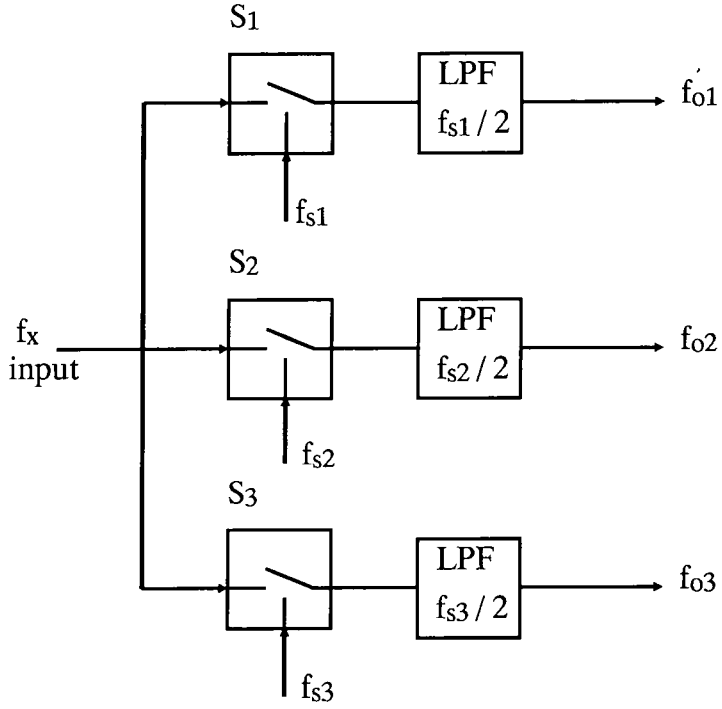


Fig. 3-17 Sub-Nyquist sampler with three sampling frequencies

$$FR = \frac{f_{s1} f_{s2}}{2\Delta f_s} \approx \frac{{f_{s2}}^2}{2\Delta f_s} \qquad \textbf{(3-28)}$$

The graphs in Fig. 3-16 and the corresponding table show that there are four distinguishable regions labelled 1 to 4. The corresponding equations are :

Region 1	$f_{o1} = f_x - Kf_{s1}$,	$f_{o2} = f_x - Lf_{s2}$	$K = L$
Region 2	$f_{o1} = Kf_{s1} - f_x$,	$f_{o2} = f_x - Lf_{s2}$	$K = L+1$
Region 3	$f_{o1} = Kf_{s1} - f_x$,	$f_{o2} = Lf_{s2} - f_x$	$K = L$
Region 4	$f_{o1} = f_x - Kf_{s1}$,	$f_{o2} = Lf_{s2} - f_x$	$K = L$ **(3-29)**

Each of the above four pairs of equations can be solved separately for f_x and three possible input frequencies f_{x1},f_{x2} and f_{x3} can be obtained :

$$f_{x1} = \frac{|f_{o2} f_{s1} - f_{o1} f_{s2}|}{\Delta f_s}$$

$$f_{x2} = \frac{f_{o2} f_{s1} + f_{o1} f_{s2}}{\Delta f_s}$$

$$f_{x3} = \frac{f_{s1} f_{s2}}{\Delta f_s} - f_{x2} \qquad \textbf{(3-30)}$$

Unique solution for f_x , however, cannot be obtained from eqn(3-30). To solve this problem, a third sub-Nyquist sampler with sampling frequency $f_{s3} = f_{s2} + \Delta f_s$ and a LPF with cut-off frequency at 0.5 f_3 is added. Similarly, samplers 2 and 3 yield three possible inputs f_{x4}, f_{x5} and f_{x6} :

$$f_{x4} = \frac{|f_{o3} f_{s2} - f_{o2} f_{s3}|}{\Delta f_s}$$

$$f_{x5} = \frac{f_{o3} f_{s2} + f_{o2} f_{s3}}{\Delta f_s}$$

$$f_{x6} = \frac{f_{s2}f_{s3}}{\Delta f_s} - f_{x5} \tag{3-31}$$

The desired input frequency is arrived at by selecting the pair of values derived from the two sets which are equal. Any final ambiguous answer which may result if any two false frequency ambiguities having equal values, thus giving equal numbers in the two sets, can be resolved by a simple validity test :

$$f_x \pm f_{oi} = Nf_{si}$$

where i = 1,2 and 3 and N is a positive integer. An invalid answer will give a non-integer N for at least one sampling frequency while a real f_x will pass such tests.

From eqn(3-28) the useful frequency range of the S_1 and S_2 pair is $FR = \frac{f_{s1}\, f_{s2}}{2\Delta f_s}$. Similarly, the useful frequency range of the S_2 and S_3 sampling pair is $FR' = \frac{f_{s2}\, f_{s3}}{2\Delta f_s} < FR$ since $f_3 > f_1$. Hence the working frequency range of the whole system is determined the smaller value of the two, i.e. FR.

To illustrate the principle of the system, let the sampling frequencies f_{s1} = 9 MHz, f_{s2} = 10 MHz and f_{s3} = 11 MHz. Then the frequency difference Δf_s = 1MHz and the working frequency range is FR = 9x10/2 = 45 MHz (FR' = 55 MHz). The results of computing three incoming signals having different frequencies are tabulated in Table 3-4.

3.4.2 Error Analysis : The characteristic graph of Fig. 3-16 which is used to derive eqn (3-30) is subject to frequency errors in the presence of noise in the input and sampling signals. The effect of frequency jitter associated with the periodic sampling impulse is much more pronounced than that for the original signal. These errors have been analyzed in detail by Sarhadi [32]. Another source of error is the limited resolution

of the FFT. If the number of points in the FFT is N, then any active element must be represented by spreading over the coefficients of the FFT but assuming that most of the energy is concentrated in the single nearest coefficient. The maximum frequency error in the aliased spectral lines is given in [33] by :

$$\Delta f_{omax} = \left| \frac{f_s}{2N} \right| \tag{3-32}$$

where f_s is the sampling frequency. From Fig. 3-16, region 4 , the output is $f_x = \frac{fo1fs2 + fo2fs1}{\Delta f_s}$. Errors on the aliased lines lead to an error in the calculated output of:

$$\Delta f_x = \left| \frac{\Delta fo1 fs2 + \Delta fo2 fs1}{\Delta f_s} \right| \tag{3-33}$$

Since $f_{s2} \approx f_{s1} = f_s$, eqn (3-33) can be written as :

$$\Delta f_x = \left| \frac{2 f_s \, \Delta fo}{\Delta f_s} \right| \tag{3-34}$$

The maximum errors are given by substituting Δf_{omax} in eqn (3-32) into eqn (3-34):

$$\Delta f_{xmax} = \left| \frac{f_s^{\,2}}{N \Delta f_s} \right|$$

Therefore, if Δf_{xmax} is to be less than $f_s/4$ giving an error band less than $f_s/2$, then $\frac{f_s}{4} > \frac{f_s^{\,2}}{N \Delta f_s}$. By rearranging the expression :

Table 3-4 : Examples of the three-sampler system (From Underhill et. al. [3])

input signal	Samplers S_1 and S_2			Samplers S_2 and S_3			Calculated frequency
f_x	f_{x1}	f_{x2}	fx3	fx4	f_{x5}	fx6	fx
34	16 †	>45	34	34	54 †	>55	34
24	6 †	>45	34	24	>55	46 †	24
7.25	7.25	42.25*	>45	7.25	>55	42.25*	7.25

*False ambiguity of f_x in the relevant pair of samplers

† Real ambiguity of f_x in the relevant pair of samplers

$$N > \frac{4f_s}{\Delta f_s} \tag{3-35}$$

It should be noted that the ratio $f_s/\Delta f_s$ is a magnification factor of the error which is a consequence of the sub-Nyquist sampling.

3.4.3 Illustrative Example : Assume that it is required to analyze signals in a broad bandwidth up to 10 GHz with a frequency resolution of 0.5 MHz and a population of the active elements much less than 1%. With conventional techniques some 20,000 filters would be required to achieve the same resolution over the frequency range of interest.

To design the sub-Nyquist system capable of operating over the 10 GHz range, eqn (3-28) can be used. If the nominal sampling rate is chosen to be 1 MHz, then Δf must be 50 Hz to give FR $= \frac{(10^6)^2}{2\times 50} = 10$ GHz. The number of frequency indices N within f_s is given by eqn (3-35) :

$$N = \frac{4\times 10^6}{50} = 80{,}000$$

The length of the sequence of samples must then be 2N. The computational load for the sub-Nyquist system is thus given by :

$$W = 3 \text{ x } 2N \log_2 (2N)$$

For N = 131,072 $= 2^{17}$:

$$W = 3 \text{ x } 2 \text{ x } 2^{17} \log_2 (2 \text{ x } 2^{17}) = 14{,}155{,}776$$

3.5 Concluding Remarks

From the discussions above, it is clear that random sampling and the three-sampler method are two different approaches to achieve sub-Nyquist sampling. One motivation of implementing sub-Nyquist sampling is to adopt slower hardware, which can be regarded as a means to lower hardware cost or to push the operation speeds of the components to their limit. The sequence length of the samples, in contrast to regular sampling which observes Shannon's theorem, can be curtailed. This curtailment may save memory storage and even computation in some cases. Anti-alias filters, which must appear in regular sampling, may also be eliminated or kept to a minimum.

Random sampling turns aliases to a broadband background noise, which is similar in nature to the leakage produced in the regular sampling. Since the noise level tends to increase with the total power of the input signal, this approach is suitable to detect a broadband signal with a sparse population of frequency components. Also, if among these components there is one which is much weaker in power than the others, this component could be covered up by the background noise and would not be recognized. As variation in timing is deliberately introduced into the sampling grid, noise or jitter inherent to the sampling hardware may be ignored if it is small, or it may be incorporated into the random variable if the jitter is measurable. With the introduction of randomness into the timing grid, symmetry is lost and the computational complexity is N^2. In chapters 4 and 5, we shall discuss how the computational load may be alleviated. Although this method in principle is alias free, when realizing the sampling in a digital system, its finite word-length sets an upper

bound to the bandwidth. The timing must, therefore, be kept in a word-length long enough for a broad spectrum.

The three-sampler system adopts regular sampling which enables the use of the FFT to compute the frequency spectrum. Having a complexity similar to the FFT, its computational load is lighter than that of random sampling. With regular sampling, noise in the timing will generate error in the output, and this error is further magnified by this sub-Nyquist sampling. This method is also suitable to detect a broadband signal with a sparse population of frequency components. Here, three low-pass filters are required. Baier and Fürst [34] suggest a system with only two samplers and no filters. This system can detect aliases and recover frequency components to a bandwidth only half of the sampling frequency. Sub-Nyquist sampling is thus not achieved.

CHAPTER 4

PARALLEL ADDITIVE RANDOM SAMPLING

4.1 Introduction

In estimating the spectrum of a band-limited signal, discrete Fourier Transform (DFT), or more often, Fast Fourier Transform (FFT) is applied to the samples of the signal taken at regular intervals. Shannon's sampling theorem states that the sampling frequency must be at least twice that of the sampled signal, otherwise alias will occur. Random sampling has been suggested to overcome the above limitation. Two methods of random sampling, namely jittered sampling and additive random sampling (a.r.s.), have been discussed in Chapter 3. When computing the frequency components from a randomly sampled sequence, however, different "random" exponential terms must be multiplied to each data point since the symmetry in regular sampling is now relinquished, giving rise to an N^2 complexity. If a higher speed of the computation is desired, some sort of regularity must be inserted into the sampling process, but to such an extent that the anti-alias property is still maintained. A sampler of m parallel blocks, each operating with an additive random sampling sequence of p points, is found to satisfy this requirement [35]. This scheme, which interlaces several a.r.s. sequences to form an anti-alias sequence, exploits trigonometrical symmetry to reduce up to 87% of the multiplications required in computing the first band of frequency components. The whole process, from sampling to computation, can be implemented by a multiprocessor system [36]. With a variable threshold, a relatively weak signal can also be recovered.

4.2 Timing of the sampling sequence

The structure of the parallel sampler is shown in Fig. 4-1, where there are m sampling blocks working together. To have the maximum saving in computation, m should be a multiple of 4. Each block takes p samples according to the additive random sampling $t_n = t_{n-1} + \tau_n$ where τ is a uniformly distributed random variable of finite variance [37]. The sequence length of the resultant signal $x(t_n)$ will be N, where N = m.p. The starting time of the first sequence $t_0 = 0$, and each subsequent sequence starts at $t_i = t_0 + i.T$ where T = 1/N of the sampling duration. With such an arrangement, $t_1 = t_0 + T$ and $t_2 = t_0 + 2T$, etc. (see Fig. 4-1). In general

$$t_{i,m+q} = t_{i,m} + q.T \tag{4-1}$$

where i = 0,1,2...p-1, and q = 0,1,2,...m-1.

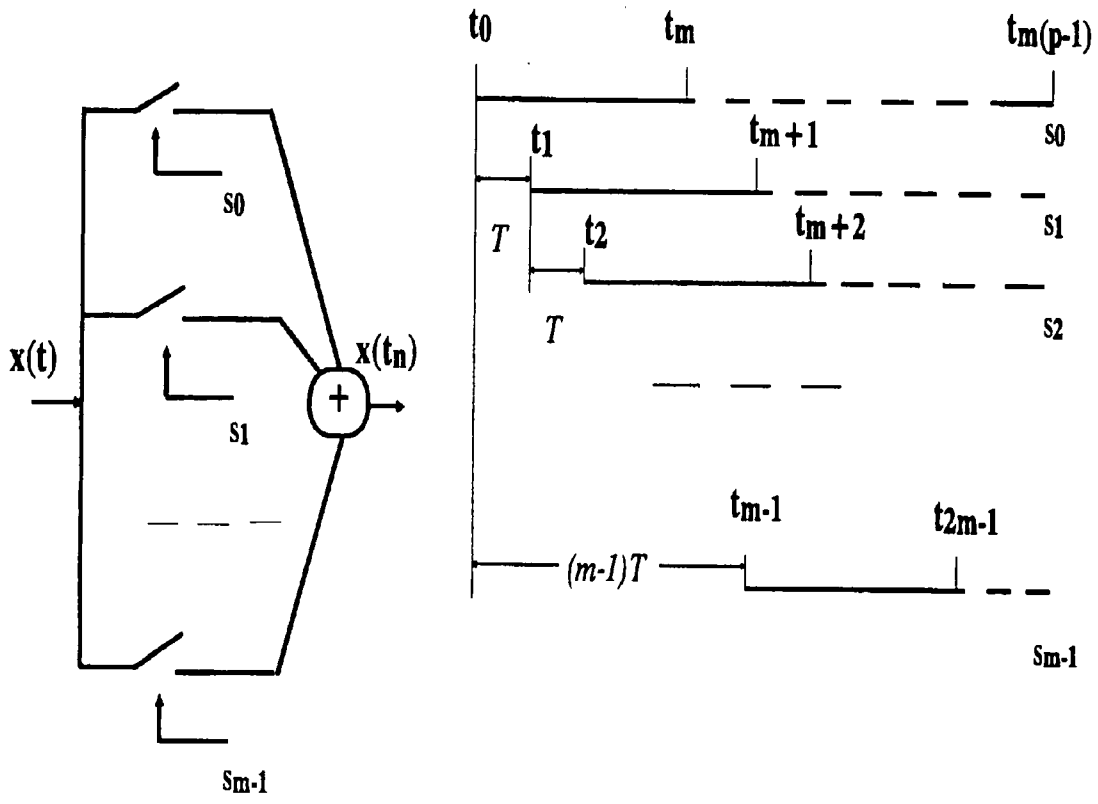


Fig. 4-1 Timing diagram of the parallel additive random sampling

4.3 Anti-alias property

Fig. 4-2 shows the frequency spectrum of a 1024-point sequence formed by interlacing 32 a.r.s. sequences of 32 points each. The interlace is equivalent to linearly convolving a 32-point a.r.s. sequence with a train of 32 unit pulses of uniform separation T. Hence in the frequency spectrum, we expect to see a unit pulse at the origin followed by groups of noise with relatively large amplitude centering at 1024.i, where i is an integer. If the spectrum is, however, examined closely, it can be seen that the phase angles of these groups are randomly distributed. Sampling in the time domain with this sequence is equivalent to convolving with its frequency spectrum in the frequency domain. By the randomness of the phase angles, the effects of these

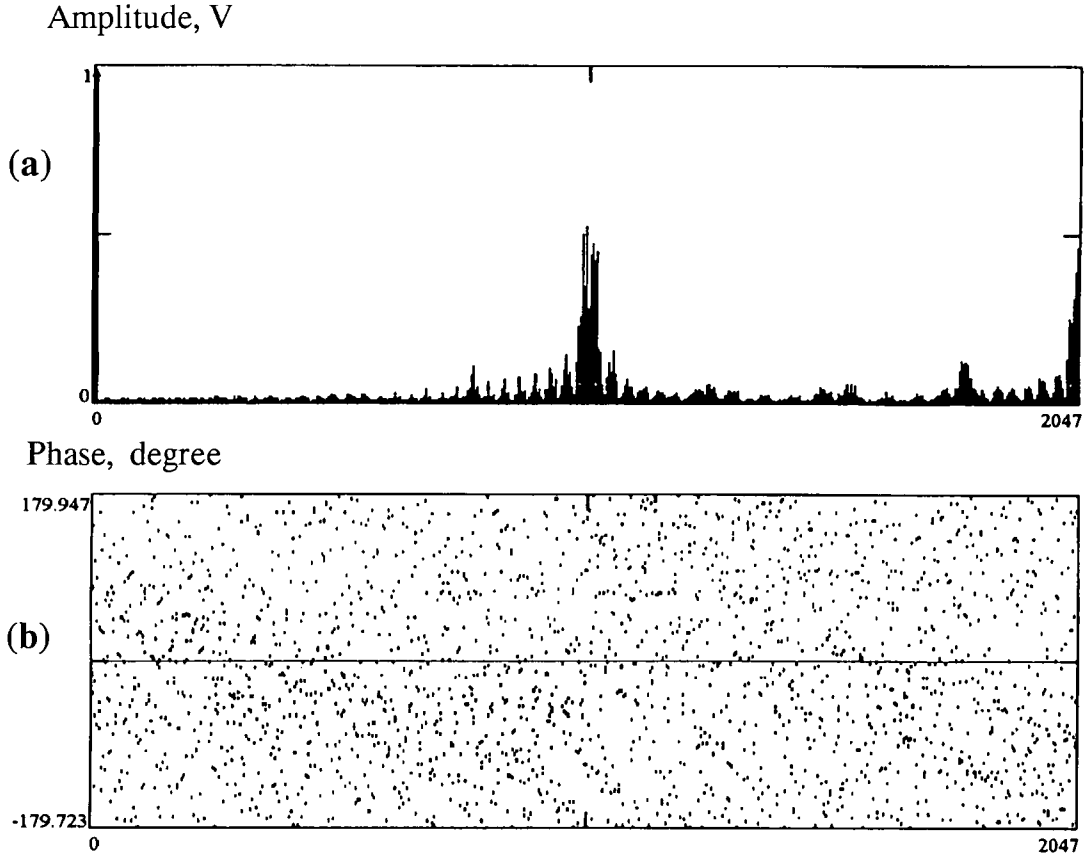


Fig. 4-2 Frequency spectrum of a 1024-point (m = 32, p = 32) parallel random sampling sequence:

(a) amplitude (b) phase in degree

residual groups tend to average out, leaving behind "bursts of noise" rather than sharp aliases.

When a signal of frequency f is sampled by this sequence, the reconstructed spectrum will show a strong component at f and groups of noise with much weaker amplitude at $i.f_S \pm f$, where i is an integer. Similar to the case of a normal a.r.s., as long as the band of signal to be recovered is not congested with a large number of frequencies, the signal can be easily identified. Fig.4-3 shows different spectra of a signal, $1\cos(2\pi.160t + \pi) + 0.3 \sin(2\pi. 400t) + 1.5 \cos(2\pi.1{,}100t)$ V, sampled regularly and by parallel random sampling. The ratio of the standard deviation of τ to the mean of the sampling period is about 10%. If sampled uniformly, the Nyquist limit is at 512Hz. With the parallel a.r.s., the 1,100Hz can clearly be identified. Its anti-alias property is thus demonstrated.

As mentioned in section 3.2, the randomness of a sampling scheme can be measured by a ratio σ/μ, where σ is the standard deviation of the sampling periods and μ their mean. (The sampling periods are defined as $t_n - t_{n-1}$, where t_n represents the sampling time at interval n.) Basically the parallel a.r.s. is a random sampling scheme with the original random variable "diluted" by a number of regular sampling intervals having a value equal to μ,which are inserted deliberately to the timing grid. In Fig. 4-4 there are five amplitude spectra of a signal sampled for 1024 points by a.r.s. and parallel a.r.s. Table 4-1 summarizes the results of the simulation. We can see that the signal-to-noise ratios depend on the sequence length N (estimated value = 20 log $\sqrt{N}$ =30 dB) rather than m or p of the parallel a.r.s. As m increases, regularity increases (or randomness decreases), the "bursts of noise" are "compressed" into

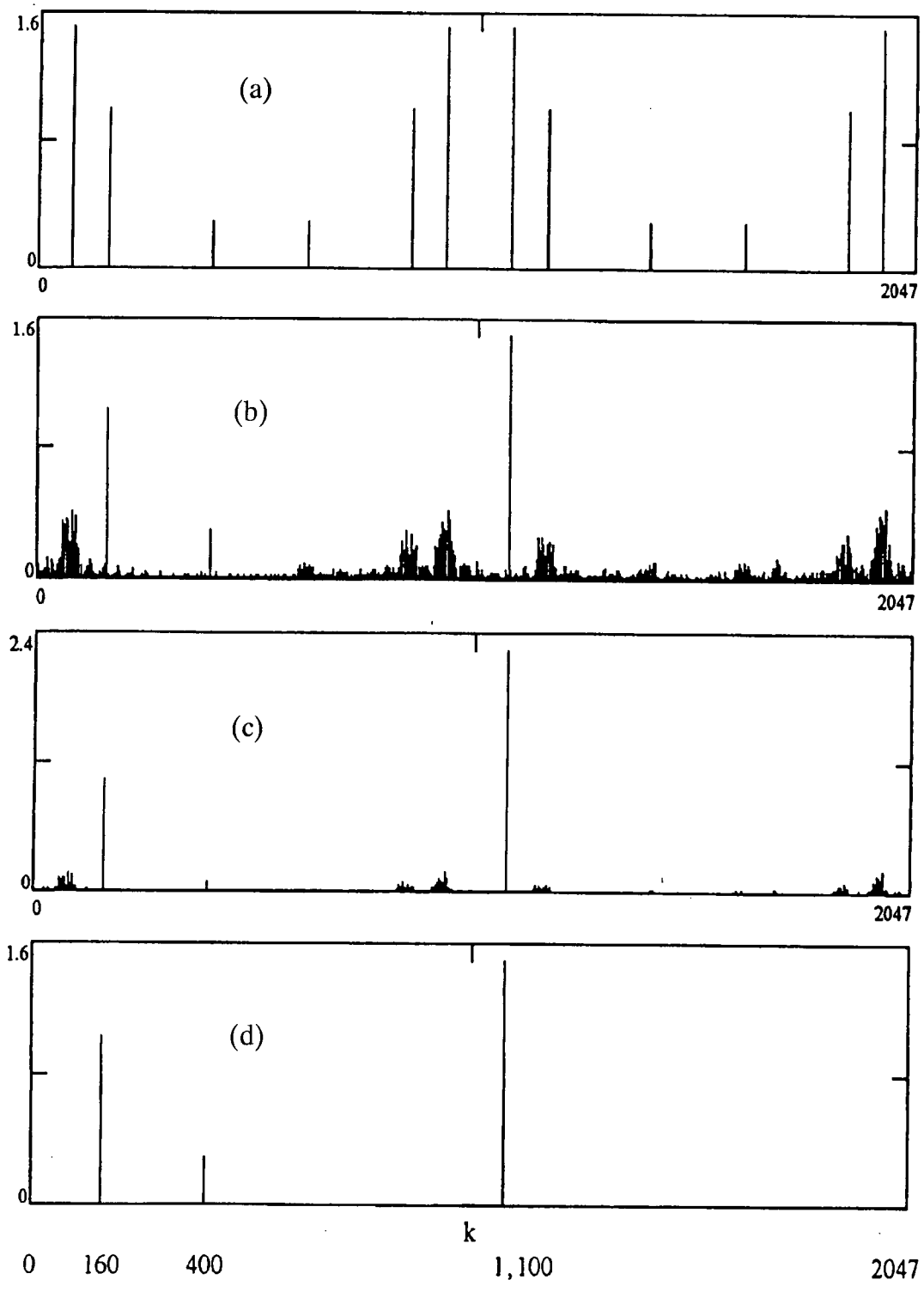


Fig.4-3 Spectra of a signal sampled for 1024 points :

(a) amplitude spectrum, by regular sampling,

(b) amplitude spectrum, by parallel a.r.s.,m = 32, p = 32,

(c) power spectrum of (b),

(d) spectrum from (b) by variable threshold method.

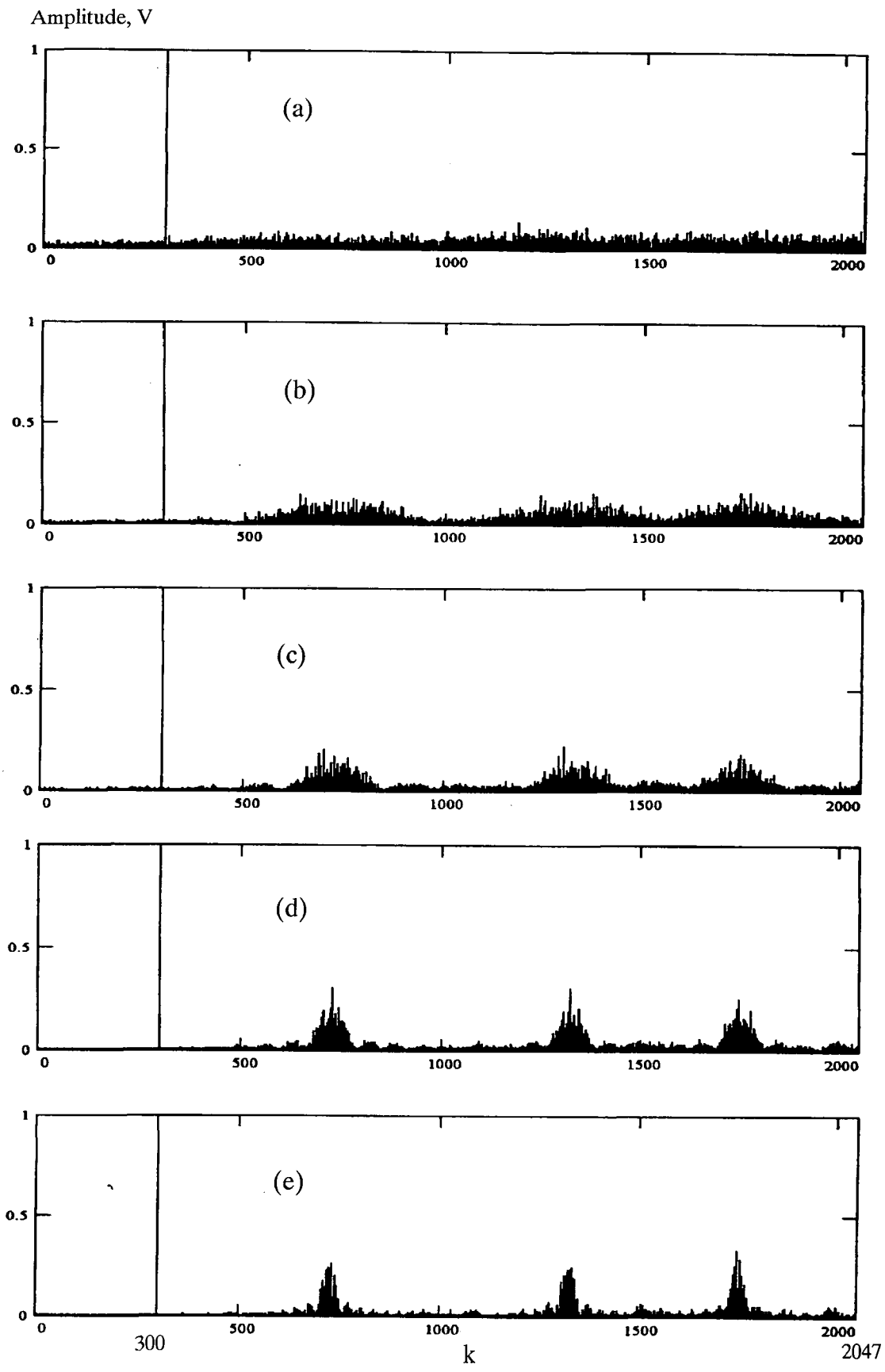


Fig. 4-4 Noise in the spectra of a signal sampled for 1024 points:

(a) by a.r.s. (b) by parallel a.r.s., m = 4, p = 256 (c) by parallel a.r.s., m = 8, p = 128 (d) by parallel a.r.s., m = 16, p = 64 (e) by parallel a.r.s. m = 32, p = 32.

Table 4-1 : Noise in the spectrum and the randomness of the sampling sequence

Sampling method	Signal, V at 300 Hz	Noise within k < 1024 r.m.s. , mV	peak, mV	S/N, dB	$\frac{\sigma}{\mu}$, %
a.r.s., 1024 points	0.999	29.0	94	34.4	29.9
parallel a.r.s., 4 x 256	0.994	31.1	147	31.9	15.6
parallel a.r.s., 8 x 128	0.997	31.2	201	31.9	11.0
parallel a.r.s., 16 x 64	1.001	31.3	301	31.9	7.65
parallel a.r.s., 32 x 32	0.998	31.3	262	31.8	6.18

narrower frequency windows and their peak values rises. If the ratio $\frac{\sigma}{\mu} \rightarrow 0$, we return to regular sampling so that the "bursts of noise" will become sharp aliases.

4.4 Computational Algorithm and Realization

4.4.1 Symmetry in Timing : The real and imaginary parts of the estimates of the frequency components (except the d.c.) are given respectively by [37]:

$$X_r(k) = \frac{2}{N}\sum_{n=0}^{N-1} x(t_n)\cos(2\pi k f t_n)$$

$$X_i(k) = \frac{2}{N}\sum_{n=0}^{N-1} x(t_n)\sin(2\pi k f t_n) \qquad \textbf{(4-2)}$$

where k = 1,2,3... and f = 1/NT = 1 in the normalized case. Hence T = 1/N.

Using the timing specified by eqn (4-1), putting $x_n = x(t_n)$ and neglecting the scaling factor, eqn (4-2) can be written as :

$X(k) = y_{k0} + y_{k1} + y_{k2} + \ldots + y_{k(p-1)}$ where

$$\begin{pmatrix} y_{k0} \\ y_{k1} \\ y_{k2} \\ \cdots \\ y_{k(p-1)} \end{pmatrix} = L_k \left\{ \begin{pmatrix} x_0 & x_1 & \ldots & x_{m-1} \\ x_m & x_{m+1} & \ldots & x_{2m-1} \\ x_{2m} & x_{2m+1} & \ldots & x_{3m-1} \\ & \cdots & & \\ x_{(p-1)m} & x_{(p-1)m+1} & \ldots & x_{N-1} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \exp(j2\pi k/N) \\ \exp(j2\pi.2k/N) \\ \cdots \\ \exp(j2\pi.(m-1)k/N) \end{pmatrix} \right\}$$

$$\text{with} \quad L_k = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & exp(j2\pi kt_m) & 0 & \dots & 0 \\ 0 & 0 & exp(j2\pi kt_{2m}) & \dots & 0 \\ & & \cdots & & \\ 0 & 0 & 0 & \dots & exp(j2\pi kt_{(p-1)m}) \end{pmatrix} \tag{4-3}$$

a diagonal matrix for post-multiplication. For simplicity, eqn (4-3) may be represented by:

$$Y_k = L_k \{D.E_k\} = L_k \{M_k\}$$

When evaluating the base-band components, k = 0,1,2,...N-1. As x_n are real numbers and m is a multiple of 4, an analogy may be drawn between M_k and the roots of unity evenly distributed in the four quadrants of the complex plane. Taking N = 16 as an example, real$\{E_1\}$ = real$\{E_9\}$ and imag$\{E_1\}$ = -imag$\{E_9\}$. Similar symmetry properties can be found in E_7 and E_{15} with E_1. For E_1 and E_3, the real and imaginary parts are interchanged alternatively with appropriate changes in sign. In general, after M_i is evaluated, $M_{N/2-i}$, $M_{N/2+i}$, M_{N-i} and $M_{N/4-i}$ can be deduced. As M_0, $M_{N/4}$, $M_{N/2}$ and $M_{3N/4}$ do not require any actual multiplications, only 1/8 of these M_k need actual multiplications. The following describes the algorithm for computation in detail.

4.4.2 Computational Algorithm :

(Step 1) Evaluate M_0, $M_{N/4}$, $M_{N/2}$ and $M_{3N/4}$. Multiply the corresponding Li to form X(0), X(N/4), X(N/2) and X(3N/4).

(Step 2) For i = 1 to N/8, repeat the following :

(Step 2.a) Break M_i into a sum of vectors:

$$M_i = \begin{pmatrix} x_0 \\ x_m \\ \dots \\ x_{(p-1)m} \end{pmatrix} + \begin{pmatrix} x_1 \\ x_{m+1} \\ \dots \\ x_{(p-1)m+1} \end{pmatrix} e^{j2\pi i/N} + \dots + \begin{pmatrix} x_{m-1} \\ x_{2m-1} \\ \dots \\ x_{N-1} \end{pmatrix} e^{j2\pi i(m-1)/N}$$

$= b_{i0}+b_{i1}+\ldots+b_{i(m-1)}$

(Step 2.b) Defining M_{iv} and M_{id} as the sums of all even-index terms and all odd-index terms of b_{in} respectively:

$$M_{iv} = b_{i0} + b_{i2} + \ldots + b_{i(m-2)}$$

$$M_{id} = b_{i1} + b_{i3} + \ldots + b_{i(m-1)}$$

then $M_i = M_{iv} + M_{id}$ and $M_{N/2+i} = M_{iv} - M_{id}$.

From M_i obtain $M_{N-i} = \text{real}\{M_i\} - j.\text{imag}\{M_i\}$, and

from $M_{N/2+i}$ obtain $M_{N/2-i} = \text{real}\{M_{N/2+i}\} - j.\text{imag}\{M_{N/2+i}\}$.

Multiply the corresponding L_i to M_i to obtain X(i).

(Step 3) From b_{ir} evaluated in (Step 2.a), deduce $b_{(N/4-i),r}$ as follows:

for $r = 0$, $b_{(N/4-i),0} = b_{i0}$

for r = even, $\text{real}\{b_{(N/4-i),r}\} = (-1)^{r/2}.\text{real}\{b_{ir}\}$,

$$\text{imag}\{b_{(N/4-i),r}\} = (-1)^{r/2+1}.\text{imag}\{b_{ir}\},$$

for r = odd, $\text{real}\{b_{(N/4-i),r}\} = (-1)^{(r+3)/2}.\text{imag}\{b_{ir}\}$,

$$\text{imag}\{b_{(N/4-i),r}\} = (-1)^{(r+3)/2}.\text{real}\{b_{ir}\}.$$

Repeat (Step 2.b), taking $i = N/4-i$.

Fig. 4-5 shows a signal flow diagram of the algorithm with $N = 4x4 = 16$. If a higher band of N frequency components are to be evaluated, all the M_i in the base band can be re-used. Take $i = N + k$ as an example, where i is the frequency index in the next

band, substituting k by N + k in (3) and noting that exp(j2 π [N + k]/N) = exp(j2 π k/N), we get

$$Y_{N+k} = L_{N+k}\ M_k, \quad \text{where}$$

$$L_{N+k} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & exp(j2\pi[N+k]t_m) & 0 & \dots & 0 \\ 0 & 0 & exp(j2\pi[N+k]t_{2m}) & \dots & 0 \\ & & \dots & & \\ 0 & 0 & 0 & \dots & exp(j2\pi[N+k]t_{(p-1)m}) \end{pmatrix}$$

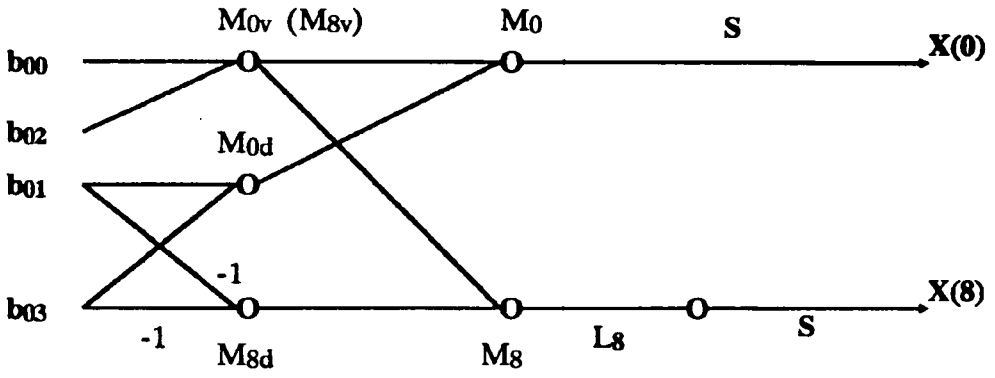
(4-4)

4.4.3 Saving in Multiplications : In general a complex multiplication is considered to comprise 4 real multiplications. In our case, since the input sequence is real, we define a complex multiplication to consist of 2 real multiplications. So when 2 complex numbers multiply each other, there are 4 real multiplications, i.e. 2 complex multiplications. With direct evaluation of N frequency indices of N sample points, there are N^2 multiplications. Applying the above algorithm, in the first quadrant, only N/8 of the points need to be multiplied by both E_i and L_i, the exponential vectors and the diagonal matrices for post-multiplication respectively. Considering a single point of full treatment, there are p(m-1) = N-p multiplications for E_i and 2(p-1) for L_i, summing up to N + p-2. The remaining N/8-1 points are to be multiplied by L_i, giving another 2(N/8-1)(p-1). In the other 3 quadrants, only L_i are required, hence there are altogether 6(N/4-1)(p-1) multiplications. For X(0), no multiplication is required. For X(N/4), X(N/2) and X(3N/4), only L_i are required, hence there are 6(p-1) multiplications. The total is :

$$\begin{aligned} tm &= N/8(N+p-2) + 2(N/8-1)(p-1) + 6(N/4-1)(p-1) + 6(p-1) \\ &= N^2/8 - 2N + p(15N/8 - 2) + 2 \end{aligned}$$

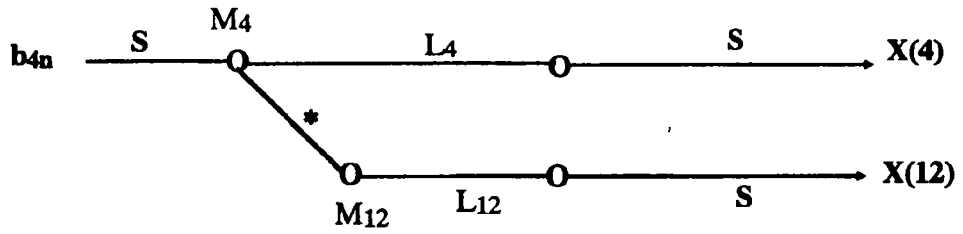
$N = 4 \times 4 = 16 \qquad D.E_i \rightarrow b_{in}$

$i = 0 \quad E_0 = 1$



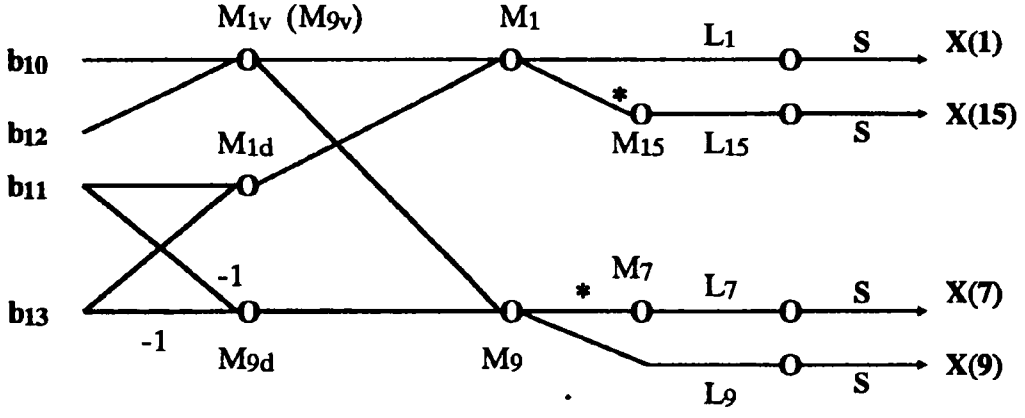
$b_{00} \rightarrow b_{40}$ $\qquad b_{01} \rightarrow j\, b_{41}$

$b_{02} \rightarrow -\, b_{42}$ $\qquad b_{03} \rightarrow j\, b_{43}$



* = complex conjugate $\qquad$ S = summation

$i = 1 \quad D.E_1 \rightarrow b_{1n}$



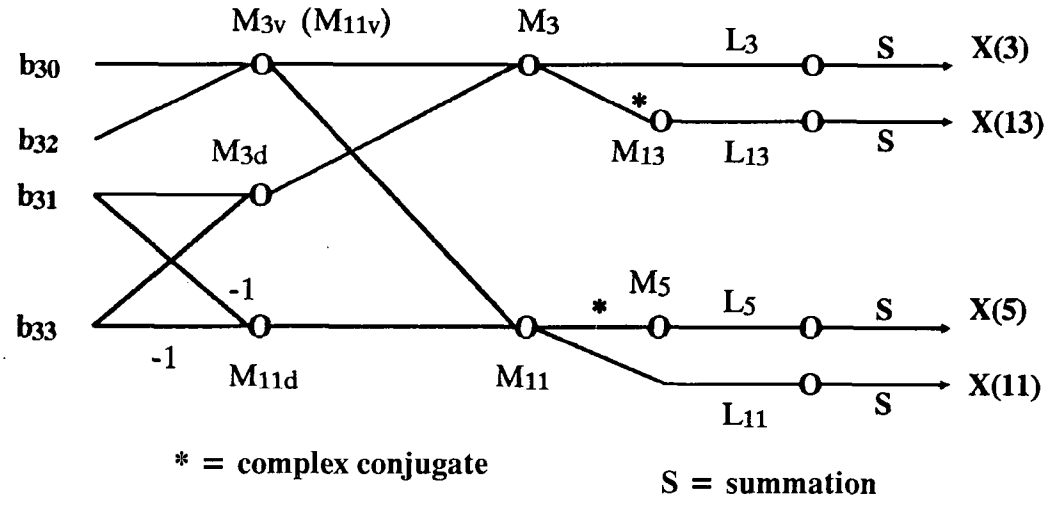
$b_{10} \rightarrow b_{30}$

$\text{real}\{b_{11}\} \rightarrow \text{imag}\{b_{31}\}$ $\qquad \text{imag}\{b_{11}\} \rightarrow \text{real}\{b_{31}\}$

$\text{real}\{b_{12}\} \rightarrow -\,\text{real}\{b_{32}\}$ $\qquad \text{imag}\{b_{12}\} \rightarrow \text{imag}\{b_{32}\}$

$\text{real}\{b_{13}\} \rightarrow -\,\text{imag}\{b_{33}\}$ $\qquad \text{imag}\{b_{13}\} \rightarrow -\,\text{real}\{b_{33}\}$

Fig. 4-5 (first part) : Signal flow diagram of the computational algorithm for N = 16.



$i = 2 \quad D.E_2 \rightarrow b_{2n}$

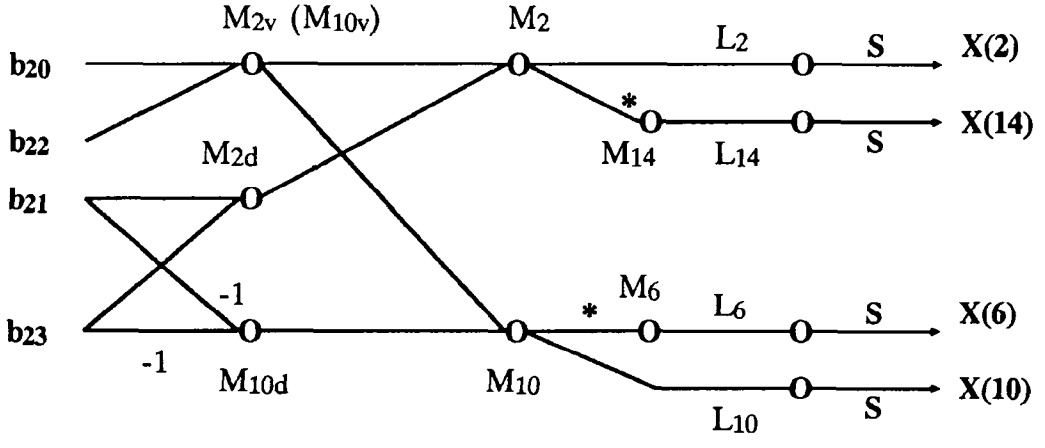


Fig. 4-5 (second part) : Signal flow diagram of the computational algorithm for N = 16 .

The fraction saving: $sv = 1 - tm/N^2$. Since $N = pm$,

$$sv = 7/8 + 2/N - 15p/8N + 2p/N^2 - 2/N^2 \qquad \textbf{(4-5a)}$$

$$\text{Since } N >> 2, \quad sv = 7/8 - 15/8m \qquad \textbf{(4-5b)}$$

With N = 1024, eqn (4-5b) gives an error smaller than 0.2%. Let N = 1024, Table 4-2 shows the saving obtained from eqn (4-5a). When m = 4, each a.r.s. sequence is comparatively long and the saving is also low. When m rises to 128, the saving seems to be very high, but with only 8 points in each a.r.s. sequence, the residues of the aliases

will also be very strong. The case with m = p = $\sqrt{N}$ is a satisfactory compromise between the two extremes. The percentage savings with p = m are tabulated in Table 4-3.

For any higher band of N frequency components, only the post- multiplications of L_i are required. Therefore there are 2p multiplications per point. The total for N components tm = 2pN = $2N^2/m$, since p = N/m. Therefore, the fraction saving:

$$sv = 1 - 2/m \qquad \textbf{(4-6)}$$

4.5 Implementation with Multiprocessor System

4.5.1 Sampling and Computing : The parallel random sampling is naturally parallel and fits perfectly into a concurrent structure. From sampling to computation, each process can be separated into several identical units which can perform

Table 4-2 : Percentage saving for p.m = N = 1024

m	p	saving %	m	p	saving %
4	256	40.87	32	32	81.84
8	128	64.28	64	16	84.77
16	64	75.99	128	8	86.32

Table 4-3 : Percentage saving for p = m

p	N	tm	N^2	saving %
16	256	15,330	65,536	76.61
20	400	34,162	160,000	78.65
32	1024	190,420	1,048,576	81.84
64	4096	4096	16,777,216	84.62

independently. Fig. 4-1 shows clearly that m samplers can be employed to read in data concurrently with a fixed delay in time relative to each other. Each sampler is thus working at only 1/m of the effective sampling rate, which offers the possibility of using slower and cheaper hardware. Apart from the delay, all samplers are the same in every aspect.

After sampling, the sequence obtained will be used to evaluate the frequency components of the spectrum. From the computational algorithm shown in Fig. 4-5, the process is neatly divided into N/8 identical blocks. An example of realization with four transputers is shown in Fig. 4-6. Since no exchange of intermediate data is required between these blocks, they are truly independent of each other. Running concurrently, the whole process can be speeded up by a maximum factor of N/8. If N is 1024, the speed-up factor may go up to 128.

4.5.2. Simulation with Transputers : The computational algorithm of the parallel a.r.s. is simulated in a computer system with an add-on transputer board of B008. On the board there are five IMS T800 transputers, four of which contain 4 Mbytes RAM and the fifth one contains 8 Mbytes RAM. T800 transputer is a 32-bit CMOS micro-computer with a 64-bit floating point unit and graphic support [38]. It has 4 Kbytes on-chip RAM, a configurable memory interface and 4 communication links. By establishing communication between these links, a concurrent system can be constructed from a collection of transputers operating simultaneously. The T800 links support the standard operating speed of 10 Mbits/sec, but they can also operate at 5 or 20 Mbits/sec. Each link can transfer data bi-directionally up to 2.35 Mbytes/sec. IMS C004 programmable link switch is used to provide full switching capabilities between 32 INMOS links.

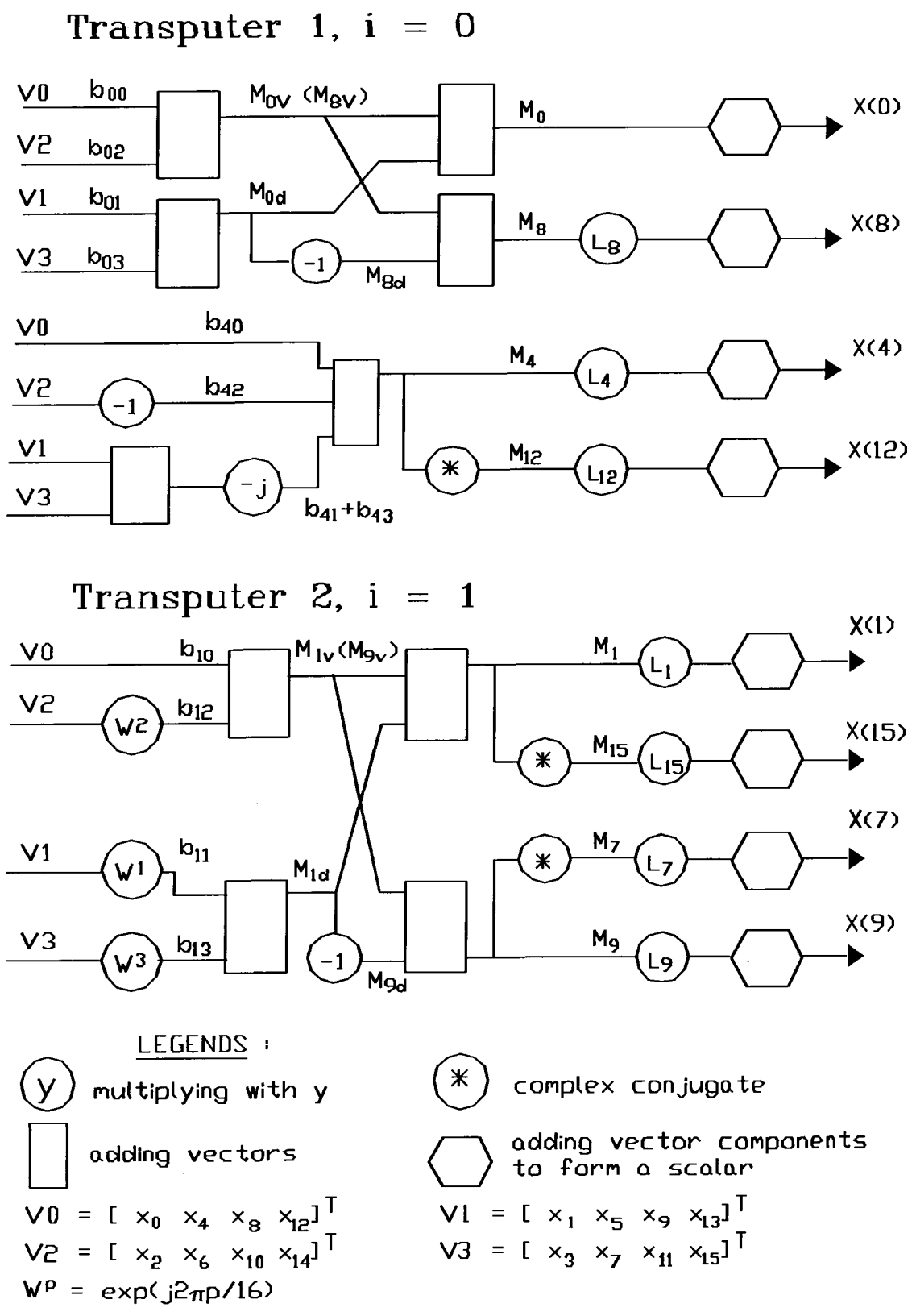


Fig.4-6 (first part) Realization of the computational algorithm shown in Fig. 4-5 with four transputers. The first transputer computes {X(0), X(8), X(4), X(12)} and the second transputer computes {X(1), X(15), X(7), X(9)}. Note that there is no data communication between the transputers.

The data flows of the third and fourth transputers are shown in Fig. 4-6 (second part).

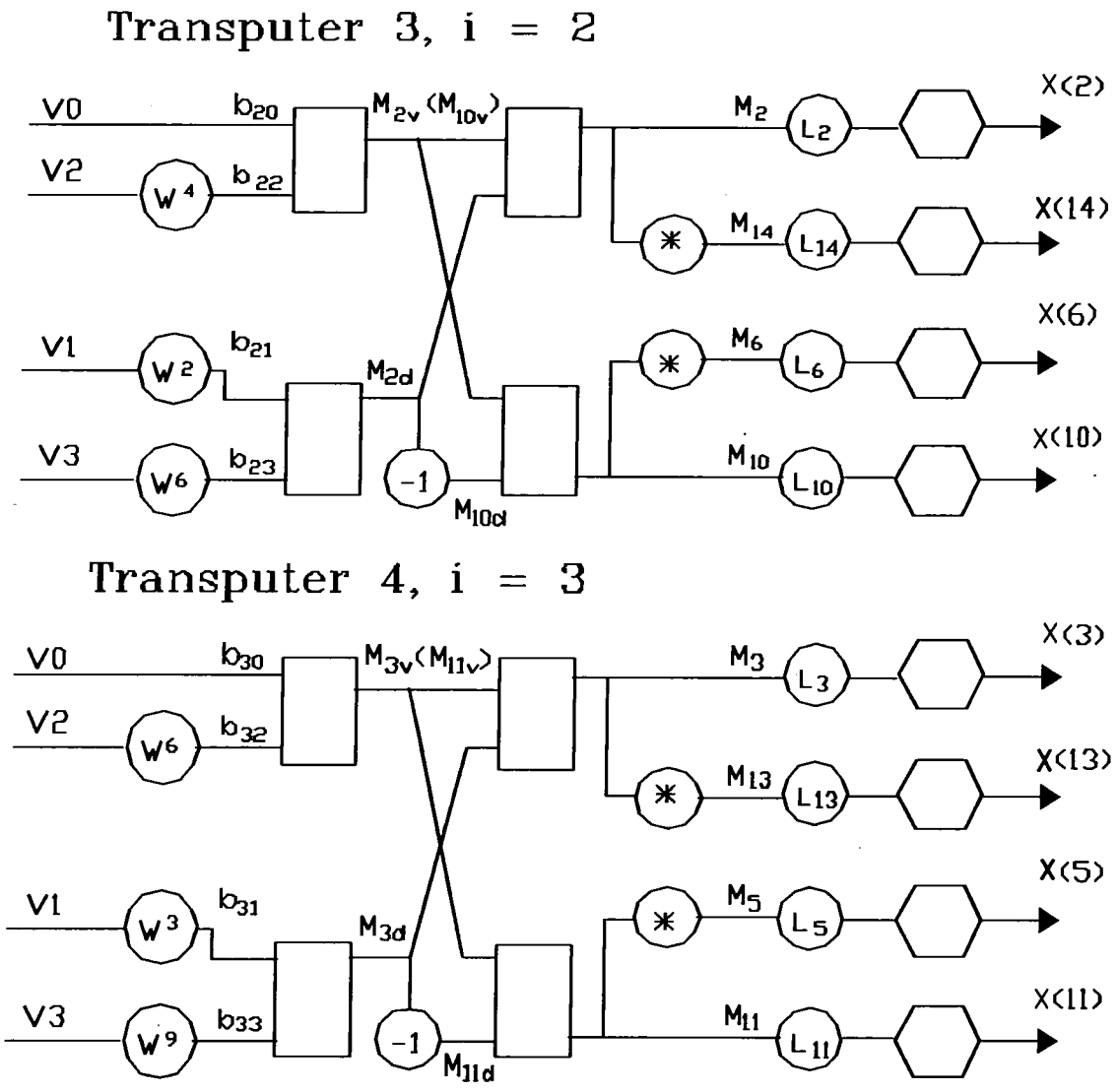


Fig.4-6 (second part) Realization of the computational algorithm shown in Fig. 4-5 with four transputers. The third transputer computes {X(2), X(14), X(6), X(10)} and the fourth transputer computes {X(3), X(13), X(5), X(11)}.

The ideal configuration for running the algorithm on 5 transputers is a star connection with the root at the centre (see Fig. 4-7(a)). Unfortunately two of the links of the root are reserved for special purposes; hence the configuration shown in Fig. 4.7(b) is adopted. Since there is no exchange of data between the transputers during computation, the communication links are used only at the initial stage when the root sends input data to the other transputers and at the final stage when the root collects the results from the others.

The simulation program is written in 3L Parallel C of INMOS Ltd [39]. Parallel C is based on the idea of communicating sequential processes on the transputer systems. In a user program, there is a collection of one or more concurrently executing tasks, each of which has its own region of memory for code and data, a vector of input ports and a vector of output ports. Each processor can support any number of tasks limited only by the availability of memory. Tasks placed on the same processor can have any number of interconnecting channels since there is no external communication involved. Tasks placed on different processors, however, can only communicate where physical wires connect the links of the processors. Each logical connection between two tasks placed on different processors is assigned the exclusive use of one of the physical links connecting the processors. The number of connections between tasks is therefore limited by the number of hardware links each processor possesses. Apart from the hardwire list specifying the physical connections of the communication channels, there is a softwire list which describes where the tasks, including the "Iserver" and "Filter", are to be placed on the transputer network.

Three programs are written to verify the saving in computation when using the parallel a.r.s. A sequence of 1024 points are sampled by the parallel a.r.s. with $m = 32$ and $p = 32$. The first program is a direct evaluation of the sequence and the second is an evaluation using the algorithm described in section 4.4.2. These two programs run on a single transputer. The third program is also an evaluation using the same algorithm, but it runs concurrently on 4 transputers (excluding the root, which is mainly for data communication). The results of the simulations are tabulated in Table 4-4, which can be compared to the saving predicted by eqn (4-5) and (4-6). When evaluating the components in the base band, the percentage saving, from eqn (4-5) or

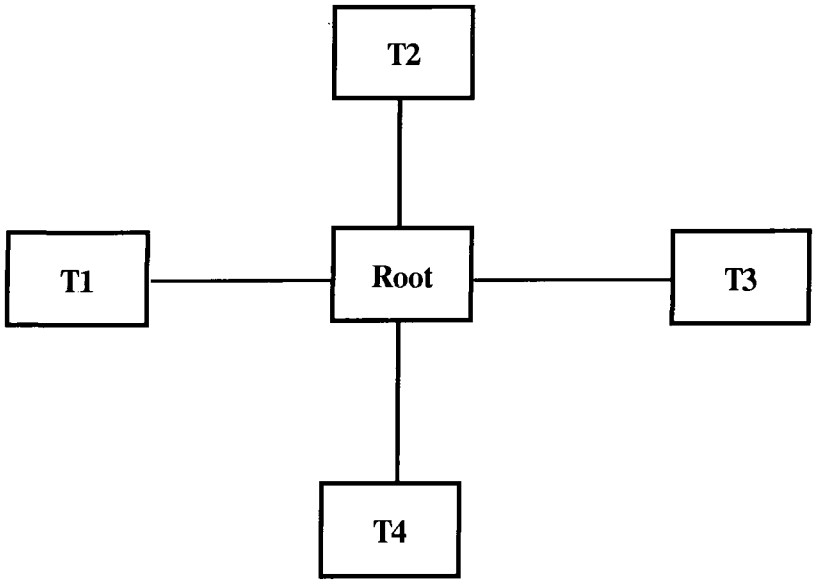


Fig. 4-7(a) Ideal configuration for 5 transputers computing parallel a.r.s. concurrently.

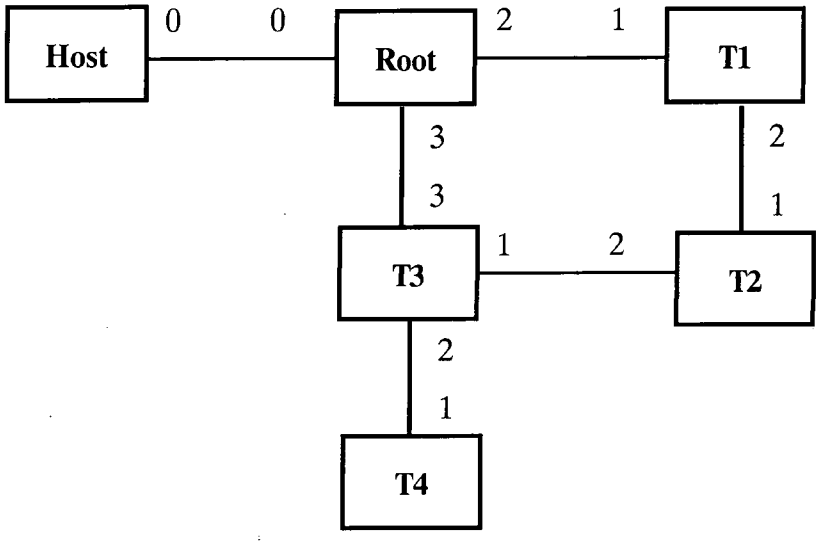


Fig. 4-7(b) Practical hardwire configuration for the 5 transputers

Table 4-2, is 82% for m = 32, p = 32, which means that 18 calculations is performed per hundred calculations required by the direct evaluation. Hence the speed-up factor is 100/18 = 5.6. From Table 4-4, the simulation result gives a speed-up of 5.9. It can be seen that with 4 transputers operating concurrently, the calculation is further speeded up by a factor of 4. From eqn (4-6), the saving for calculating the next band of 1024 frequency components is 15/16, which means that 1 calculation is required per 16 in the direct evaluation. Hence the speed-up factor should be 16. From Table 4-4, the simulation result gives 12, which is lower than expected. The difference should come from the extra time consumed by the program overheads, such as fetching from the memory those vectors M_k stated in eqn (4-4) for the multiplications with L_{N+k}. As a whole, the simulation results are close to the expected theoretical values.

4.5.3 Recovery of Signal by Variable Threshold : After an amplitude spectrum is evaluated from a sequence, a threshold may be applied to the spectrum to pick out its relatively strong components. At a first glance, the amplitude of the bursts generated by the aliases is relatively high. A closer look, however, reveals that these bursts

Table 4-4 : Computational time by transputers for a 1024-point (m = 32, p = 32) sequence sampled by parallel a.r.s.

method	base band	speed-up	next higher band	speed-up
direct evaluation, 1 transputer	127.7 sec. 1995584 ticks#	1	127.7 sec. 1995535 ticks	1
parallel a.r.s. algorithm, 1 transputer	21.6 sec. 337840 ticks	5.9	10.7 sec. 167119 ticks	11.9
parallel a.r.s. algorithm, 4 transputers *	5.3 sec. 82528 ticks	24.1	2.9 sec. 46088 ticks	42.6

* excluding the root transputer
\# 1 tick is equal to 64 μs

exhibit a "ringing" which is absent from a frequency component (see Fig. 4-8). This information provides one criterion to differentiate a signal from a residue of an alias. When a strong signal is detected, its amplitude and frequency f are noted. From f, we can predict that bursts are found at $|i.f_s \pm f|$, where i is an integer and f_s is the sampling frequency. Hence there are two tests for a signal : (1) Is the component located within the "band" of a burst ? (2) Is there a ringing around the component ? If a frequency component passes both tests , it should be classified as a signal with a high degree of confidence.

Referring to the example in Fig. 4-3, f_s is 1024Hz and we consider up to 2fs (4 times the Nyquist limit). An initial threshold of ±0.5V may be set and two voltages,-1.023V and 1.499V are detected at 160 Hz and 1,100 Hz respectively. Bursts are expected to appear at 864Hz, 1184Hz and 1888Hz (which are generated by the 160- Hz component), and at 76Hz, 948 Hz and 1972 Hz (which are generated by the

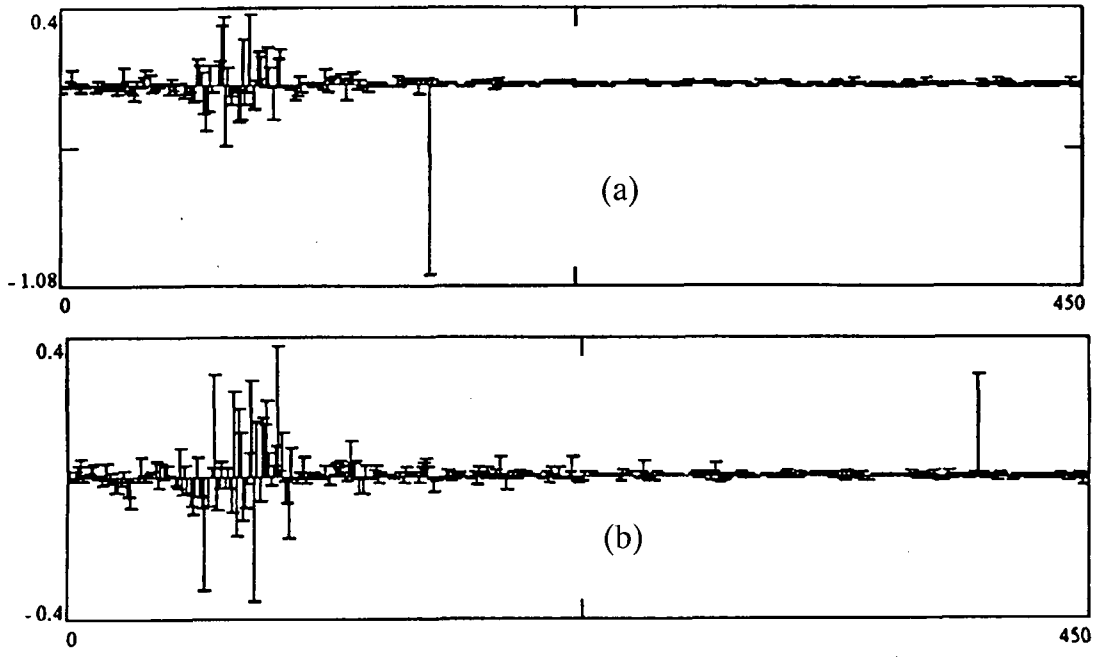


Fig. 4-8 Residue of alias and signal
(a) real part (b) imaginary part

1,100 -Hz component). Let these frequencies be called burst centres and denoted by f_c . Judging from the spectrum of the sampling sequence, we may choose ± 15 Hz from f_c as the bandwidth of a burst. Within such a band, there are no less than 10 components of comparable amplitude but randomized phase angles.

Eliminating the two voltages at 160 Hz and 1,100 Hz, the maximum of the remaining "noise" voltages is 0.42V and the average is 0.042V. Any frequency component located within the bandwidth of a burst with a magnitude smaller than 0.42V is considered as noise. Now we may lower the threshold level to ± 0.1V ($\approx \frac{1}{3}$ {maximum - average noise }). At 400 Hz, a voltage of 0.296V is detected . Since there is no component having a magnitude greater than 0.1V from 385 Hz to 415 Hz, this voltage is taken as a signal. Hence the recovered signal is $-1.023 \cos(2\pi\, 160t) + 0.296 \sin(2\pi\, 400t) + 1.499 \cos(2\pi\, 1{,}100t)$, giving an average accuracy of 98.8% in amplitude when compared with the original.

4.6 Concluding Remarks

The parallel random sampling maintains the anti-alias property of the additive random sampling. A signal well above the Nyquist limit can be recovered by this method. In general, saving in computation of a fast algorithm comes from the symmetry in the trigonometrical terms being multiplied to the sampled data for evaluating the frequency components. For FFT, there are so many symmetry terms to be exploited that the complexity is reduced to NlogN [40]. Symmetry, however, is the source of aliasing. Hence in parallel random sampling, only a limited degree of symmetry is introduced in order to avoid the occurrence of sharp aliases. It is not surprising that the complexity of the problem is still N^2 and the upper limit of the

saving in multiplications is only 87% in the base band . The number of additions remains substantially the same as that of the direct evaluation.

This sampling scheme offers another advantage - a genuine concurrent architecture in nature. During the sampling process, there can be as many as m samplers working independently at 1/m of the composite sampling rate, which relaxes the requirement of using fast hardware. The computation process is also neatly divided into N/8 identical functional blocks , which can be implemented by similar structures running concurrently, thus speeding up the whole process by a maximum factor of N/8.

The insertion of a random variable into the timing of the sampling sequence turns an alias into a burst of phasors having random phase angles. This characteristic provides information to differentiate a weak signal from the residue of an alias. By varying the threshold, a weak signal smaller than the burst can be detected provided that the signal is not located within the burst. A high accuracy in the amplitude of the recovered signal can be achieved when the ratio of the variance of the random variable to the mean of the sampling period is around 10%.

CHAPTER 5

HYBRID ADDITIVE RANDOM SAMPLING

5.1 Introduction

In Chapter 3, additive random sampling (a.r.s.) is introduced as one of the methods in random sampling which offers the advantage of being alias-free. When applying this sampling technique, instead of getting a sharp alias at a particular frequency, a broadband noise in the whole spectrum is seen. This method, however, generates an N^2 complexity in computing the frequency components. For example, when a signal is sampled regularly for 1024 points in one second, a frequency resolution of 1 Hz is obtained. The Nyquist limit is at 512 Hz and the number of complex multiplications required is 5,120 ($N/2 \log_2 N$) by applying FFT. If a lower frequency resolution of 4 Hz and the resulting noise are acceptable, we may sample the signal for 128 points in 1/4 second with a randomized sampling method and evaluate the spectrum up to 512 Hz. The number of complex multiplications required is 16,384 (N^2). To speed up the computation, a method of inserting limited regularity into the random sampling process by interlacing a.r.s. sequences (parallel a.r.s.) has been suggested and described in Chapter 4. With parallel a.r.s., bursts of residual noise appear at the alias locations of the frequency spectrum (Fig. 5-1). A different approach of inserting regularity, which does not generate these bursts of noise, can also be achieved by concatenating a.r.s. sequences. (Three resulting spectra are shown in Fig. 5-2). An anti-alias sampling sequence, called the hybrid additive random sampling (a.r.s.) sequence, can be formed by concatenating an a.r.s. sequence to a

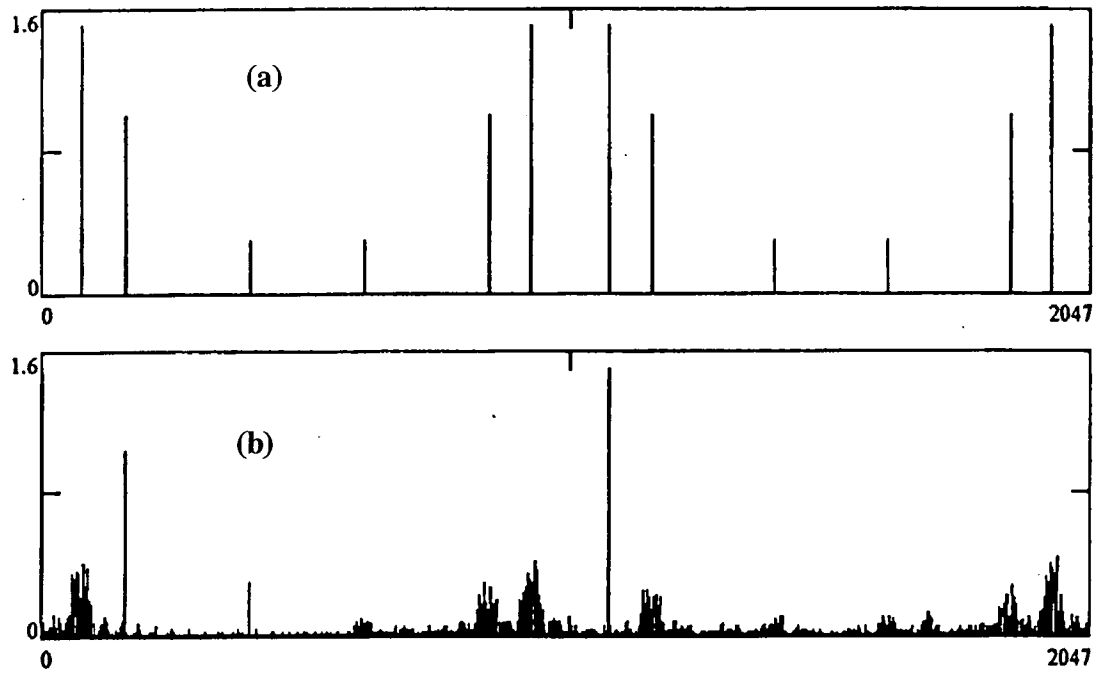


Fig . 5-1 Amplitude spectra of a signal sampled for 1024 points (a) by regular sampling and (b) by parallel a.r.s.

reverse a.r.s. sequence [41]. Under suitable manipulation, at least 75% of the multiplications and additions required in computing the frequency components are saved. The computational algorithm for the hybrid a.r.s. can be implemented in modular form, which can also be realized in either a "recursive" or parallel format.

5.2 Reverse and Hybrid a.r.s.

5.2.1 Timing : The timing of a.r.s. is defined as $t_n = t_{n-1} + \tau_n$, where τ is a uniformly distributed random variable of finite variance [37]. If t_n runs in a reverse order, another sequence $t'_n = t_{N-n}$ is obtained, where N is the sequence length . Obviously t' is also an a.r.s. sequence which can be derived from the a.r.s. equation by redefining τ . A hybrid a.r.s. sequence is formed by taking its first N/2 elements from a normal a.r.s. sequence and the remaining elements from a reverse sequence, i.e.. for n = 1,2,...,N/2-1 :

Table 5-1 : Comparison of Amplitude Spectra

N = 1024; k up to N-1. Samples taken by :
(a) an a.r.s. sequence,
(b) a hybrid a.r.s. sequence with m = 2, p = 512 and
(c) a hybrid a.r.s. sequence with m = 4, p = 256.

	average signal accuracy,%	noise level r.m.s., V	signal-to-noise ratio*, dB	peak noise level, V	peak signal to peak noise,dB
a	95.3	0.0722	29.1	0.189	17.3
b	99.1	0.0731	29.4	0.249	15.5
c	98.7	0.0730	29.3	0.294	14.0

* as defined in chapter 3.

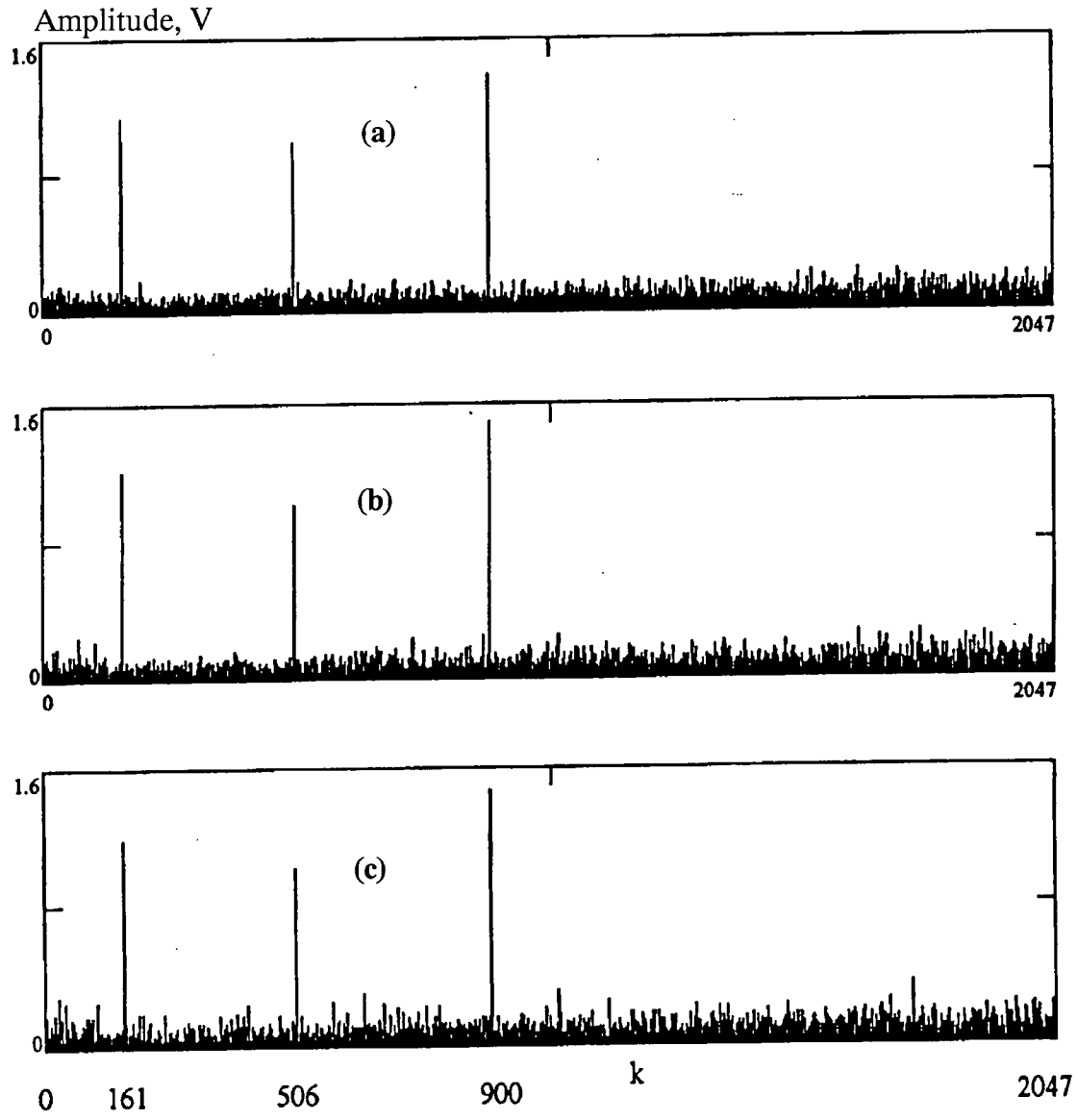


Fig. 5-2 Amplitude spectra (a), (b) and (c) as tabulated in Table 5-1. Note that the Nyquist limit is at k = 512.

$$\begin{cases} t_n = t_{n-1} + \tau_n \\ t_{N-n} = 1 - t_n \end{cases} \tag{5-1}$$

with $t_0 = 0$, $t_{N/2} = 0.5$ and the total sampling period is unity. To compare the performance, a simulated signal, $1.2\cos(2\pi \times 161t) + 1\sin(2\pi \times 506t) + 1.5\cos(2\pi \times 900t)$ volts is sampled for 1024 points by different a.r.s. sequences (with the ratio of the standard deviation to the mean of the sampling periods $\approx 30\%$) and the results of reconstruction are recorded by Table 5-1and Fig. 5-2. It can be seen that the hybrid a.r.s. (b) rivals the a.r.s. (a) in performance regarding noise and accuracy.

The method of concatenation can be extended by adding two N/2 -point hybrid sequences sequentially. The timing equation is :

$$\begin{cases} t_n = t_{n-1} + \tau_n \\ t_{N/2-n} = 0.5 - t_n \\ t_{N/2+n} = 0.5 + t_n \\ t_{N-n} = 1 - t_n \end{cases} \tag{5-2}$$

with $n = 1,2,...,N/4-1$, $t_0 = 0$, $t_{N/4} = 1/4$ and $t_{N/2} = 1/2$. In general, let N samples be taken in unit time and divided equally into m sections. N and m are multiples of 4 and $N >> m$. Defining $q = N/m$, $p = 1,2,...,q-1$ and $s = 1,2,...,m$, the timing for the N samples is given by:

$$\begin{cases} t_p = t_{p-1} + \tau_p \\ t_0 = 0, \quad t_{sq} = \dfrac{s}{m} \\ t_{(s-1)q+p} = \dfrac{s-1}{m} + t_p \qquad \text{for } \textit{odd } s \\ t_{sq-p} = \dfrac{s}{m} - t_p \qquad \text{for } \textit{even } s \end{cases} \tag{5-3}$$

5.2.2 Anti-alias Property : The sequences from eqn (5-2) and eqn (5-3) are anti-alias. The concatenation expressed by eqn (5-2) is equivalent to a linear convolution in the time domain of an N/2-point hybrid a.r.s. sequence with a train of two unit impulses $\delta(0) + \delta(n-N/2)$. The frequency spectrum of the N/2-point hybrid sequence comprises a unit impulse at the origin and a broadband noise, while the spectrum of the pulse train is also a pulse train with a period of 2/N. Convolution is equivalent to multiplication in the frequency domain. Multiplying the above two spectra will give a spectrum similar to that of an a.r.s. sequence, which implies that the hybrid a.r.s. is anti-alias.

It can be shown that the distribution of the sampling periods of the reverse a.r.s. is the same as that of normal a.r.s. For normal a.r.s., the timing at interval n :

$$t_n = nT + \sum_{i=0}^{n-1} \tau_i ,$$

where T is the mean sampling period = 1/N. The sampling period at interval n is :

$$d_n = t_n - t_{n-1} = T + \tau_n$$

where n = 0,1,..., N-1. For reverse a.r.s., the timing at interval r :

$$t_r = rT + \sum_{i=0}^{N-r} \tau_i ,$$

where r = 0,1,...,N-1. The sampling period at interval r is :

$$d_r = t_r - t_{r-1} = T - \tau_r$$

Since τ is a random variable having a zero mean and distributing symmetrically, d_n and d_r have the same distribution. As the normal a.r.s. is alias free, the reverse a.r.s, which has the same random variable and distribution, is thus alias free. When the hybrid a.r.s. is formed according to eqn (5-2), it is obvious that the addition of a section

having the same sampling periods doubles the number of occurrences in each class but the shape of the distribution and the ratio of σ / μ remain the same. Fig. 5-3 shows the distributions of the sampling periods of the hybrid a.r.s. sequence which is tabulated as item (c) in Table 5-1. There are 4 sections in this sampling sequence with 256 points in each section. Fig. 5-3 (a) shows the histogram for the 256 sampling periods of the first section. The histograms of the sampling periods of the first two sections, the first three sections and the whole sequence are depicted in Fig. 5-3 (b),

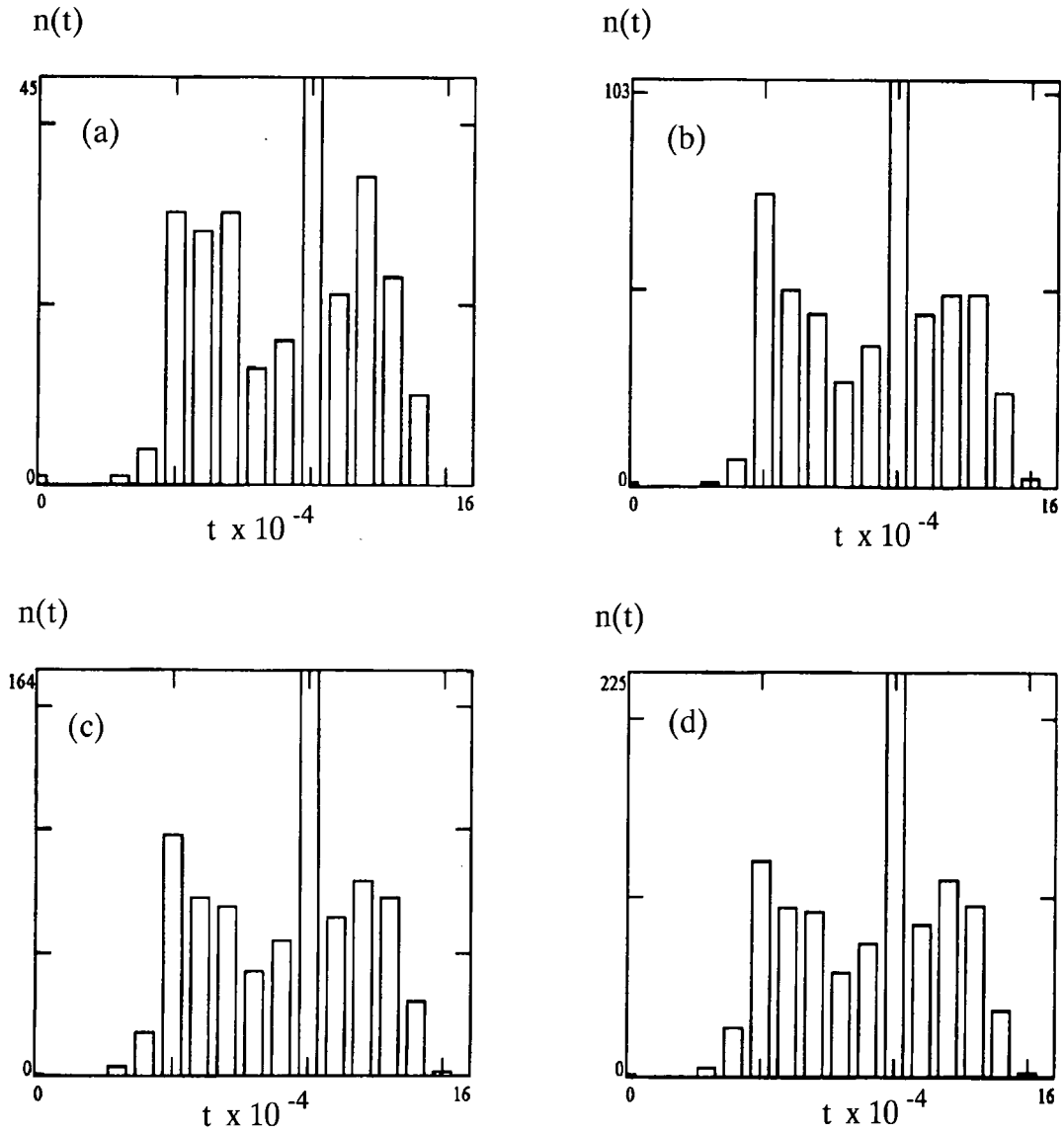


Fig. 5-3 The distributions of the sampling periods of a hybrid a.r.s. with m = 4 and p = 256 : (a) the histogram for the first 256 sampling periods (b) the histogram for the first 512 sampling periods (c) the histogram for the first 768 sampling periods and (d) the histogram for all the sampling periods.

Table 5-2 : Distributions of sampling periods for hybrid a.r.s. sequence shown in Fig. 5-3 with m = 4 and p = 256.

class (x 10^{-4})	periods			
	1 to 256	1 to 512	1 to 768	1 to 1023
0 - 0.9	1	1	1	1
1 - 1.9	0	0	0	0
2 - 2.9	0	0	0	0
3 - 3.9	1	1	3	5
4 - 4.9	4	7	17	27
5 - 5.9	30	74	97	120
6 - 6.9	28	50	72	94
7 - 7.9	30	44	68	91
8 - 8.9	13	27	42	57
9 - 9.9	16	36	55	74
10 - 10.9	45	103	164	225
11 - 11.9	21	44	64	84
12 - 12.9	34	49	79	109
13 - 13.9	23	49	72	95
14 - 14.9	10	24	30	36
15 - 15.9	0	2	2	2
16 - 16.9	0	0	0	0
mean (x 10^{-4})	9.722	9.747	9.753	9.759
standard deviation (x 10^{-4})	4.320	4.304	4.300	4.300

(c) and (d) respectively. It can be seen that the shapes of the four distributions are similar.

Another approach to explain this anti-alias property is by studying the circular auto-correlation of the sampled sequence [42], which will be discussed in Chapter 6. In brief, when assessing components at a higher frequency band, the width of the steps in the time frame of the auto-correlation is reduced, hence a signal and its aliases yield different auto-correlation sequences. In performing the auto-correlation, as the sampled data have a higher probability to "overlap" at the instants when the a.r.s.

sequences are joined to form the hybrid a.r.s. sequence, noise tends to cluster at these corresponding frequencies. For example, for the hybrid a.r.s. sequence denoted by eqn (5-2), noise tends to cluster at the even or odd frequency indices depending on whether the frequency of the signal being sampled is even or odd. This "binning" of noise will be elaborated in the following section.

5.2.3 Binning of Noise : The hybrid a.r.s. sequence formed by concatenating several a.r.s. sequences exhibits periodicity which is reflected from the spectrum of the sampling sequence or the spectrum of the reconstructed signal. In eqn (5-1), two sequences are concatenated (m=2), but the second sequence is not a repetition of the first; hence no periodicity is observed. Fig. 5-4 (a) and (b) show respectively part of the amplitude spectrum of a 1024-point a.r.s. sequence and a hybrid a.r.s. sequence with m = 2. Spikes appear at every frequency index. When the sequence is formed according to eqn (5-2) with m = 4, the last two sections are a repetition of the first two sections, which gives a periodicity of 2 in the sampling sequence. In Fig. 5-4(c), we can see that spikes appear at the even indices. Similarly, when m = 8, the first two sections are repeated 4 times so that a periodicity of 4 is expected. Fig. 5-4(d) reveals that spikes appear at the indices which are a multiple of 4.

Suppose an input signal containing a component with an odd frequency index is sampled by a hybrid a.r.s. sequence with m = 4, we expect that the background noise will tend to gather at every odd index of the reconstructed spectrum. The sampling in time domain is equivalent to a convolution in the frequency domain. As the spectrum of the signal is convolved with the spectrum in Fig. 5-4(c), "overlapping" occurs at every other step so that amplitude of the signal or the noise will appear on the odd indices only. If the input contains both even and odd frequency components, the

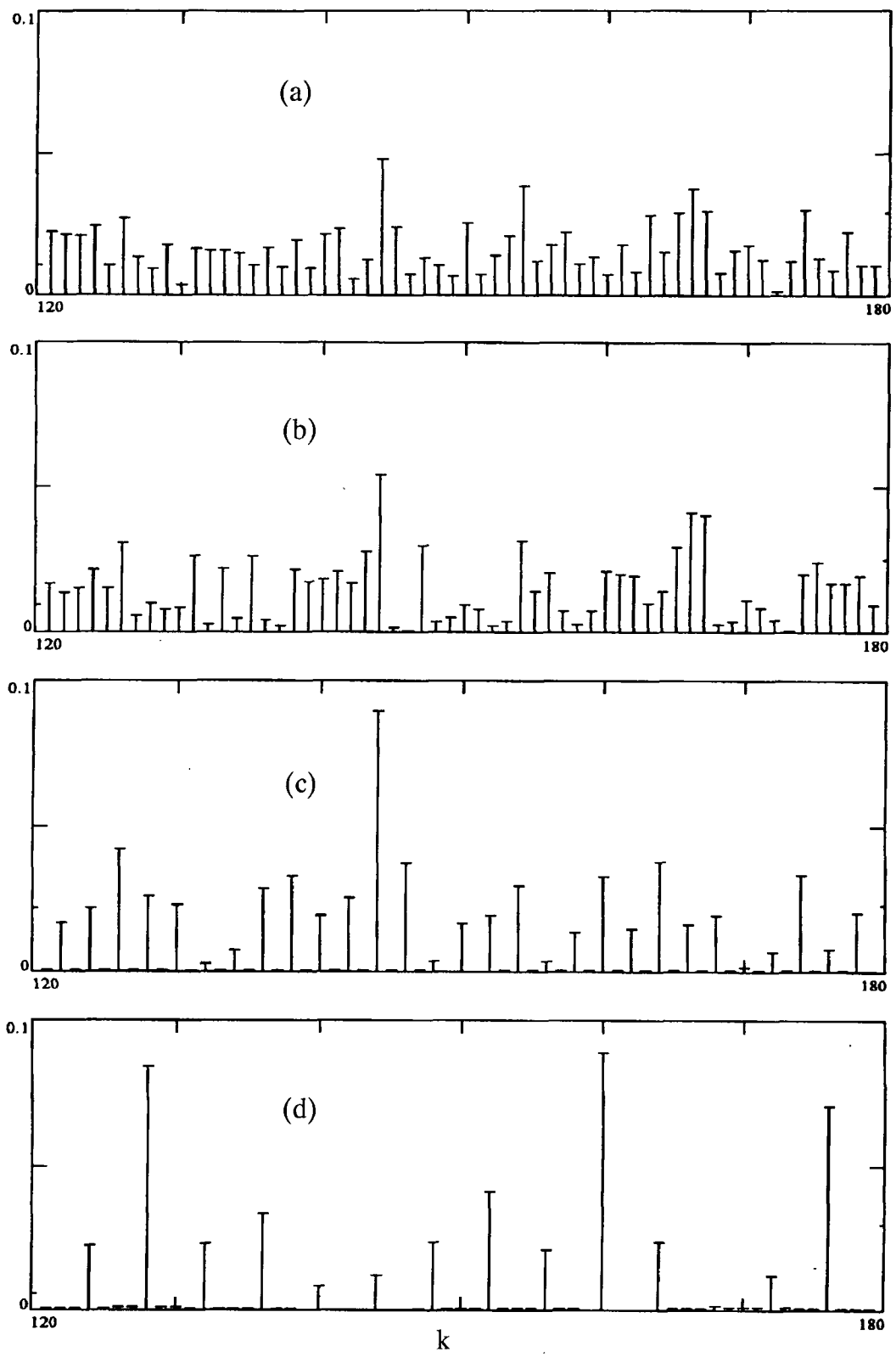


Fig. 5-4 Parts of the amplitude spectra of 1024-point sampling sequences to show the "binning" of hybrid a.r.s. :

(a) a.r.s.

(b) hybrid a.r.s. with m = 2, p = 512

(c) hybrid a.r.s. with m = 4, p = 256

(d) hybrid a.r.s. with m = 8, p = 128.

"binning" of noise will not be apparent as noise will fill the whole spectrum. If m = 8, with an input of frequency index k, noise will gather at every index i = k modulo 4. Fig. 5-5 shows part of the spectrum of a signal $1\sin(2\pi.506t)$ volt sampled by a hybrid a.r.s. with m = 8, p = 128. The input frequency 506 Hz is congruent to 2 modulo 4, hence noise is expected to gather at k = 502, 510, 514, etc.

Although the overall signal-to-noise ratio does not depend on m (but on N, the sequence length), the maximum noise level becomes higher as m increases because of the "binning". In the worst case, with m = 8, the maximum noise level could be 4 times that of a normal a.r.s. sequence. In Table 5-3, the results of two signals sampled by a.r.s. and hybrid a.r.s. are tabulated for showing the effect on the noise levels. The signal-to-noise ratios are about the same for all sampling sequences, but the ratios of signal to peak noise drop about 4 to 5 dB from a.r.s. to hybrid a.r.s with m = 4. A similar drop is observed from hybrid a.r.s. with m = 4 to m = 8.

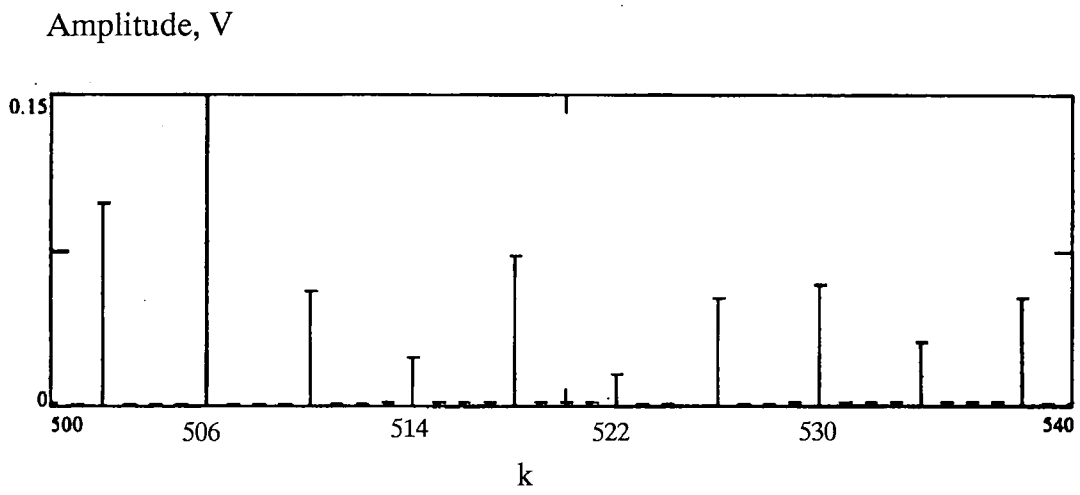


Fig. 5-5 Part of an amplitude spectrum of a signal $1\sin(2\pi.506t)$ sampled by a hybrid a.r.s. with m = 8, p = 128. Note that noise tends to gather at the indices congruent to 2 modulo 4.

Table 5-3 : Noise levels of hybrid a.r.s. sequences

Input signal,V	sampled by	signal amp. , V	noise, r.m.s., mV	peak noise, mV	S/N ratio, dB	signal to peak noise, dB
1.2 cos (2π.161t)	a.r.s., 1024 points	1.202	34.6	120	30.8	20.0
	hybrid a.r.s., m = 4, p = 256	1.185	34.8	189	30.6	15.9
	hybrid a.r.s., m = 8, p = 128	1.199	35.5	294	30.6	12.2
1sin (2π.506t)	a.r.s., 1024 points	0.980	31.7	85	29.8	21.2
	hybrid a.r.s., m = 4, p = 256	1.004	31.0	160	30.2	16.0
	hybrid a.r.s., m = 8, p = 128	0.937	30.4	243	29.8	11.7

5.3 Computation of Signal Amplitude

5.3.1 Symmetry in Timing : The estimates of the frequency components (except the d.c.) are given by [37]:

$$X(k) = \frac{2}{N} \sum_{n=0}^{N-1} x(t_n) \exp(j\, 2\pi k f t_n) \qquad \textbf{(5-4)}$$

where k = 1,2,3,... and f = 1 in the normalized case. Let us consider the case m = 4 in detail. Eqn (5-4) can be rearranged according to the timing given by eqn (5-2). Because of the symmetry between t_n, t_{N-n}, $t_{N/2+n}$ and $t_{N/2-n}$, we find groups of four cosine terms in X (k) having the same magnitude but different signs . This is also true for the sine terms. Using subscripts r and i to denote the real and imaginary parts respectively and x(n) to represent x (t_n), eqn (5-4) becomes :

$$X_r(k) = \frac{2}{N} \left\{ \sum_{n=1}^{N/4-1} [\, (x(n) + x(N-n)\,) + (-1)^{<k>_2} \times \right.$$

$$(x(N/2+n) + x(N/2-n))]\cos(2\pi k t_n) + C_r(k)\}$$

$$X_i(k) = \frac{2}{N}\{\sum_{n=1}^{N/4-1}[(x(n) - x(N-n)) + (-1)^{<k>_2} \times$$

$$(x(N/2+n) - x(N/2-n))]\sin(2\pi k t_n) + C_i(k)\}$$

where $C_r(k) = [x(0)+(-1)^{<k>_2}x(N/2)]+[x(N/4)+x(3N/4)]\cos(2\pi k/4)$,

$C_i(k) = [x(N/4)-x(3N/4)]\sin(2\pi k/4)$,

and $<k>_2$ = k modulo 2 . Neglecting the scaling factor, we can write :

$$X_r(k) = \sum_{n=1}^{N/4-1}[A(n,N-n) + (-1)^{<k>_2}A(N/2+n, N/2-n)]$$

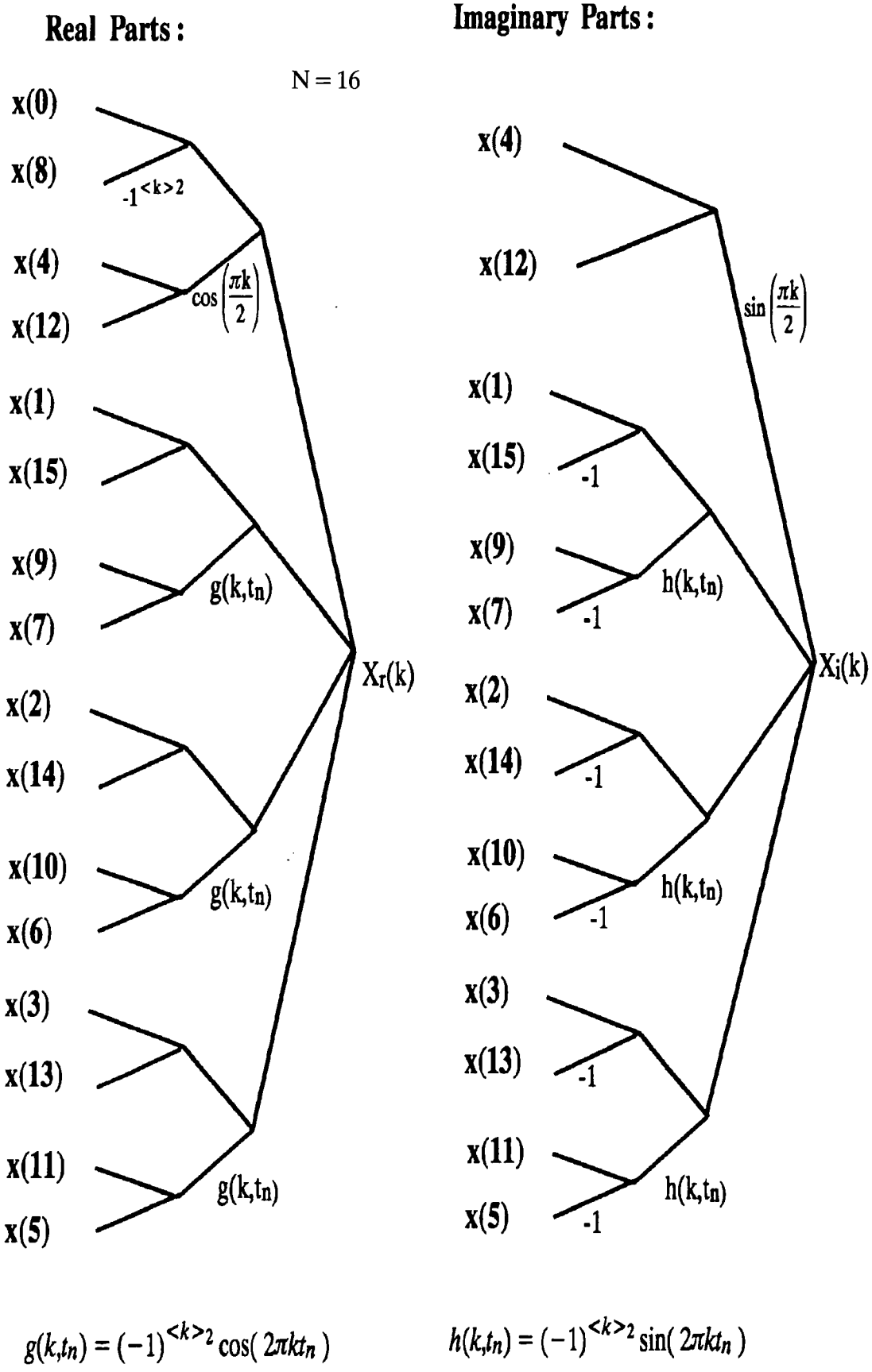
$$\times \cos(2\pi k t_n) + C_r(k)$$

$$X_i(k) = \sum_{n=1}^{N/4-1}[S(n,N-n) + (-1)^{<k>_2}S(N/2+n, N/2-n)]$$

$$\times \sin(2\pi k t_n) + C_i(k) \qquad \textbf{(5-5)}$$

where A(u,v) = x(u) + x(v) and S(u,v) = x(u)-x(v). Fig. 5-6 shows the signal flow diagram of eqn (5-5) with N = 16.

5.3.2 Saving in Computation : In general, to compute N frequency components of a randomly sampled real sequence, it takes approximately $2N^2$ real multiplications and $2N^2$ real additions if $N >> 1$. In eqn (5-5), C_i and C_r require no actual multiplications. (Multiplications with only ± 1 are involved.) For the remaining part, since 1 multiplication is performed every 4 data points, there are in total $N^2/2$ real multiplications and 75% saving is attained. Evaluating all the partial sums of the groups of A,S and C requires 3N additions. These partial sums may be stored and subsequently multiplied to the appropriate sines and cosines. Each X(k) is then computed by adding up N/2 of these products and a C. The total number of additions



$$g(k,t_n) = (-1)^{<k>2} \cos(2\pi k t_n)$$

$$h(k,t_n) = (-1)^{<k>2} \sin(2\pi k t_n)$$

Fig. 5-6 Signal flow diagram of the computational algorithm for eqn (5-5)

is therefore $N^2/2 + 3N$, which is approximately $N^2/2$ if N is large. The percentage saving is approximately 75%. In eqn (5-5), $<k>_2$ means keeping track of k being odd or even. Since the operation is simple, the load incurred is minimal.

As m increases, the saving in multiplications also increases. If $m = 8$, $q = N/8$. Comparing to $m = 4$, there are more symmetry groups between $\exp(j2\pi k t_n)$ and $\exp(j2\pi k t_{2q+n})$ available when k is even; namely, symmetry between $\exp(j4\pi t_n)$ and $\exp(j4\pi t_{2q+n})$, between $\exp(j8\pi t_n)$ and $\exp(j8\pi t_{2q+n})$, etc. For odd k, the frequency components are given by eqn (5-5). Denoting $TA(n,k) = [A(n,N-n) + (-1)^{<k>_2} A(N/2+n, N/2-n)]$ and similarly for TS(n,k), we have for even k :

$$X_r(k) = \sum_{n=1}^{q-1} [TA(n,0) + (-1)^y TA(2q+n,0)] \cos(2\pi k t_n) + TA(q,0)\cos(2\pi k/8) + C_r(k)$$

$$X_i(k) = \sum_{n=1}^{q-1} [TS(n,0) + (-1)^y TS(2q+n,0)] \sin(2\pi k t_n) + TS(q,0)\sin(2\pi k/8) + C_i(k) \qquad \textbf{(5-6)}$$

where $y = 0$ if 4 divides k and $y = 1$ if otherwise. Only 1 multiplication is performed every eight data points when computing the even components.

To estimate the saving, let us consider the odd and even indices separately. For the odd indices, $N^2/4$ multiplications are required. When computing the even indices, only $N^2/8$ multiplications are performed. Hence the total work done is $3N^2/8$, which means that the saving is about 81%. Roughly the same amount of additions can also be reduced if more partial sums can be stored. Although the number of symmetry groups having even indices increases with m, the number of symmetry

groups with odd indices remain the same as the case of m = 4. So the upper bound in the percentage saving is 87%.

When recovering a narrow band in the frequency spectrum, random sampling may require fewer multiplications than regular sampling. Recall that for the FFT, computation is performed in stages so as to reduce the complexity to N log N for a sequence of length N. Owing to this arrangement, there is virtually no extra saving even if fewer components than N are computed. Suppose we are interested in a band of 40 frequency indices from 461 to 500, we need to sample 1,024 points from the input and perform (in general) 10,240 complex multiplications to obtain the spectrum by the FFT. Assuming that the background noise is acceptable, a scheme with sub-Nyquist random sampling and a partial evaluation of the spectrum will meet the specification. Keeping the same resolution, 256 points may be sampled and 40 x 256 = 10,240 multiplications are performed, which is a draw. However, by applying the hybrid additive random sampling with 4 sections, 75 % of the multiplications will be saved, i.e. only 2,560 are performed.

5.4 Realization

By examining the signal flow diagram in Fig. 5-6, we discover that a regular pattern occurs in computing the signal amplitude. Fig. 5-7 shows a modular approach to realize the algorithm, which can easily be implemented by either software or hardware. The structure shown in Fig. 5-7 is in a "recursive" form, the beauty of which is its simplicity. To speed up the computation, a parallel structure can also be derived from this basic module. Referring to Fig. 5-6 or eqn (5-5), every 4 input data points form a group. Therefore the computational algorithm can be implemented naturally

with N/4 processors, each of which is a module deleting the z^{-1}, C_i and C_r from the basic module in Fig 5-7. With N = 1024, the number of processors is then 256. In fact, the number of processors used can be reduced to any number smaller than N/4, and Fig. 5-8 shows how four modules can be employed to share the computation. With four of these modules working in parallel, the computation can be speeded up by a factor of four.

5.5 Concluding Remarks

The amplitude spectra in Fig. 5-2 reveal that the hybrid a.r.s. sequences are anti-alias. The signal amplitudes recovered from all the sampling sequences are accurate and the signal to noise ratios are high. By exploiting the symmetry in timing of a hybrid a.r.s. sequence, at least 75% of the computation required can be eliminated, although the complexity is still N^2. It is obvious that the percentage saving increases with m, but so does the peak noise level. Reasonable choices of m are 4 and 8. (A

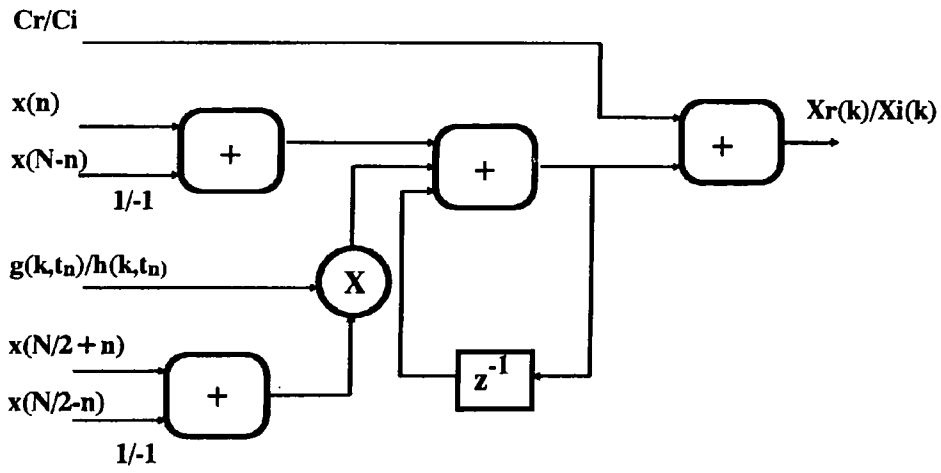


Fig . 5-7 Modular realization of the computational algorithm

trade-off between the computation load and the noise level is inevitable in random sampling.) The hybrid a.r.s. can be implemented systematically either by software or hardware. With the parallel implementation shown in Fig. 5-8, a speed-up factor of N/4 can be attained when there are N/4 processors.

Another advantage of the hybrid a.r.s. is that the amount of saving in computation is a constant irrespective of the number of frequency components to be evaluated, i.e. we achieve 75% saving even if only one frequency component is computed. With other algorithms, e.g. FFT, the maximum amount of saving is attained only if the whole band of N components are computed. Hence this method is especially beneficial for estimating a narrow band of frequency above the Nyquist limit.

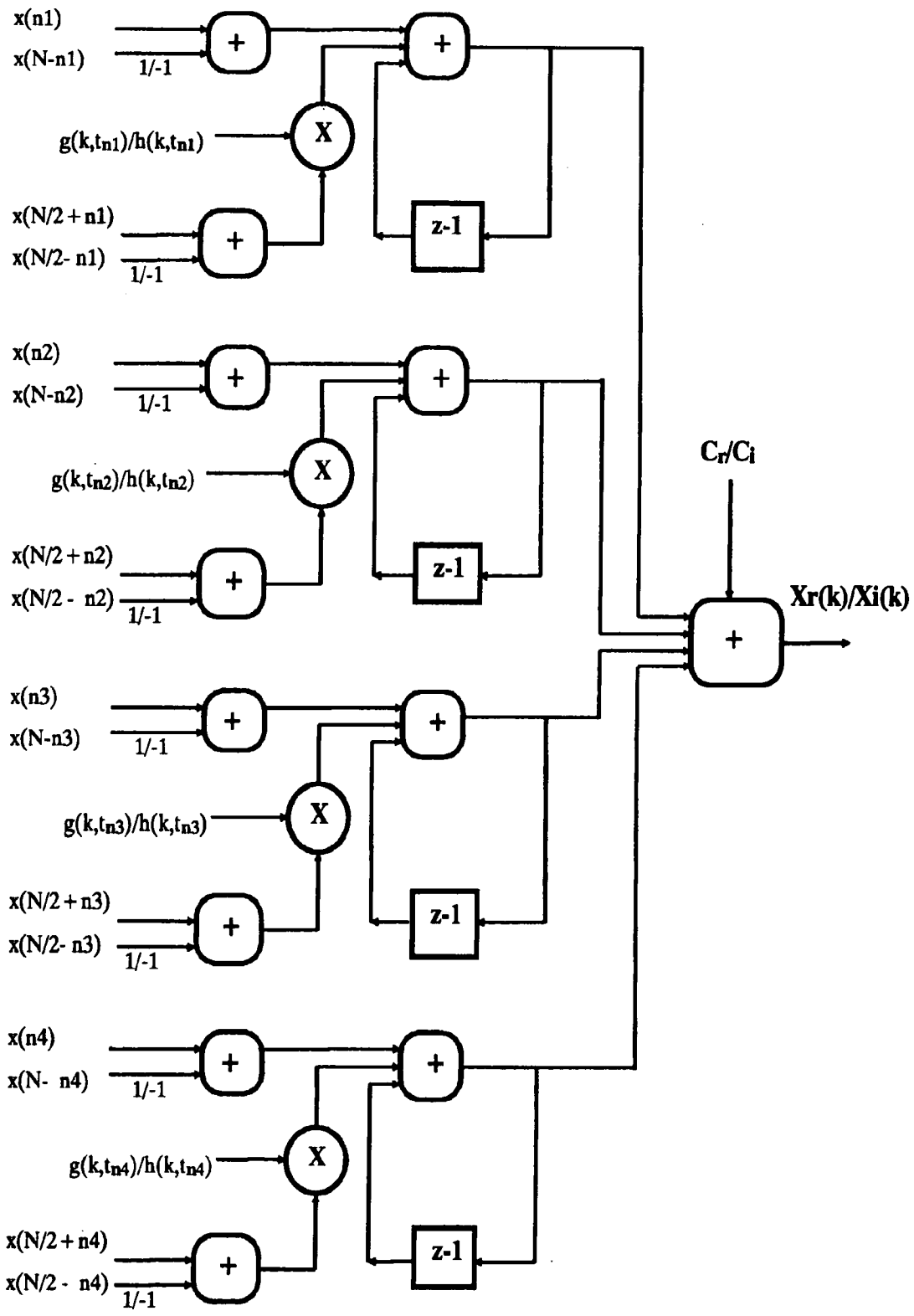


Fig. 5-8 Block diagram for the computational algorithm realized by 4 basic modules. To share the load evenly, $n1 = 1,2,..., \frac{N}{16}-1$, $n2 = \frac{N}{16}, \frac{N}{16}+1, ..., \frac{N}{8}-1$, $n3 = \frac{N}{8}, \frac{N}{8}+1, ..., \frac{3N}{16}-1$, *and* $n4 = \frac{3N}{16}, \frac{3N}{16}+1, ..., \frac{N}{4}-1$.

CHAPTER 6

AUTO-CORRELATION AND POWER SPECTRUM OF RANDOMLY SAMPLED SEQUENCES

6.1 Introduction

Correlation is a mathematical operation which closely resembles convolution. An auto-correlation is performed when a sequence is correlated *with itself*. Since the auto-correlation of a sequence is related to its power density spectrum, this operation can be applied to a randomly sampled sequence as an aid to study and explain the anti-alias property of random sampling. The process can also be regarded as a method to convert a randomly sampled sequence to a regularly sampled sequence of a desired sequence length.

Auto-correlation is only one of the possible means to explain the anti-alias property of random sampling. The other method is to study the outcome of the *convolution* of the frequency spectra of the sampling sequence and the sample sequence, which is already mentioned in chapters 4 and 5. (Note that auto-correlation involves only the sample sequence but the convolution process involves both the spectra of the sampling sequence and the sample sequence.) For the sake of completeness, the convolution method is briefly repeated below.

6.1.1 Convolution in the Frequency Domain : When a signal is sampled in the time domain, the resulting sequence is obtained by multiplying the signal with the sampling sequence. Multiplication in the time domain is equivalent to convolution in the frequency domain. Hence by examining the frequency spectrum of a sampling

sequence, one will easily envisage the spectrum of the sampled signal. Let us consider the spectrum of a regular sampling impulse sequence which is also a periodic impulse sequence. Hence the spectrum of a signal sampled by it is repetitive as shown in Fig. 3-2 after the convolution is performed . With random sampling, the situation is very different. Fig. 6-1 (a) shows the amplitude spectrum of a length-1024 additive random sampling (a.r.s.) sequence which comprises a unit impulse at the origin and a broadband noise. When this spectrum is convolved with a signal spectrum, the original signal spectrum together with a background noise is obtained. Fig. 3-8 (b) shows a typical result where no aliases are found. With parallel a.r.s., bursts of noise appear as residues of the aliases, which can be seen in Fig. 6-1(b). A detailed discussion is given in section 4.3. The spectrum of a hybrid a.r.s. sequence is found in Fig. 6-1(c). Its anti-alias property is obvious since its spectrum resembles that of an genuine a.r.s. sequence. The binning effect created by the concatenation is described in section 5.2.3.

6.1.2 Auto-correlation of a Sequence : Starting from here a totally *different approach* will be introduced. This approach is to study the auto-correlation of the *sample (or data)* sequence in the time domain so as to confirm the anti-alias property of the sampling sequence.

Suppose a signal sequence x(n) has finite energy. The linear auto-correlation of x(n) is defined as a sequence :

$$R_x(l) = \sum_{n=-\infty}^{\infty} x(n+l)\,x(n) \qquad l = 0, \pm 1, \pm 2, \ldots \tag{6-1a}$$

or

$$R_x(l) = \sum_{n=-\infty}^{\infty} x(n)\,x(n-l) \qquad l = 0, \pm 1, \pm 2, \ldots \tag{6-1b}$$

Eqn(6-1a) represents that the sequence x(n) is shifted to the left and (6-1b) to the right during the auto-correlation.

The Fourier transform of the auto-correlation function of a signal sequence is called the power spectral density function [43] and is denoted by $S_x(e^{j\omega})$. The power spectral density function is given by :

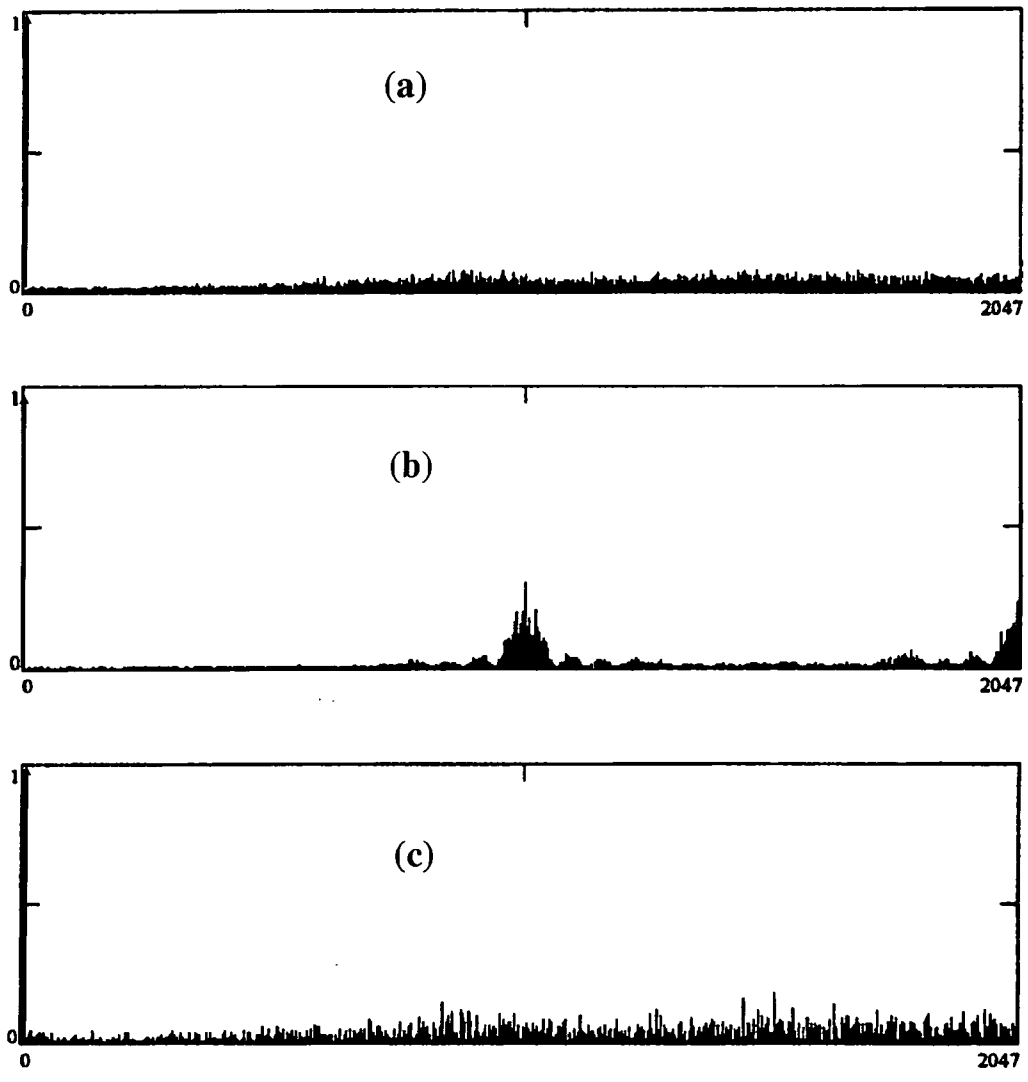


Fig. 6-1 Amplitude spectra of 3 random sampling sequences having a sequence length of 1024 points:

(a) additive random sampling (a.r.s.)sequence, (b) parallel a.r.s. sequence with 4 sections, and (c) hybrid a.r.s. sequence with 4 sections.

$$S_x(e^{j\omega}) = \sum_{l=-\infty}^{\infty} R_x(l)\, e^{-j\omega l} \qquad \textbf{(6-2a)}$$

and its inverse is given by :

$$R_x(l) = \frac{1}{2\pi} \int_{-\pi}^{\pi} S_x(e^{j\omega})\, e^{j\omega l}\, d\omega \qquad \textbf{(6-2b)}$$

Eqn (6-2) is called the Wiener-Khintchine relations, the proof of which can be found in many books [10,43]. A plot of $S_x(e^{j\omega})$ versus ω is the power density spectrum (or simply the power spectrum), and its value at a given radian frequency ω is called the power spectral density.

The Wiener-Khinchine relations assert that auto-correlation of the sample sequence is related to its power spectrum. This fact implies that studying the auto-correlation of the *sample* sequence in the time domain can reveal the anti-alias property of the sampling sequence. Before going into this study, the *circular* auto-correlation must first be defined.

6.2 Circular Auto-correlation

6.2.1 Regular Sampling : As discussed in section 2.1, when a continuous signal is sampled to become a sequence of finite length, a periodic extension of the sequence in the time domain is assumed in calculating its Fourier transform. Corresponding to this periodic extension, the auto-correlation of a sequence of finite length is to be performed *circularly* rather than linearly. Parallel to eqn (6-1), the circular auto-correlation of x(n) of length N is :

$$\tilde{R}_x(l) = \sum_{l=0}^{N-1} x(<n + l>_N)\, x(n) \qquad \textbf{(6-3a)}$$

or

$$\tilde{R}_x(l) = \sum_{l=0}^{N-1} x(n)\, x(<n - l>_N) \qquad \textbf{(6-3b)}$$

where $<y>_N$ denotes y modulo N. Fig. 6-2 shows a graphical illustration of the circular auto-correlation. For simplicity, *auto-correlation* will refer to circular auto-correlation in this chapter when no ambiguity arises.

It can be shown that $\tilde{R}_x(l) = \tilde{R}_x(N-l)$ and whether the shifting of x(n) in the correlation is to the left or right will give the same power spectrum. As an illustration of the Wiener-Khinchine relations, consider a 16-point sequence {x(n)} = {2, 0.8, -2.212, 6.498, -7, 0.346, -3.356, 8.035, 4, -4.557, 2.121, -2.741, 1,-12.588, 3.536, 4.208}. The auto-correlation of x(n) according to eqn (6-3a) is a sequence $\{\tilde{R}_x(l)\}$ = {26, -8.096, 0.836, -7.312, 3.5, 0.948, -16.839, 14.46, -1, 14.46, -16.839, 0.948, 3.5, -7.312, 0.839, -8.096}. Let X(k) and FR(k) be the Fourier transform of x(n) and $\tilde{R}_x(l)$, and it is found that $\{|X(k)|^2\} = \frac{1}{2}\{\ FR(k)\}$ = {0, 1, 9, 0, 16, 1, 0, 25}.The sequences $\{\tilde{R}_x(l)\}$, $\{|X(k)|^2\}$ and $\frac{1}{2}\{\ FR(k)\}$ are plotted in Fig. 6-3.

When a continuous signal is sampled for N points in a total duration of T seconds, the sampling period t_s = T/N and the sampling frequency $f_s = 1/t_s$ = N/T. In the frequency spectrum, there will be N spectral lines representing a bandwidth of f_s. Hence the frequency resolution of the Fourier transform $\Delta f = f_s / N = 1/T$. In these N spectral lines there are only N/2 distinct values because after $f_s/2$, the spectral lines are the images of the first N/2 lines (see Fig. 6-4). In order to push up this Nyquist limit, t_s must be decreased.

Let us now consider the auto-correlation of x(n) and its power spectrum. When the auto-correlation is performed, the step size of each shift is t_s so that N values for

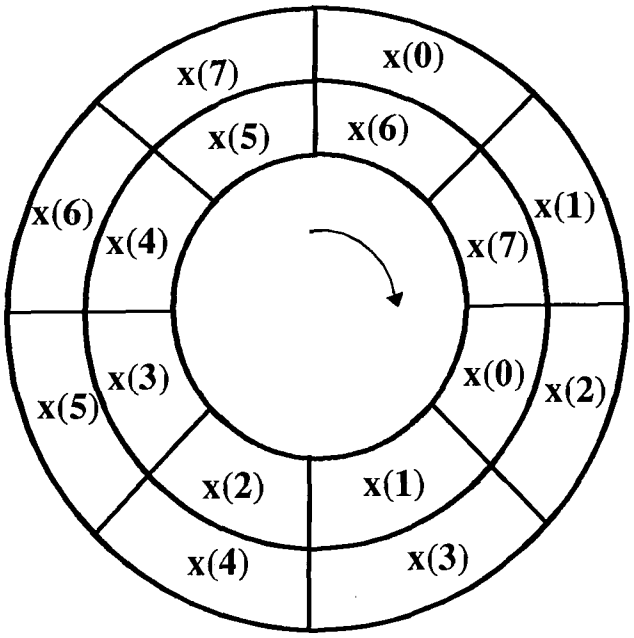


Fig. 6-2 Graphical illustration of the circular auto-correlation of an 8-point sequence with $l = 2$, i.e. $\tilde{R}_x(2)$.

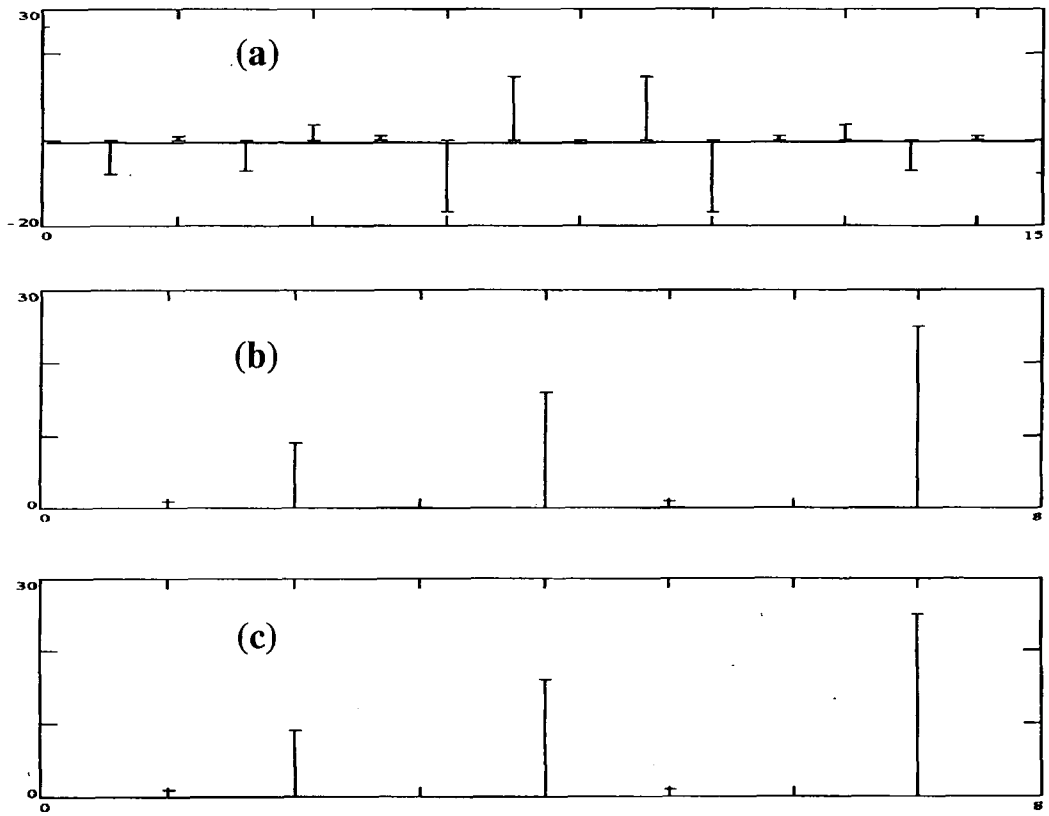


Fig. 6-3 Illustrating the Wiener-Khintchine relations:

(a) the circular auto-correlation $\tilde{R}_x(l)$ of a length-16 sequence x(n),

(b) the magnitude spectrum of the Fourier transform of $\tilde{R}_x(l)$,

(c) the power spectrum of x(n).

$\widetilde{R}_x(l)$ are obtained, thus yielding a power spectrum of a period of N. Suppose the step size is now halved so that the number of points obtained for $\widetilde{R}_x(l)$ is doubled. Would the Nyquist frequency be extended ? The answer is no. In this 2N-point sequence, all the odd-numbered points are equal to zero and the even-numbered points are equal to the values of the former N-point sequence. Zeros result in these "half steps" because there are no sample values between two steps (or two sampling time intervals) in regular sampling; hence there are no "overlaps" in the correlation. The 2N-point $\widetilde{R}_x(l)$ padded with N zeros may be considered as an up-sampling or expansion of an N-point sequence. Although the number of points is increased, its spectrum is "compressed" in the frequency domain so that there is no change in the Nyquist limit

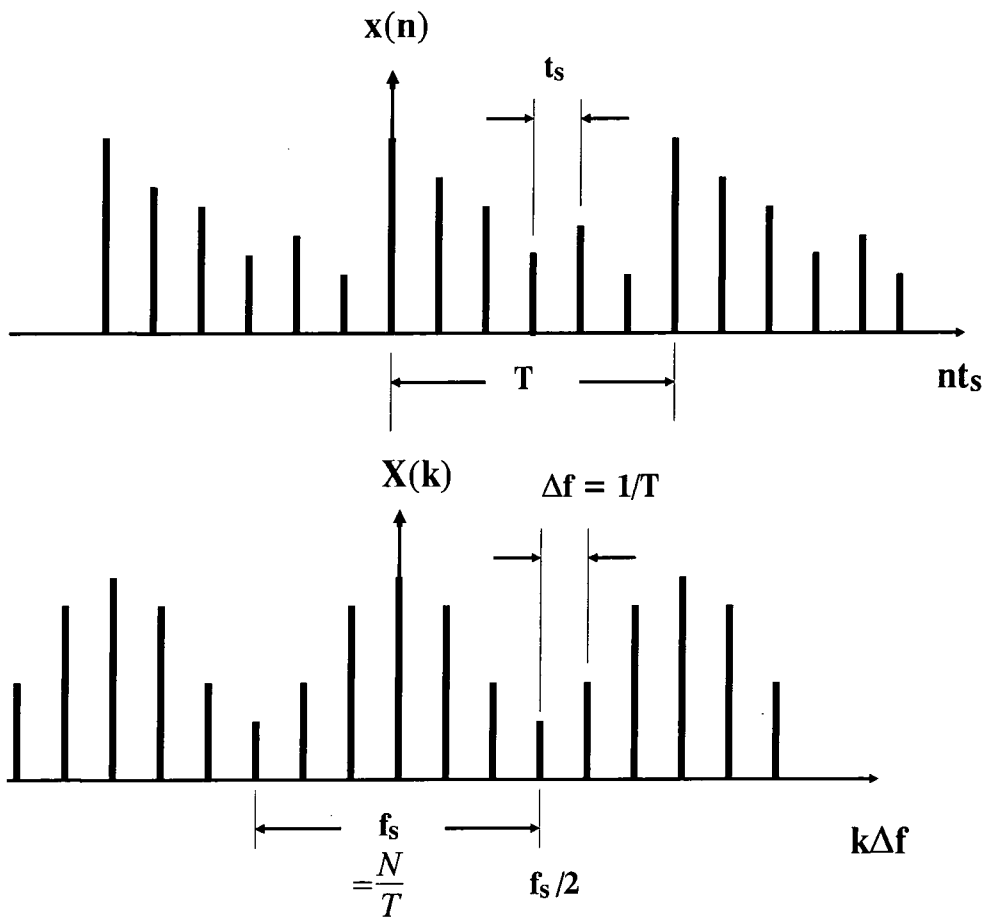


Fig. 6-4 Relationship between the timing of an extended sample sequence and the resolution of its frequency spectrum.

[44]. Obviously, since there is no extra information obtained in the time sequence, the information content in the frequency spectrum should remain unchanged.

6.2.2 Random sampling : One can argue that a randomly sampled sequence is alias-free by considering the possibility of *decreasing the step size* of its auto-correlation function. For a continuous signal, a linear auto-correlation can be performed at any step size because there is a probability of finding overlaps everywhere within the duration of the signal. When sampling is randomized in the time domain, the resulting sequence is also a "continuous" signal in the sense that there is a probability, however small, that a sampling point will fall into a particular instant. Hence, in contrast to regular sampling, overlaps exist when the step size of the auto-correlation decreases, which demonstrates that the *Nyquist frequency* with random sampling is not limited by the apparent sequence length.

Let a sample sequence of N points be sampled by a random sampling scheme with a mean sampling period μ, and the auto-correlation is performed with a step size t_s smaller than μ. Assume that the probability density function (p.d.f.) of a point has a gaussian distribution as shown in Fig. 6-5. Since the probability of finding a sampling point *exactly* at time t is zero, we consider a region in the time frame. The probability of finding a sampling point between t_2 and t_1 is:

$$P(t) = \int_{t_1}^{t_2} p(\tau)\, d\tau \qquad \textbf{(6-4)}$$

When this point is shifted by a step, the probability of finding a sampling point in the corresponding region is :

$$P(t') = \int_{t'_1}^{t'_2} p'(\tau)\, d\tau \qquad \textbf{(6-5)}$$

Then the probability of an overlap is :

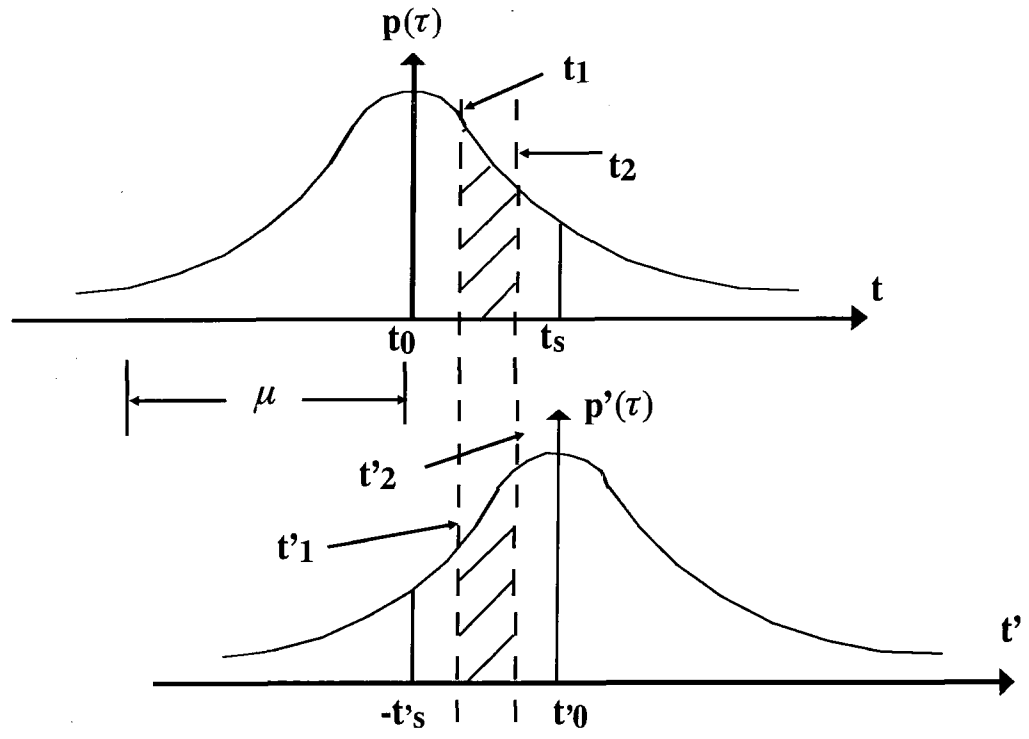


Fig. 6-5 The overlapping of the probability density function of a point in an auto-correlation when the step size is smaller than the mean sampling period.

$$P(v) = P(t).P(t') > 0$$

since for a gaussian distribution, P(t) and P(t') are non-zero. For example, if the standard deviation $\sigma = 0.3\mu$ and the step size $= 0.5\mu$, and the whole step is considered as the region of overlapping, then the probability P(t) = P'(t) = 0.4082, and P(v) $= 0.1666$. According to Bilinskis and Mikelsons [28], the probability density function of a sampling point in an additive random sampling scheme approaches $1/\mu$ when the time t is large enough (see Fig. 3-7). Hence the probability of finding a sampling point in an interval Δt is $\Delta t/\mu$ and this probability is greater than zero so long as Δt is greater than zero.

Let us consider the effect of reducing the step size successively. Suppose the step size is $\mu/2$ (a *regular* step size) so that there will be 2N points in the auto-correlation sequence $\tilde{R}_x(l)$. Since the probability of finding an overlap in each

step is non-zero, $\widetilde{R}_x(l)$ will *not* be a sequence with zero values in alternate positions, which is the case with regular sampling, although there are zeros distributed randomly inside the sequence. In effect there are 2N distinct data points in $\widetilde{R}_x(l)$ so that its power spectrum has N distinct values, which means that the Nyquist limit is twice that of an auto-correlation of N points while the frequency resolution remains unchanged. Thus if the step of the auto-correlation is a regular size of $\mu/2^i$, the Nyquist limit of the resulting power spectrum will be further extended to 2^{i-1}N distinct values, where i is an integer. Following this argument, the Nyquist limit could be extended in principle to any even multiple of N. Intuitively speaking, the extra information in the frequency spectrum is obtained because information in the time domain may be available between two consecutive points of a regular time grid.

6.2.2.1 A Pseudo-continuous sampling : Coming back to those zeros distributed randomly inside $\widetilde{R}_x(l)$ mentioned previously, they are the source of the random background noise seen in the frequency spectrum. Conceptually the random sampling is a corrupted case of continuous sampling. Suppose $\widetilde{R}_{xe}(l)$ is the exact auto-correlation, then conceptually we can write :

$$\widetilde{R}_{xe}(l) - \widetilde{R}_x(l) = N(l)$$

or

$$\widetilde{R}_{xe}(l) = \widetilde{R}_x(l) + N(l) \qquad \textbf{(6-6)}$$

where N(l) is the sequence of "drop-outs" whose non-zero values corresponds to the positions of those zeros in $\widetilde{R}_x(l)$. Therefore, according to eqn(6-6), the exact spectrum of the sampled signal is split into two parts, and both the spectra of $\widetilde{R}_x(l)$ and $N(l)$ contain the frequency information of the exact spectrum but corrupted by noise. From the above argument, a random sampling scheme may be considered an irregular

selection of points from a continuous or an infinitesimally dense and regular time grid.

For illustration, consider a sequence $\{y_1(n)\}$ which contains 32 non-zero data points sampled irregularly from a regular sequence $\{y(n)\}$ of 128 points. At those locations j where data are not selected, $y_1(j) = 0$. Therefore, the difference between $\{y(n)\}$ and $\{y_1(n)\}$is a sequence $\{y_2(n)\}$ having 96 non-zero values. The power spectrum of these three sequences are shown in Fig. 6-6. It can be seen that the Nyquist limit is not affected by the irregular down-sampling and it remains at $k = 64$ for all three spectra. From Fig. 6-6 (b), where only one-quarter of the original sequence is selected, the signal at $k = 60$ can still be recognized although the signal-to-noise ratio is low (13.6 dB) as compared to the case in Fig. 6-6 (c), where the signal-to-noise ratio is 23 dB.

The aim of the above example is to show that a randomly sampled sequence, as an irregular down-sampling from a continuous signal, would maintain the Nyquist limit of a continuous signal, i.e. at infinity. However, in handling the missing data problem that drop-outs occur in a sequence, interpolation methods and linear least-square fitting such as Lomb's algorithm [45,46] can be applied to construct a better periodogram than a direct Fourier transform of the sequence.

6.3 Evaluation of Auto-correlation

The theoretical background of the circular auto-correlation has been covered and the next step is to write computer programs for computing its numerical values. When doing so, practical difficulties are encountered and solutions to them are suggested in the following sections.

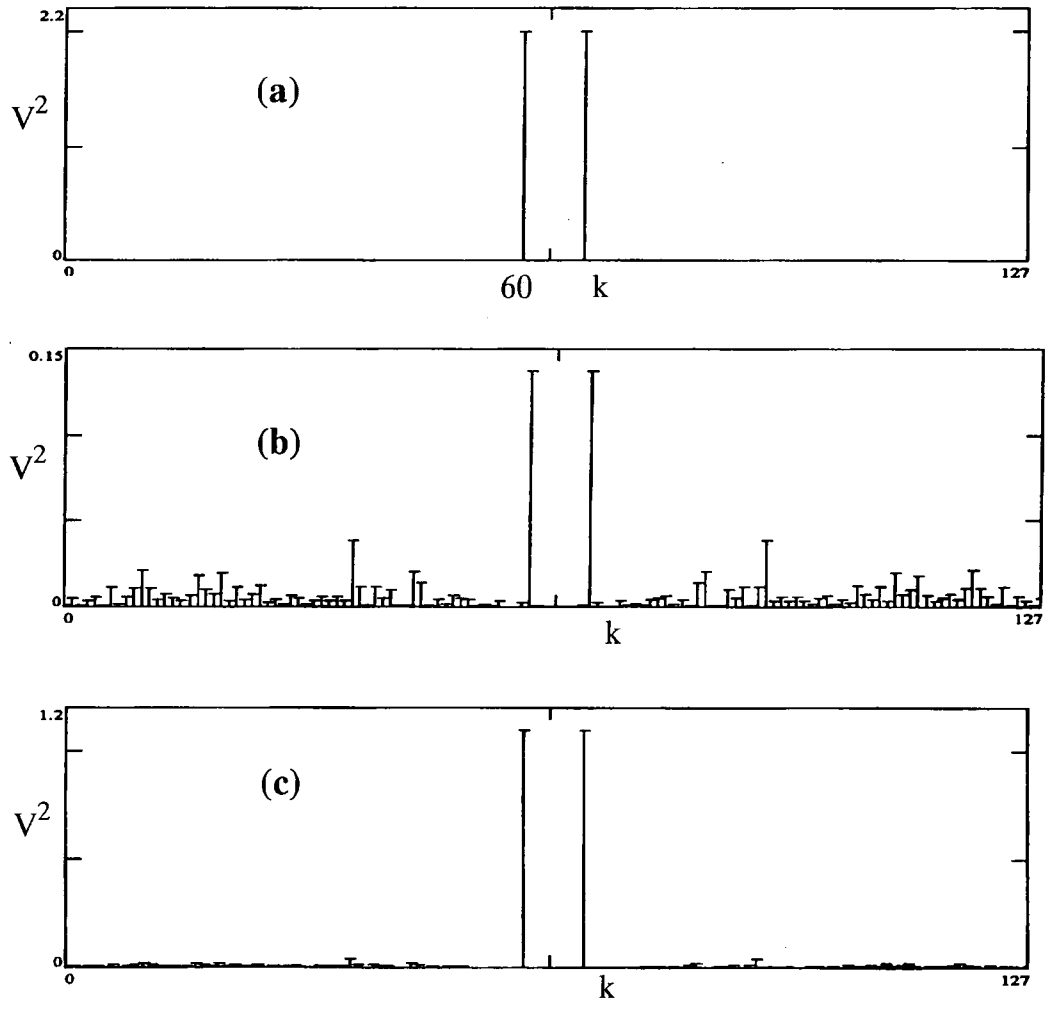


Fig. 6-6 Power spectra of :

(a) a sequence {y(n)} sampled regularly for 128 points,

(b) a sequence {$y_1(n)$} of 32 samples selected unevenly from the above 128-point sequence, and

(c) the sequence {$y_2(n)$} containing the remaining 96 samples.

6.3.1 Step Size and Window Width : The auto-correlation of a randomly sampled sequence $x(t_n)$ can be written as :

$$\tilde{R}_x(l) = \sum_{l=0}^{N-1} x(<t_n + l.t_s>_{N\mu})\, x(t_n) \qquad \textbf{(6-7a)}$$

or

$$\tilde{R}_x(l) = \sum_{l=0}^{N-1} x(t_n)\, x(<t_n - l.t_s>_{N\mu}) \qquad \textbf{(6-7b)}$$

where t_s is the step size, μ is the mean sampling period, and $N\mu$ is the total sampling period. To evaluate the auto-correlation of a sequence sampled randomly is more

complicated than a sequence sampled uniformly. For a uniformly spaced sequence, the step size is always one sampling interval (t_s in Fig. 6-4). So in programming eqn (6-3), the rotation of the input array in the time frame is achieved simply by modifying their indices n. Then the products between the corresponding elements of the two arrays can easily be formed. When evaluating a randomly sampled sequence, the results are sensitive to the computational errors of the host system. Unlike regular sampling, the sampling times of a randomized sequence must be recorded. To rotate the sequence means adding the value of the required step to the sampling times of the input sequence (see eqn (6-7)). The next step after rotation is to compare *point by point* the original times with the shifted times to search for equal values. (In theory, a point has no width. As discussed in section 6.2.2 and shown in Fig. 6-5, we have to consider a region for the overlapping.) Besides the involvement in searching[1], another problem arises when comparing the time values, which are not integers but most likely floating-point numbers. Usually the sampling times for random sampling are represented by many significant digits. Thus an exact match of two points requires that all the bits representing the values, up to the least significant bit, are equal. When the step size is added to the original set of times, rounding off is bounded to occur, especially in circular auto-correlation where modulo arithmetic is involved. So the probability of finding *exact* matches in the two sets of times would be extremely low, and many zeros will be obtained in the results.

To overcome this problem, a tolerance may be incorporated into the comparison such that the chance of matching can be increased. Instead of comparing the two sets of time values by points, windows of width w are opened in one of the

1 The techniques to avoid modulo arithmetic and to reduce search effort will be discussed in section 6.3.2.

sets. Now the process becomes checking whether or not a set of points fall into these windows. By this modification, the influence of machine errors is completely eliminated and the probability of matching is enhanced. The idea is illustrated graphically in Fig. 6-7, where part of a randomly sampled sequence shown in (a) is shifted by a step t_s as shown in (b). No overlaps can be found although there are three pairs of points lying closely together in time. In Fig. 6-7 (c), windows are opened and the three points in (a) are now captured. Referring to eqn (6-4), P(t) is the probability of finding a sampling point in the original sequence in a region where t_1 and t_2 correspond to the beginning and the end of a specific window, i.e. $w = t_2 - t_1$. In an N-point sequence with a mean sampling period of μ, the total sampling period is $N\mu$ in which N windows appear. The probability of the occurrence of a window is thus $\frac{w}{\mu}$. The probability of an overlap in time is then :

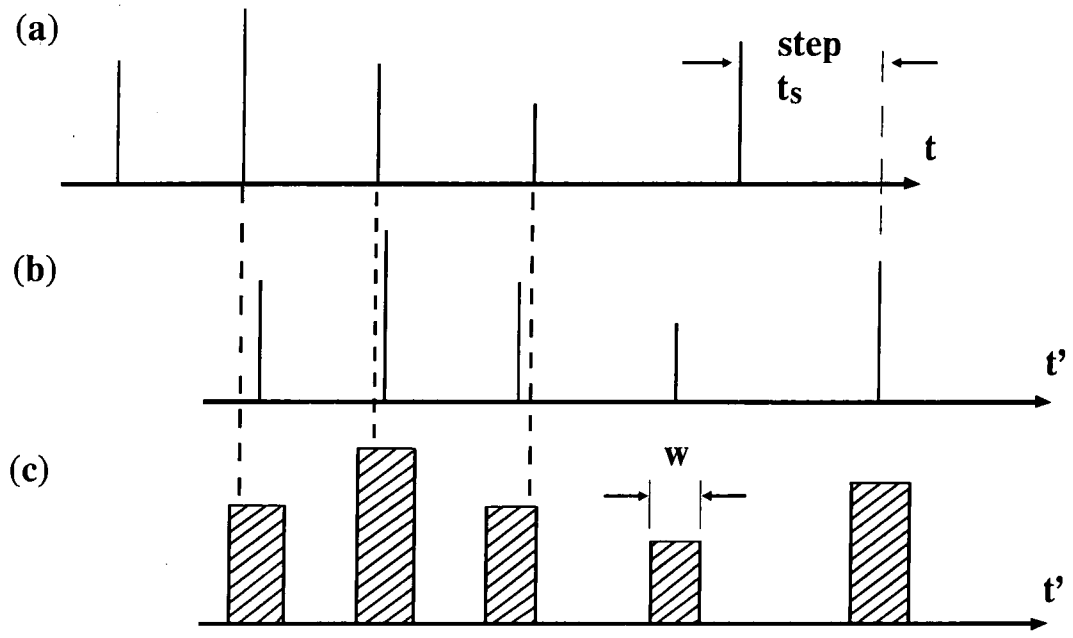


Fig. 6-7 Using windows in auto-correlation :
(a) part of a randomly sampled sequence,
(b) the sequence in (a) shifted by a step t_s in the time frame, and
(c) windows of width w opened in the shifted sequence.

$$P(v) = \frac{w}{\mu}\int_{t_1}^{t_2} p(\tau)d\tau = \frac{w}{\mu}P(t)$$

According to Bilinskis and Mikelsons [28], the probability density function of a sampling point in an additive random sampling scheme approaches $1/\mu$ when the time t is large enough. Therefore, P(t) = w/μ and P(v) = $(w/\mu)^2$. Hence P(v) increases with the increasing window width w. Returning to the example in section 6.2.2 where the gaussian distribution has a standard deviation of 0.3μ. The step size of correlation is 0.5μ(= the region of interest) and windows are not used in the shifted sequence. Given that a sampling point is found in the original sequence between t_0 and t_s, i.e. P(t) = 1, then P(v) = P(t') = 0.4082. If a window centred at t'$_0$ of w = 0.8μ is used, the conditional probability of P(v) will become 0.8, i.e. nearly doubled.

6.3.1.1 Nyquist Limit and Noise : Consider a signal x(t) = 1.5 cos (2π . 50 t) +2 sin (2π . 506 t) +1 cos (2π . 1400 t) sampled by a.r.s. for 512 points in 1 second. Its power spectrum is recovered up to k = 1535 and shown in Fig. 6-8 for reference. To illustrate the effects of the step size and window width in auto-correlation, the above sequence x(t_n) is auto-correlated with a step size of 1/1024 = 976.5 μs and different window widths. The results $\tilde{R}_x(l)$ with l = 1,2, ..., 200 are shown in Fig. 6-9 where the window width decreases from $\pm 146\mu s$ in (a) to $\pm 24.4\mu s$ in (d). It can be seen that all the sequences share a similar "shape" in the time frame, implying that their frequency contents are more or less similar. Their corresponding DFT's are shown in Fig. 6-10, where the Nyquist limit is at k = 512 as the time sequences comprise 1024 points regularly spaced in time. The component at k = 1,400 is aliased to 376 and 648. The amplitudes of $\tilde{R}_x(l)$ and their DFT's are proportional to the widow width, e.g. the maximum amplitudes in Fig. 6-9 (a) and 6-10(a) are about 6 times those in Fig. 6-9 (d) and 6-10 (d) respectively. This happens because the chances of overlapping are 6

times higher in (a) than in (d). It can also be seen in Fig. 6-10(d) that drop-outs are prominent, which results in high noise in its power spectrum. Turning to their Fourier transforms in Fig. 6-10, the signal-to-noise ratios are 19.3 dB in (a), 17.6 dB in (b), 14.7 dB in (c) and 11.8 dB in (d). In Fig. 6-10 (d), the maximum noise level is even higher than the weakest signal component. As a reference, the signal-to-noise ratio of the spectrum obtained by direct evaluation and shown in Fig. 6-8 is 24.8 dB. In Fig. 6-10 (a) and (b), the power of the three signal components maintain roughly a ratio of 2.25 to 1 to 4.

Let us continue the auto-correlation with finer step sizes. Fig. 6-11 (a) shows the correlation results $\tilde{R}_x(l)$, l = 0,1,..., 200, with a step size of 1/2048 = 488 μs and (b) shows the correlation results with a step size of 1/4096 = 244 μs. The window widths are ±20 % of the respective step size in both cases. Hence the sequence lengths are 2,048 points and 4,096 points respectively. It is obvious, even in the time domain, that the auto-correlation sequence in Fig. 6-11 (b) should contain a higher frequency content than in (a). Their corresponding Fourier transforms confirm the above conjecture. In Fig. 6-12 (a), the Nyquist limit is at k = 1,024 since the sequence length is 2,048. Comparing to the spectra in Fig. 6-10, the aliases at 376, 518 and 974 are

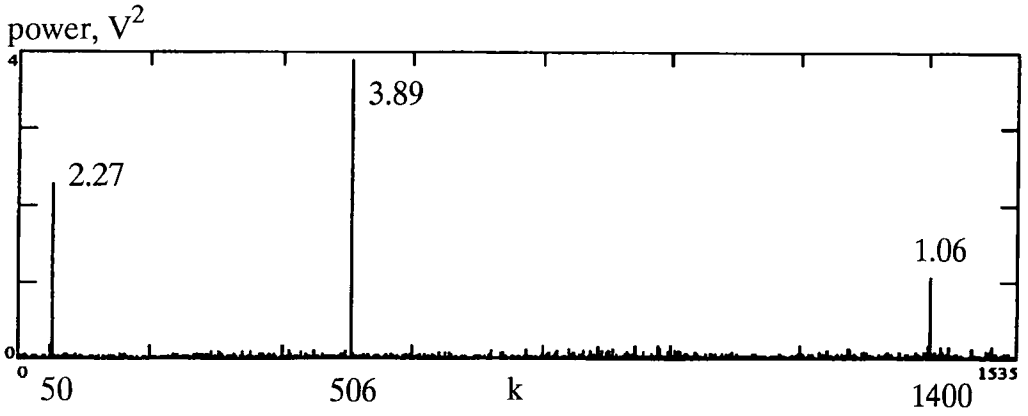


Fig. 6-8 The power spectrum of a signal x(t) = 1.5 cos (2π . 50 t) + 2 sin (2π . 506 t) + 1 cos (2π .1400 t) sampled by a.r.s. for 512 points in 1 second.

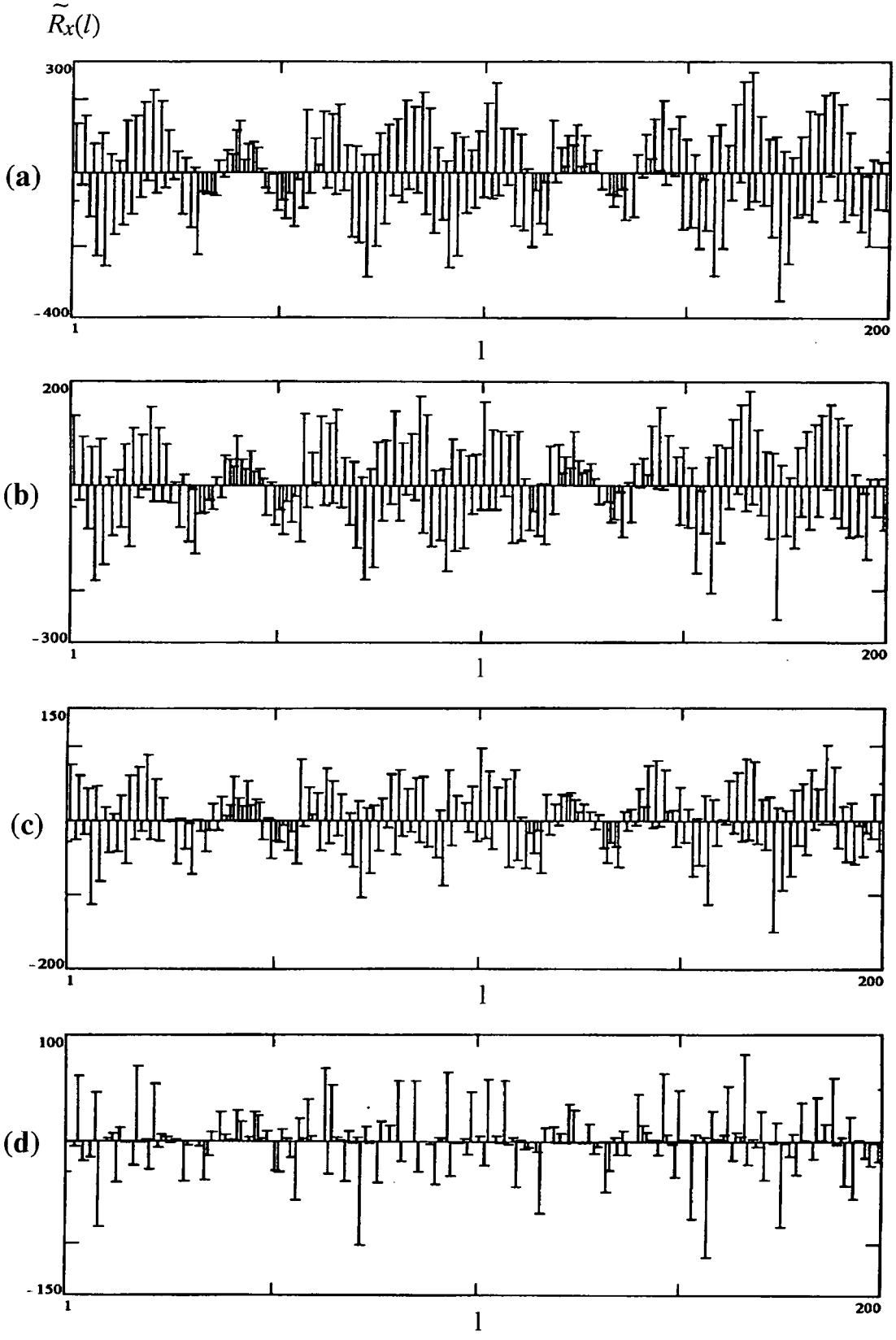


Fig 6-9 Parts of the auto-correlation results $\tilde{R}_x(l)$ in time domain of an a.r.s. sequence sampled for 512 points in 1 second. The step size for correlation is 976.5 μs, i.e. half of the average sampling period and the window widths are:

(a) ±15% of step size = ±146 μs , (b) ±10% of step size = ±97.6 μs,

(c) ±5% of step size = ±48.8 μs , (d) ±2.5% of step size = ±24.4 μs.

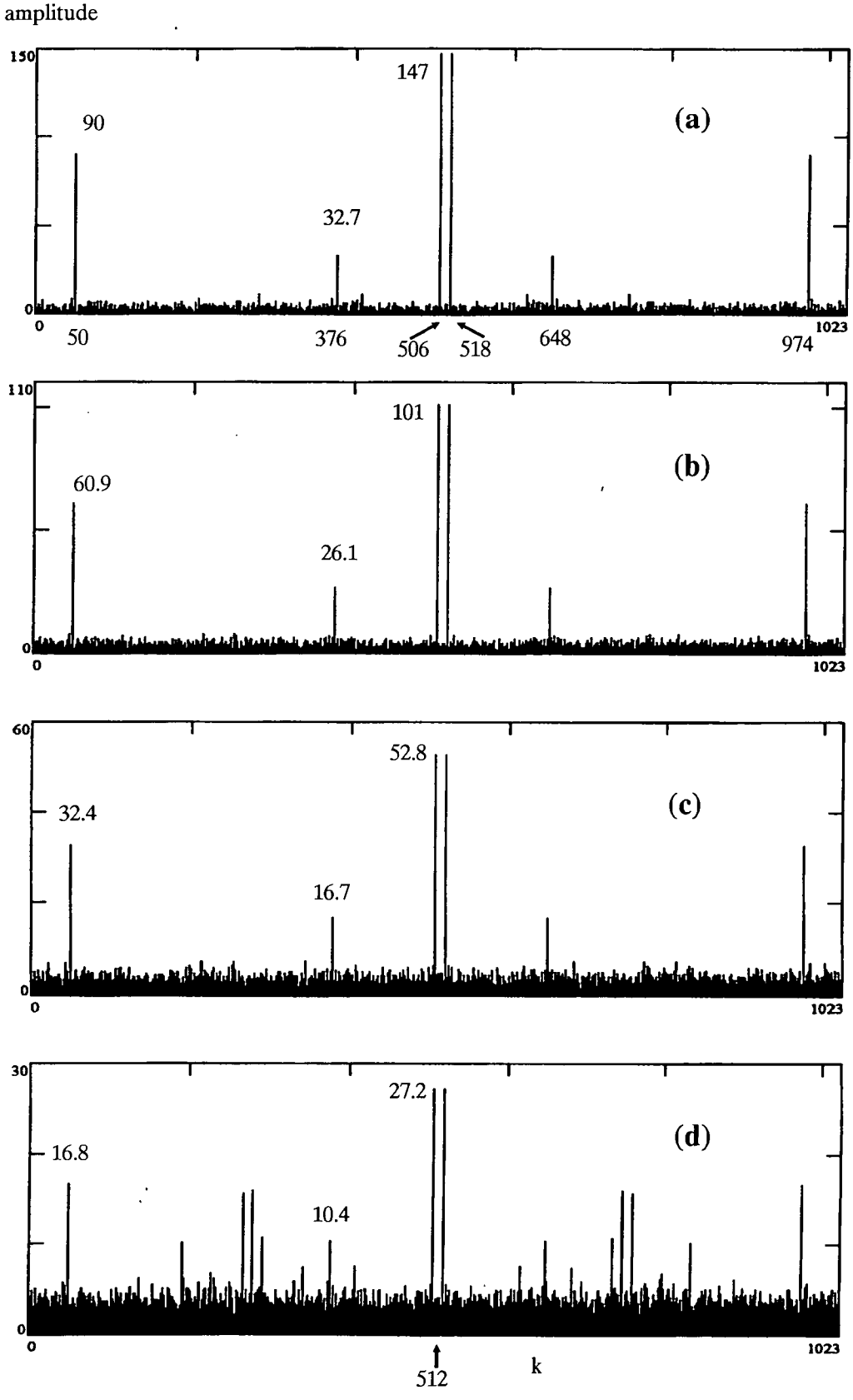


Fig. 6-10 The Fourier transforms corresponding to the auto-correlation sequences shown in Fig. 6-9 (a) to (d).

removed, but aliases at 1,542 and 1,998 are generated because of the change in Nyquist limit. In Fig. 6-12 (b), the Nyquist limit is at k = 2048 so that all aliases are removed and a spectrum similar to the one shown in Fig. 6-8 is achieved. The data of the above spectra are summarized in Table 6-1.

From the above discussion, we can conclude that a randomly sample sequence is alias-free so long as a proper evaluation method is adopted.

6.3.2 Programming Techniques : The auto-correlation results are computed by programming eqn (6-7a) with Microsoft C. In the equation, modulo arithmetic and searching are involved, both of which are time consuming. Two techniques may be used to speed up these operations.

To eliminate the usage of modulo arithmetic with floating point numbers, circular correlation is replaced by linear correlation using the concept of periodic extension of an aperiodic signal shown in Fig. 2-1. For illustration, Fig. 6-13 (a) shows a sequence of 4 points from which $\tilde{R}_x(1)$ is to be computed. The sequence is circularly shifted to the left by 1 step as shown in (b). Correlating the above two sequences yields

Table 6-1 : Auto-correlation results of a sequence sampled by a.r.s. for 512 points in 1 second using different step sizes and window widths.
(Mean sampling period is 1/512 = 1.95 ms.)

Spectrum in Fig.	Step size (μs)	Window width % of step	(μs)	Nyquist limit (Hz)	aliasing	component power ratio	S/N (dB)
6-10 (a)	976.5	± 15%	± 146	512	yes	2.7 : 1 : 4.5	19.3
6-10 (b)	976.5	± 10%	± 97.6	512	yes	2.3 : 1 : 3.9	17.6
6-10 (c)	976.5	± 5%	± 48.8	512	yes	1.9 : 1 : 3.2	14.7
6-10 (d)	976.5	± 2.5%	± 24.4	512	yes	1.6 : 1 : 2.6	11.8
6-12 (a)	488	± 20%	± 97.6	1024	yes	2.5 : 1 : 4.1	20.3
6-12 (b)	244	± 20%	± 48.8	2048	no	2.1 : 1 : 3.7	20.3

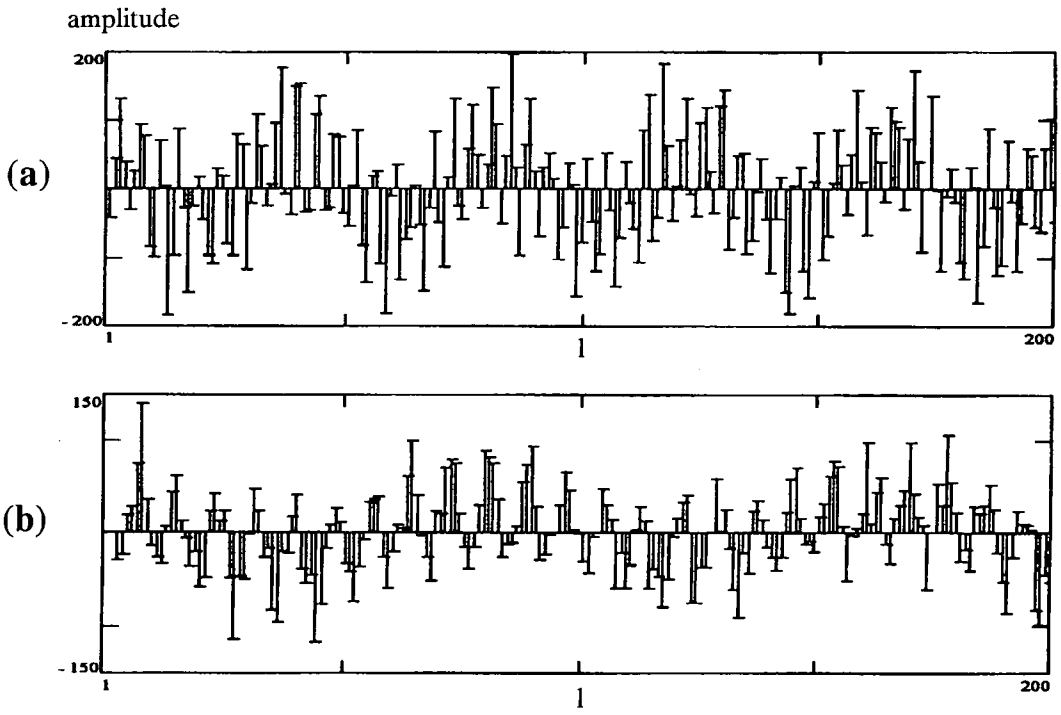


Fig. 6-11 Parts of the auto-correlation results of the same sample sequence in Fig. 6-9 but with different step sizes :

(a) step size = 488 μs, window width = ±97.6 μs , (b) step size = 244 μs, window width = ±48.8 μs

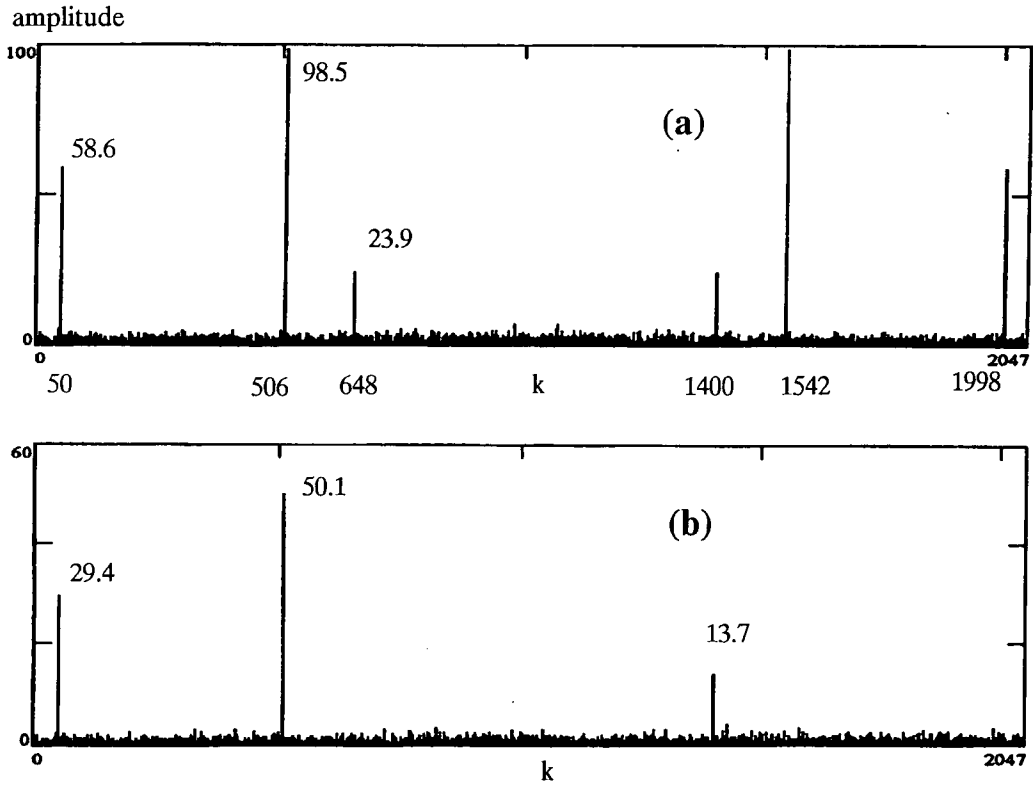


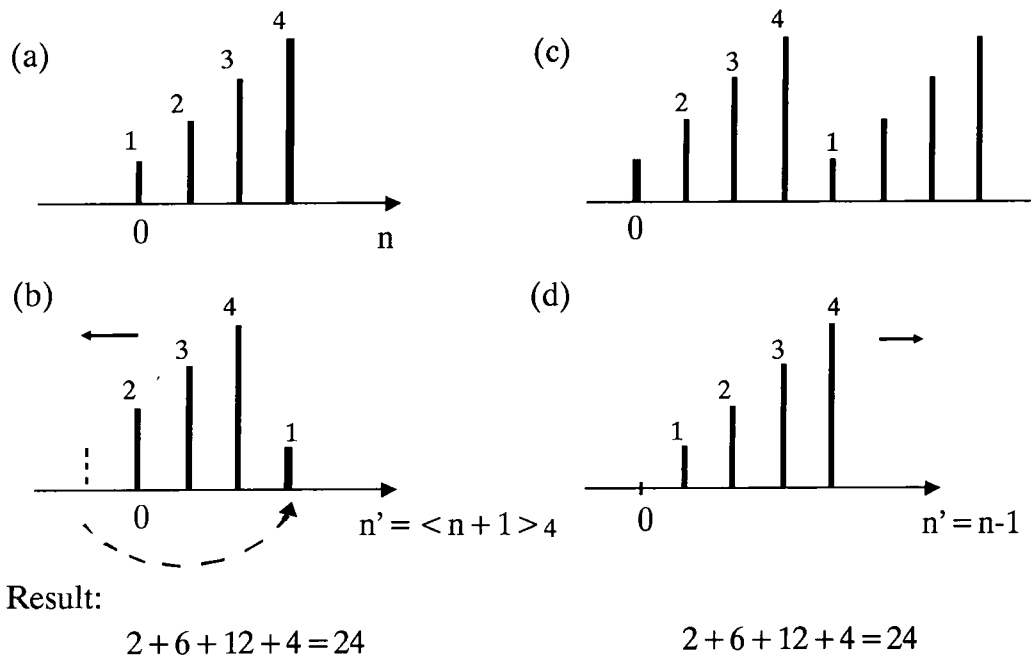
Fig. 6-12 The Fourier transforms corresponding to the auto-correlation sequences shown in Fig. 6-11 (a) and (b).

a result of 24. Suppose the sequence is now extended to two periods as shown in (c). When it is correlated with the sequence in (d), which is the original sequence linearly shifted to the right by one step, the result is the same as before. Hence the circular shift in eqn (6-7a) can be programmed as a linear shift as follows :

(i) Let $x(t_n)$ and t_n be the input data sequence and time sequence respectively. Assuming that the total sampling time ($N\mu$) is known, the input sequences can be extended to two periods as shown in Fig. 6-13 (c). In the second period :

$$x(t_{N+n}) = x(t_n) \quad \text{and} \quad t_{N+n} = t_n + N\mu$$

where n = 0,1,...,N-1.Denote these sequences by $\{x_{2N}(i)\}$ and $\{t_{2N}(i)\}$ respectively.



Result:

$2+6+12+4=24$ $\qquad$ $2+6+12+4=24$

Fig. 6-13 Circular correlation is replaced by an equivalent linear correlation:
(a) The original sequence x(n) is to be correlated with (b);
(b) $x(<n+1>_4)$, which is obtained by shifting x(n) to the left circularly one step.
(c) x(n) is extended to two periods and to be correlated with (d);
(d) x(n-1), which is x(n) shifted to the right linearly one step.
The results of the two correlations are identical.

(ii) Set $l = 0$.

(iii) The input sequences are modified by keeping the data values $x(t_n)$ but shifting the corresponding sampling times by :

$$t(j) = t_j + l.t_s$$

where j = 0,1,...,N-1 and t_s is the correlation step size.

(iv) Set $\tilde{R}_x(l) = 0$. For each t(j) search for a $t_{2N}(i)$ such that

$$|t(j) - t_{2N}(i)| \leq \frac{w}{2}, \quad \text{where w is the window width.}$$

If a match is found, compute $\tilde{R}_x(l) = \tilde{R}_x(l) + x(t(j))x_{2N}(i)$

(v) Increment l by 1and repeat steps (iii) and (iv) until $l = N$.

Note that step (i) is performed only once for the whole process.

In step (iv), there are potentially $2N^2$ searches to be performed for each l since there are N time values in the shifted sequence and 2N time values in extended sequence. For computing N correlations, the order of magnitude for searches is therefore N^3, which is undesirable. Fortunately all time values are stored in an ascending order, i.e. t(j + 1) > t(j), the index j can be treated as a pointer to minimize the number of searches. Let us specify a "distance" D, say, equal to one step size t_s. In the search process, $t_{2N}(i)$ is fetched as a target to be compared with t(j), where j is being incremented. When t(j) - $t_{2N}(i)$ is greater than D, we know that we have passed the target and the search should stop. Then we decrement j by 2 if j > 2, otherwise we set j to 0, to ensure that we do not pass the next target before the next search starts since we have set back the pointer. A new search begins by fetching the next target in $\{t_{2N}(i)\}$ with i incremented by 1 until i = 2N-1. After one value of $\tilde{R}_x(l)$ is computed,

the pointer j is reset to 0. By this method, only a few (about 2 or 3) searches are done per t(j) so that the order of magnitude is N for each l and N^2 for the whole process, which is one order lower than that of an exhaustive search.

The following is the search times spent in performing the auto-correlation with a 80486 machine having a 33-MHz clock. The programs are written in Microsoft C version 6 :

programming method	length N	search time (seconds)	comp. time (seconds)	overhead (seconds)
(a) with preset distance but no pointer	1,024	2,314	0.5	1.8
(b) with preset distance and pointer	1,024	8.8	0.5	1.8
(c) with preset distance and pointer	4,096	39	0.9	6.4

When a "distance" in time is set, the search for matches for a particular point stops when that distance is reached, but, without the pointer, the search will resume at the initial time for the next point. The average number of searches is thus N/2 per point, i.e. 512 per point in (a). With a pointer installed, about 2 or 3 searches are required per point. Indeed the ratio of the search time in (b) to that in (a) is 1 to 260. For computing 4,096 points, it takes, as shown in (c), a search period 4 times as long as in (b).

6.4 Auto-correlation of Parallel a.r.s. and Hybrid a.r.s.

In Chapters 4 and 5, the generation and the implementation of the parallel a.r.s. and the hybrid a.r.s., as well as their savings in computation, are fully described. In this section, the sequences sampled by these two methods will be analyzed by the auto-correlation approach developed above with an aim to confirm their anti-alias property. The principle is to check whether a valid auto-correlation sequence is

available (as described in section 6.3.1.1) when the step size is smaller than the sampling period. For consistency, the input signal used in this section, namely x(t) $= 1.5\cos(2\pi.50t) + 2\sin(2\pi.506t) + \cos(2\pi.1400t)$, is the same as the one in section 6.3.

6.4.1 Parallel a.r.s. : Let the input signal x(t) be sampled by a 32 x 32 parallel a.r.s. format (refer to section 4.3) yielding a data sequence $\{x(t_n)\}$ of 1024 points. This sequence $\{x(t_n)\}$ can, of course, be used directly to reconstruct a power spectrum, but here it is auto-correlated with a regular step size equal to 1/4 of the mean sampling period to generate an $\widetilde{R}_x(l)$, $l = 0,1,2,\ldots, 4097$. Part of the auto-correlation sequence and the power spectrum (computed by FFT) are shown in Fig. 6-14 (a) and (b), from which the characteristics of the parallel a.r.s. can be observed.

6.4.1.1 Bursts of Noise : Comparing to those auto-correlation sequences of a genuine a.r.s. shown in Fig. 6-9, one can easily observe that $\widetilde{R}_x(l)$ in Fig. 6-14(a) has 22 relatively large components from $l = 0$ to 88 at an interval of 4. To illustrate this point numerically, the first 9 values of $\widetilde{R}_x(l)$ are {3715, 12.45, -1.624, 6.04, -983.5, 5.08, 9.73, 8.06, 1606, . . . }, where the large values are underlined. This phenomenon is easily explicable if we recall that in the 32 x 32 parallel a.r.s. there are 32 groups of sampling points separated from each other by a random variable (see Fig. 4.1). Within each of these groups, there are 32 sampling points equally spaced in time. Since every 4 correlation steps correspond to one sampling period, a strong correlation is expected every 4 steps where overlaps occur certainly when the shift l is equal to the first 32 sampling periods . As the shift l progresses, randomness increases and we can see that the shape of $\widetilde{R}_x(l)$ begins to look similar to those in Fig. 6-11. In fact the " block effect"

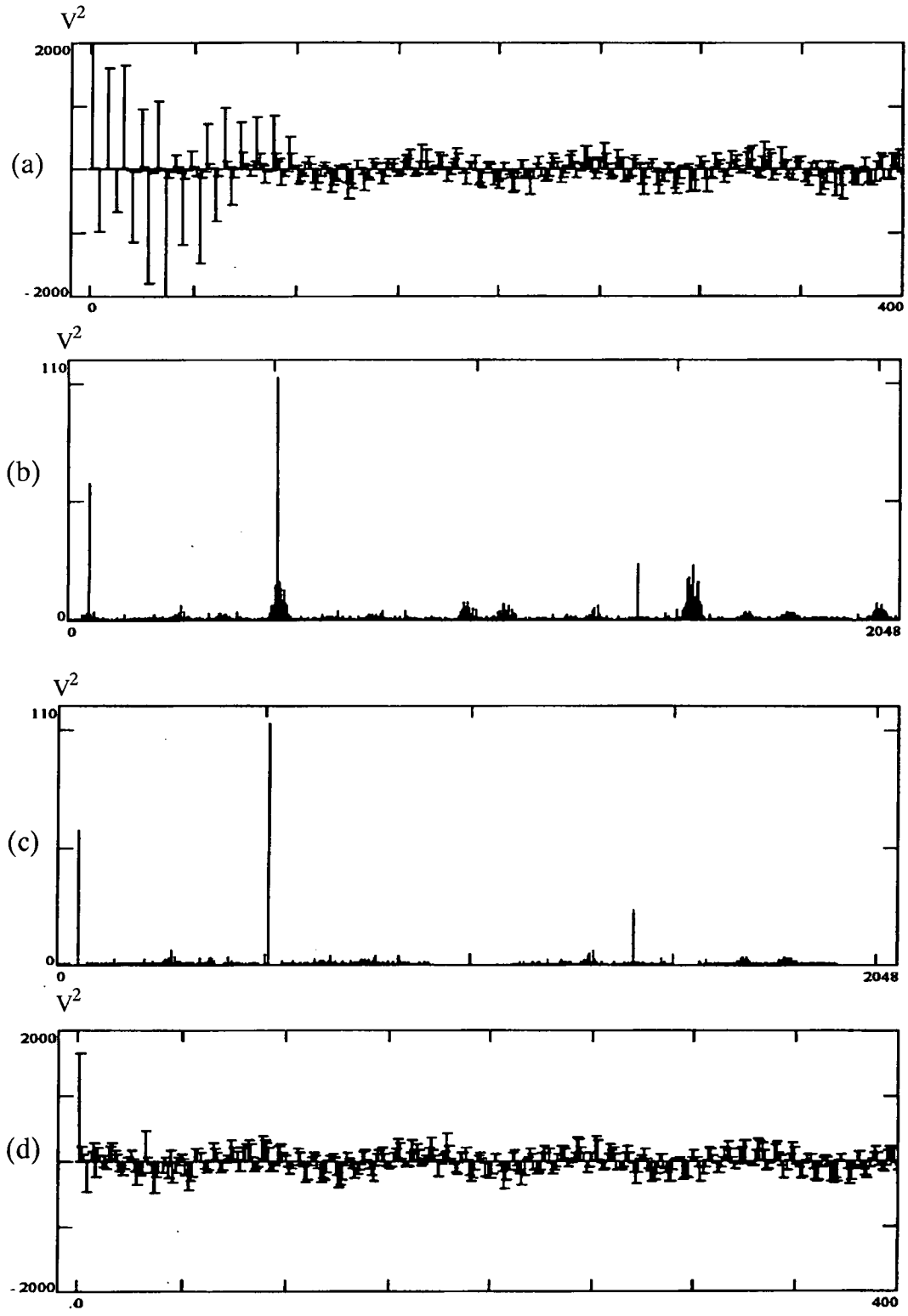


Fig. 6-14 :

(a) Part of the 4096-point auto-correlation result, $\tilde{R}_x(l)$, of a 1024-point data sequence sampled by a 32x32 parallel a.r.s.; step size of correlation = 1/4 of the mean sampling period and window width = ± 20% of the step size.

(b) The spectrum of $\tilde{R}_x(l)$.

(c) The spectrum of $\tilde{R}_x(l)$ with the prominent bursts of noise removed.

(d) Part of the sequence $\tilde{R}_y(l)$ which is the inverse DFT of (c) .

exists in the whole auto-correlation sequence but it is most prominent in the first 88 steps and the last 88 steps since $\widetilde{R}_x(l) = \widetilde{R}_x(N-l)$, where N is the sequence length.

Let us remove the prominent bursts of noise in the power spectrum and observe the effect in the auto-correlation sequence. The resulting spectrum is shown in Fig. 6-14(c) and its inverse DFT, $\widetilde{R}_y(l)$, is partly depicted in Fig. 6-14 (d). It is obvious that the large components in the original $\widetilde{R}_x(l)$ shown in Fig. 6-14 (a) are absent here, which is an evidence that the bursts of noise are generated by the blocks in the time domain.

The block effect in the auto-correlation sequence also lends itself to the explanation of the appearance of the bursts of noise seen in the power spectrum. Recall that a rectangular pulse in the time domain transforms to a sinc pulse (sine x/x) in the frequency domain, and the wider the pulse width is in time, the narrower in frequency and higher in amplitude is the main lobe of the sinc pulse . By analogy, the wider a block we have in the parallel a.r.s. (i.e. a larger value in p of eqn(4-1)), the narrower and higher the burst of noise in the spectrum. This effect is illustrated in Fig. 4-4.

To further demonstrate the block effect of the parallel a.r.s., let us suppress the first 21 large components and the last 21 components of the sequence, which are $\widetilde{R}_x(l)$ with $l = 4,8, \ldots, 88$ and $l = 4008, 4012, \ldots, 4092$, by 100 times to force them into the same order of magnitude of its neighbours. The resulting spectrum is shown in Fig. 6-15 (b), which, when compared to the original spectrum, has lower bursts of noise. Note that the total noise power is not reduced but spread out more evenly instead. The signal-to-noise ratios of both spectra are about 22 dB, but the peak noise

level in (a) is 22.6 V^2, whereas in (b) it is only 12.6 V^2. In (a) the ratio of the weakest component to peak noise is about 0.12 dB but in (b) the ratio increases to 2.53 dB.

6.4.1.2 Nyquist Limit : In section 6.4, it is pointed out that if a uniformly sampled data sequence is auto-correlated with a step size smaller than the sampling period, zeros (null information) will be found in a regular pattern inside the resulting sequence. For example, a regularly sampled sequence auto-correlated with a step size of 1/4 of the sampling period would yield a sequence $\{R_0,0,0,0, R_4,0,0,0, R_8, ...\}$ where R_i stands for non-zero values. Although the sequence length is increased by four times, the effective sampling period, which is the interval between the non-zero values, remains unchanged. Such an up-sampling by padding with zeros does not change the Nyquist limit [44].

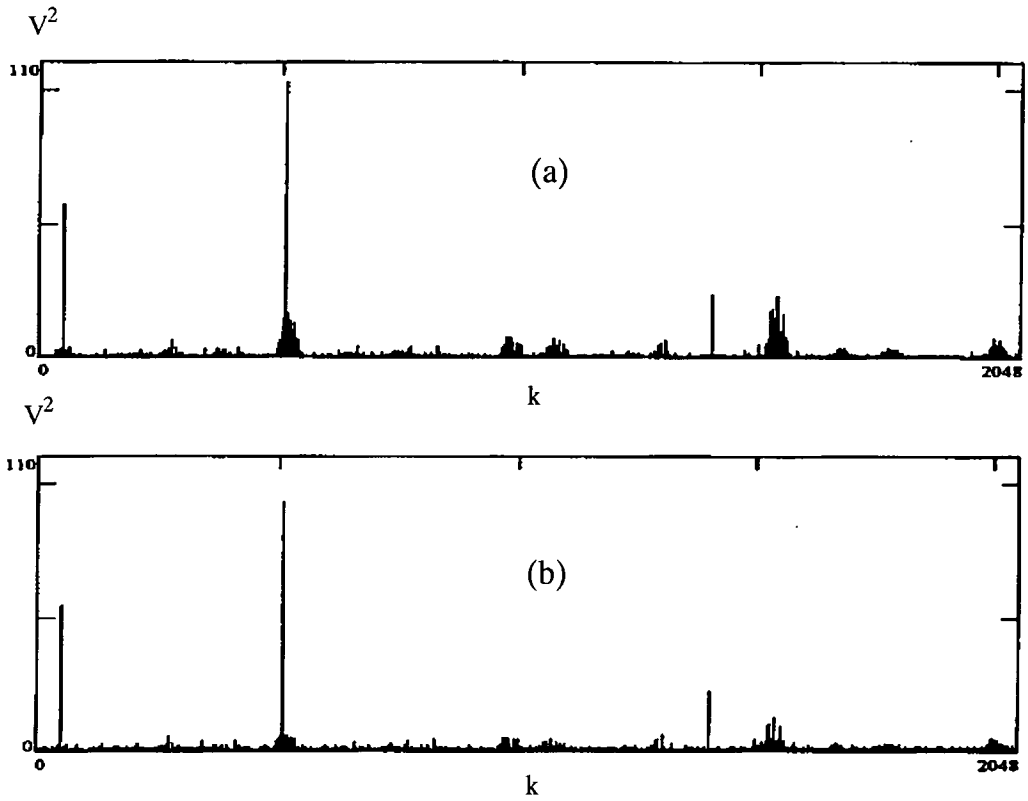


Fig. 6-15 :

(a)The original spectrum of $\tilde{R}_x(l)$, which is the same as Fig. 6-14 (b).

(b) Spectrum of $\tilde{R}_x(l)$ with the amplitude of the first 21 and last 21 large components reduced by 100 times.

Since the auto-correlation sequence $\tilde{R}_x(l)$ of the parallel a.r.s. contains 4,096 data regularly spaced in time, the Nyquist limit of its spectrum should be at k = 2048 *if its effective sampling period is indeed 1/4096.* Referring to Fig. 6-14 (a), in the first 88 steps, we do see a regular pattern of low values (effectively zeros) enclosed by large values in every interval of 4 steps. At about l = 100, however, the low values begin to grow in amplitude whilst the large components disappear. As a numerical illustration, let us compare the two auto-correlation sequences in Fig. 6-14 (a) and (d) for l = 200, 201, . . . , 209 :

$\tilde{R}_x(l)$: { . . . ,-13 8.5, -13.5, 100, 6.8, -91.2, -134.4, -343.8, -332.1, -67.67, -146, . . . },

and

$\tilde{R}_y(l)$: {. . . , -192.5, -6, 96.8, -3.41, -60.74, -140.5, -413.5, 269.3, -128.1, -122 , . . . }.

In $\tilde{R}_x(l)$ no fixed pattern of zeros is discernible. The values are also close to those in $\tilde{R}_y(l)$, which is a filtered version of $\tilde{R}_x(l)$. Observing from the simulation result, in the 4,096-point sequence $\tilde{R}_x(l)$, patterns of zeros appear only in a very small portion of the total sampling period, revealing that information is available most of the steps. Hence we can conclude that the effective sampling period is 1/4096 and the Nyquist limit is at k = 2048. The spectrum in Fig. 6-14 (b) in fact confirms this assertion; the component at k = 1,400 can be clearly seen.

6.4.2 Hybrid a.r.s. : To study the properties of the hybrid a.r.s., an input signal, x(t) = $1.5\cos(2\pi.50t) + 2\sin(2\pi.506t) + \cos(2\pi.1400t)$, is sampled by a 4 x 256 hybrid a.r.s. sequence and an 8 x 128 hybrid a.r.s. for 1,024 points (refer to section 5.2). The resulting sequences are auto-correlated with a step size of 1/4 of the mean sampling period and a window width equal to ±20% of the step size. The auto-correlation

sequences, together with its Fourier transforms, are shown in Fig. 6-16 and Fig. 6-17 respectively.

6.4.2.1 Binning of Noise : By scrutinizing the auto-correlation sequence from the 4 x 256 hybrid a.r.s. in Fig. 6-16 (a), we can observe that the overall shape of this sequence is similar to the auto-correlation sequences of data sampled by genuine a.r.s. (shown in Fig. 6-11 but with a different scale) except that there is a large value at $l = 2{,}048$, which is the mid-point of $\tilde{R}_x(l)$. This value, $\tilde{R}_x(2{,}048)$, is as large as $\tilde{R}_x(0)$. Recall that the timing of the second half section of this hybrid a.r.s. sequence is equivalent to the first half plus 1/2 if the total sampling period is unity (see eqn 5-2) ; therefore when the shift l of the correlation reaches half of the total sampling period, a total match in timing occurs as at $l = 0$ so that a large correlation result is obtained. These components at $l = 0$ and at $l = 2{,}048$ are part of a pulse train of a periodicity of 1/2. Thus in the reconstructed power spectrum, we expect to see the effect from this pulse train. In Fig. 6-16(c), it can be seen that the amplitudes of the background noise at the even frequency indices are substantially larger than those at the odd frequency indices, which means that noise tends to gather at the even indices. In fact, if the frequency of the input signal is f, noise will cluster at indices congruent to f modulo 2. As all the components of the input signal in this case have even frequency indices, the background noise accumulates at the even indices. To show that this binning effect is attributed to $\tilde{R}_x(2{,}048)$, let us set this value to 0 and perform the DFT again. The resulting spectrum in Fig. 6-16 (d) shows that noise is redistributed into the odd indices making the peak noise level lower than that in (c).

All other formats of hybrid a.r.s., e.g. 8 x 128, exhibit this binning of noise in the frequency spectrum. For the 8 x 128 format, the whole sampling period is divided

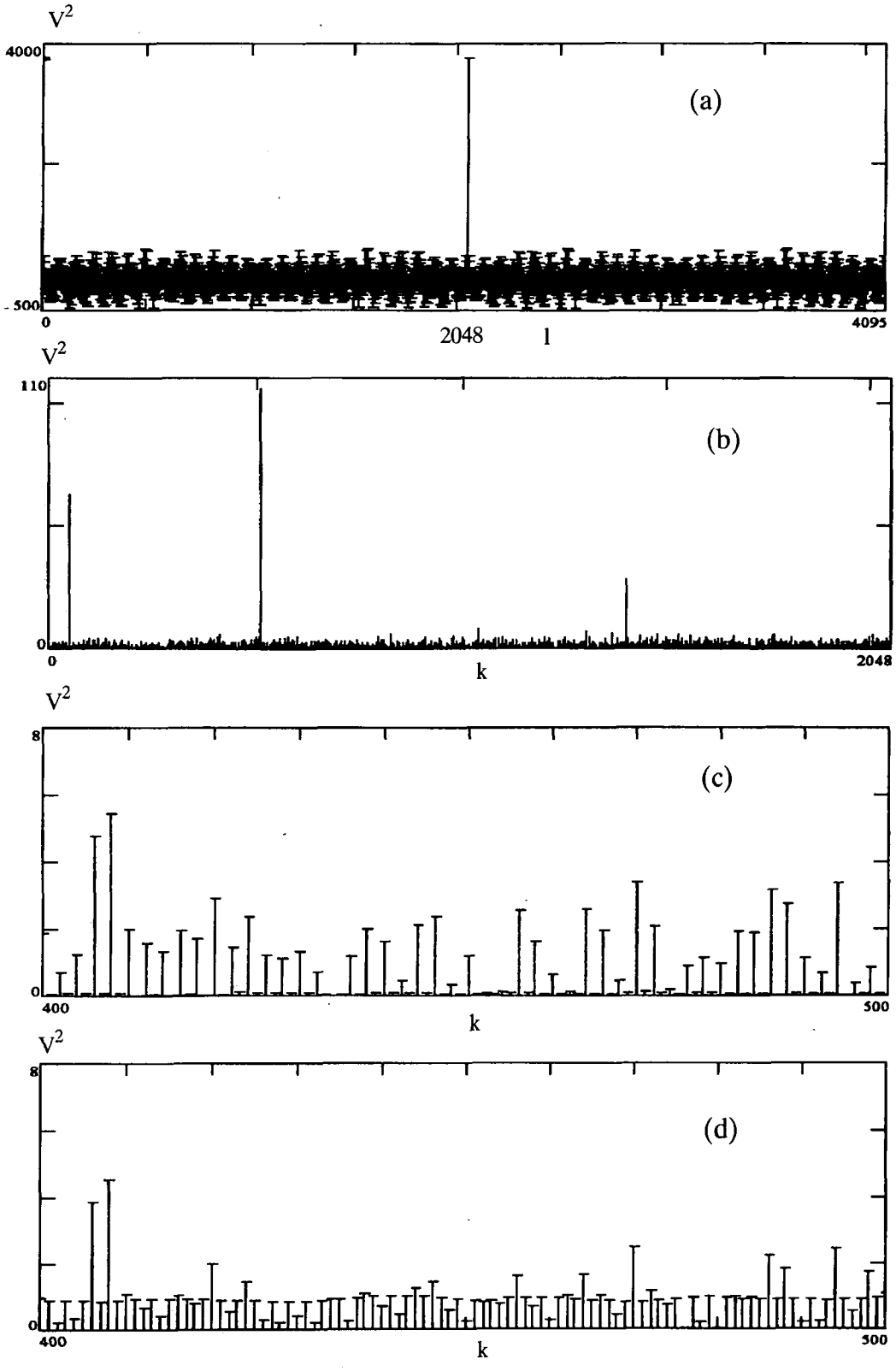


Fig. 6-16 : (a) The auto-correlation result, $\tilde{R}_x(l)$, of a 1024-point data sequence sampled by a 4 x 256 hybrid a.r.s.; step size of correlation = 1/4 of the mean sampling period and window width = ± 20% of the step size.

(b) The spectrum of $\tilde{R}_x(l)$.

(c) Part of the spectrum in (a). Noise tends to cluster at even indices.

(d)Part of the spectrum of the same sequence in (a) with $\tilde{R}_x(2{,}048)$set to 0. Noise is spread out more evenly.

into four sections (see eqn 5-3). The timing of the second, third and fourth sections are the same as the first if the timing in each section is referred to the beginning of the respective sections. Hence for the 4,096-step auto-correlation, total match in timing is expected when the shift l reaches 1,024, 2,048 and 3,072 . These three conspicuous values can be seen in Fig. 6-17(a). (Note that the phase reversal of $\widetilde{R}_x(1{,}024)$ and $\widetilde{R}_x(3{,}072)$ is attributed to the phase change of the input signal.) Together with $\widetilde{R}_x(0)$ these three components form part of a pulse train having a frequency of 4. Therefore, for an input frequency f, noise will cluster at indices congruent to f modulo 4. Since our input components have frequencies at 56, 506 and 1,400, noise is expected to accumulate at k = 0 and 2 modulo 4, which are even integers (see Fig. 6-17 (c)). Two input components at 56 and 506 are, however, congruent to 2 modulo 4. Thus noise at these indices are relatively stronger. If the component at 1,400 is changed to 1,402, then all the indices of the components are 2 modulo 4 and noise will accumulate at only one index per period of 4, which is shown in Fig. 6-18. When $\widetilde{R}_x(1{,}024)$, $\widetilde{R}_x(2{,}048)$ and $\widetilde{R}_x(3{,}072)$ are all set to 0, noise in the frequency spectrum is redistributed more evenly as shown in Fig. 6-17 (d).

Filtering the extra large components in $\widetilde{R}_x(l)$ does not improve the overall signal-to-noise ratio of the power spectra, but only reduces the peak noise level. The signal-to-noise ratios of the 4 x 256 or 8 x 128 hybrid a.r.s. in Fig. 6-16 and 6-17, whether filtered or not, are about 23 dB. After filtering, for the 4 x 256 format, the peak noise level is reduced from 8 V^2 to 7 V^2, making the ratio of the weakest signal to peak noise increase from 5.5 dB to 5.9 dB. For the 8 x 128 format, the peak noise level is reduced from 14.8 V^2 to 12.7 V^2, making the ratio of the weakest signal to peak noise increase from 2.5 dB to 3.3 dB.

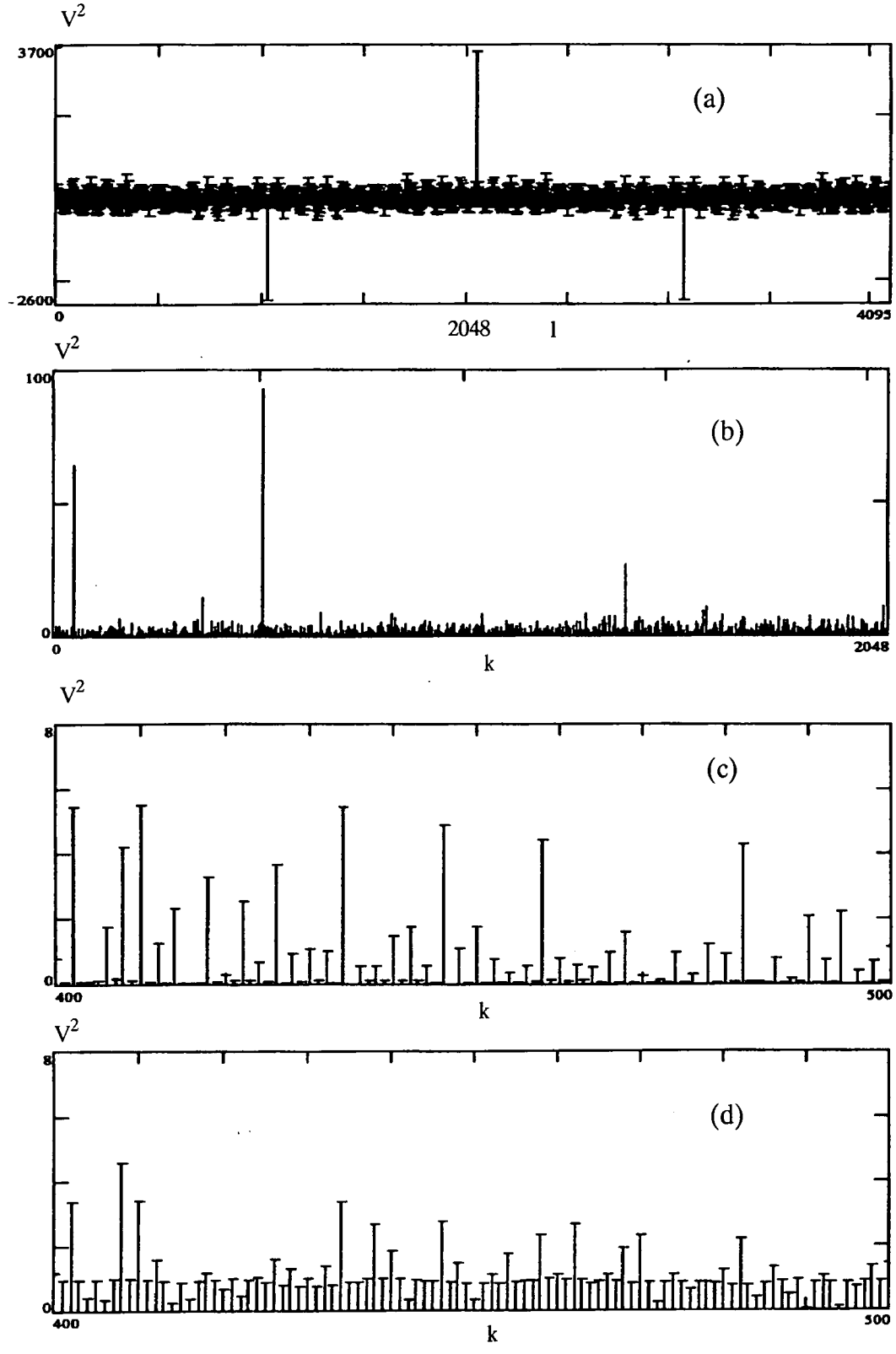


Fig. 6-17 : (a) The auto-correlation result, $\tilde{R}_x(l)$, of a 1024-point data sequence sampled by an 8 x 128 hybrid a.r.s.; step size of correlation = 1/4 of the mean sampling period and window width = ± 20% of the step size. (b) The spectrum of $R_x(l)$.(c) Part of the spectrum in (a); noise tends to cluster at indices congruent to 0 and 2 mod 4. (d)Part of the spectrum of the same sequence in (a) with $R_x(1,024)$, $R_x(2,048)$ and $R_x(3,072)$ set to 0. Noise is spread out more evenly.

6.4.2.2 Nyquist Limit : Similar to the analysis in section 6.4.1.2, if the effective sampling period of a 4,096-point auto-correlation sequence generated by the hybrid a.r.s. is 1/4,096, then the Nyquist limit is at k = 2,048. Although there are large components at the beginning of each section, the auto-correlation sequences of both 4 x 256 and 8 x 128 formats contain no patterns of zeros at all. Parts of $\widetilde{R}_x(l)$ from the 4 x 256 and 8 x 128 hybrid a.r.s are shown in Fig. 6-19 (a) and (b), and the numerical values of $\widetilde{R}_x(l)$ with l = 10 to 20 are listed below :

For 4 x 256 hybrid a.r.s. (Fig. 6-19 (a)) :

{. . . 101.4, -38.2, -43.4, -176, 42.6, 192.1, 149.8, 28.2, 69.5, -208.1, -71.95 . . .}

For 8 x 128 hybrid a.r.s. (Fig. 6-19 (b)) :

{. . . 96.1, -29.4, -67.3, -181.9, 37.8, 103.9, 130.4, 9.4, 29.5, -181.1, -126.2 . . .}

From Fig. 6-19 and the values of $\widetilde{R}_x(l)$ listed above, we can conclude that information is available at every step of the auto-correlation; thus the Nyquist limit is at k = 2,048.

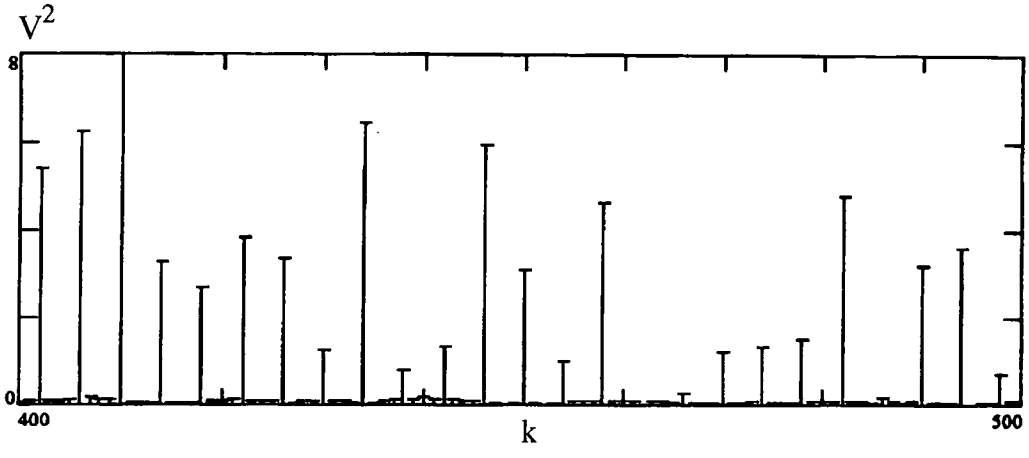


Fig. 6-18 Part of the power spectrum of an input x(t) = 1.5 cos(2π. 50t) + 2 sin(2π. 506t) + cos(2π. 1402t) reconstructed from the auto-correlation sequence of an 8 x 128 hybrid a.r.s. Noise clusters at indices congruent to 2.mod 4.

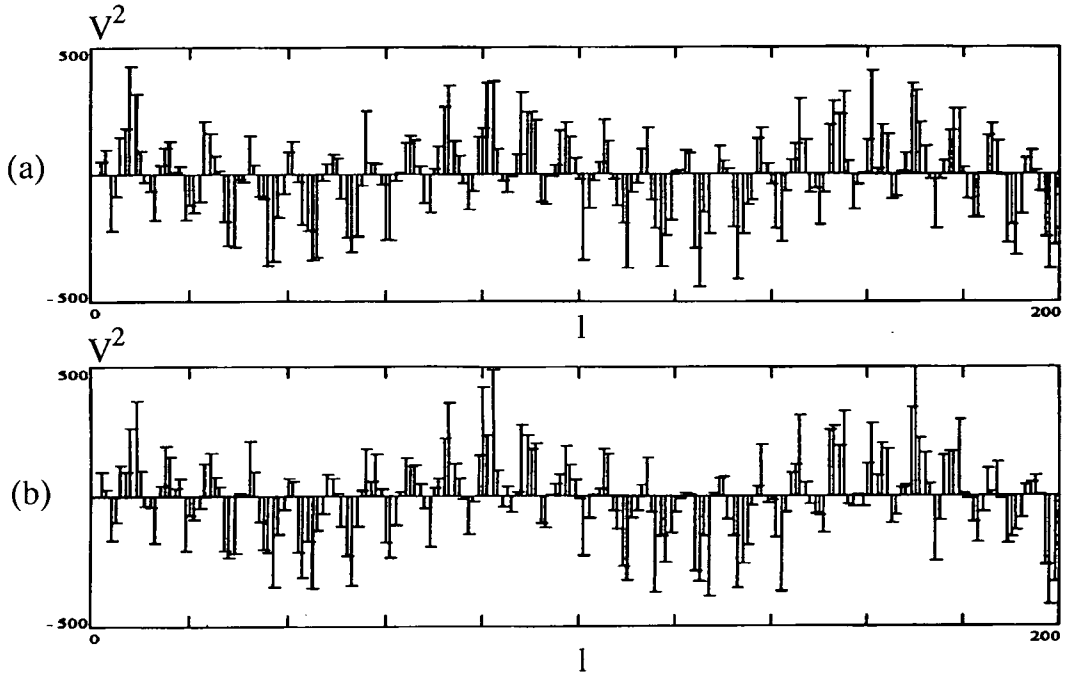


Fig. 6-19 :

(a) Part of the auto-correlation sequence in Fig. 6-16(a) generated from the 4 x 256 hybrid a.r.s.

(b) Part of the auto-correlation sequence in Fig. 6-17(a) generated from the 8 x 128 hybrid a.r.s.

In their respective spectra shown in Fig. 6-16 (b) and 6-17 (b), the component at k = 1,400 can be unambiguously recognized.

6.5 Concluding Remarks

Since the auto-correlation of a sequence is related to its power spectrum, the anti-alias property of a randomly sampled sequence can be studied and explained via its auto-correlation sequence. For a regularly sampled sequence, the step size of its auto-correlation cannot be smaller than its sampling period ; otherwise null information (zero values) is obtained between two consecutive points of the sampling grid. The randomly sampled sequence, however, is a pseudo-continuous signal, and it holds information (with a certain probability) along the whole sampling period.

Hence, in theory, the step size of the auto-correlation can be reduced to an arbitrarily small interval so as to squeeze out the information of the high frequency components. In section 6.3, the effect of the step size on extending the Nyquist limit is illustrated.

Based on the above theory, the auto-correlation of the parallel a.r.s. and the hybrid a.r.s. are studied in section 6.4. Apart from some large values, the auto-correlation sequences derived from both methods contain information in every step even if the step size is smaller than the mean sampling period, thus confirming their anti-alias characteristics. The large values in these auto-correlation sequences are related to noise in their frequency spectrum. Eliminating these large values in the time domain does not improve the overall signal-to-noise ratio of the spectrum, but redistributes the noise more evenly within the spectrum so that the peak noise level is reduced. If a weak signal component exists in the spectrum, it may be desirable to lower the peak noise level to make the signal more conspicuous.

In general, using the DFT to reconstruct a spectrum of a randomly sampled sequence is more direct than using the auto-correlation method. Assuming that an N-point real sequence is obtained by random sampling, it takes N^2 real multiplications ($= N^2/16$ complex multiplications) for evaluating its spectrum by the DFT and another N complex multiplications for computing their power, i.e. a total of $N^2/16 + N$ complex multiplications. For auto-correlation, $N^2/16$ complex multiplications are required for computing the auto-correlation sequence and $N \log_2 N$ complex multiplications are required for transforming the sequence to the frequency domain by the FFT, i.e. a total of $N^2/16 + N \log_2 N$ complex multiplications. Although the order of magnitude of computation is the same for both methods, the auto-correlation method needs extra time in searching for overlaps. With the auto-correlation program

written in C, both the computation and the search process are implemented by using the mathematical functions of the C library. It turns out that the search time dominates the whole process time, making the computational time a negligible factor. In section 6.3, we can see that the search time is 39 seconds for a 4,096-point auto-correlation while the computational time is less than a second.

The auto-correlation method, however, may be considered a means to convert a randomly sampled sequence to a regularly sampled sequence since the resulting auto-correlation sequence is regularly spaced in time. If a randomly sampled sequence is so "pre-processed" once, the timing information of the sampling points may be discarded and the FFT may be used to compute the power spectrum. In some cases where correlation is desired, e.g. checking similarity between two randomly sampled signals, the techniques introduced in this chapter can be adopted.

CHAPTER 7

RAPID EVALUATION OF SPECTRUM

7.1 Introduction

After an N-point data sequence x(n) is obtained by sampling, its frequency spectrum can be reconstructed by the DFT :

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{nk}, \quad k = 0,1,\ldots,N-1 \tag{7-1}$$

where $W_N^i = \exp(-j2\pi i/N)$. It is well known that for the above evaluation, the complexity of computation is N^2. For general-purpose computers manufactured before the nineties, the computational time required to perform a multiplication is significantly longer than an addition (e.g. a 8086 microprocessor takes a few clock cycles to do an ADD but over 100 clock cycles to do a MUL). To speed up the computation, fast algorithms were devised with a target to reduce as many multiplications as possible, even if in exchange more additions had to be performed. Among those fast algorithms, radix-2 FFT is the one most commonly used. Nowadays, however, microprocessors (like 80486 and Pentium 5) can perform floating-point multiplication as fast as floating-point addition. Hence overall optimization in both multiplication and addition must be considered.

In eqn (7-1), W_N^{nk} is the kernel of the transform, which, if programmed in a high-level language, requires the use of trigonometric functions in the library. Suppose the accuracy of X(k) can be sacrificed for the sake of computational speed, the kernel

may be approximated in value such that multiplications required can be reduced to a small number or even totally eliminated. This approach of optimization is completely different from exploiting the symmetry property of the kernel, which is adopted by all conventional fast algorithms discussed in Chapter 2.

Considering all factors, the FFT may still be a more appropriate algorithm than the approximate method for evaluating a uniformly sampled sequence. However, when the sequence is randomly or irregularly sampled, the FFT cannot apply and the approximate method will become an attractive choice to speed up the computation.

7.2 Approximate Fourier Transforms

7.2.1 Coarse Quantization of the Kernel

7.2.1.1 Basic Principle: The kernel of the DFT, W_N^{nk}, is composed of a set of orthogonal sine and cosine basis functions. Let us consider the case when k = 1; X(1) is evaluated by multiplying the input sequence by cos (2πn/N) and sin (2πn/N). Suppose the sequence length N is 1,024, then there are 1,024 different values (if the sign is also considered) for the sine and cosine functions respectively. By representing values lying within a range by their mean, which is basically a clustering or re-quantization, fewer values remain and the computational load can also be reduced.

Let the magnitude of cos (2πn/N) and sin (2πn/N) be divided into m levels and denoted by c(m) and s(m) respectively. If, for example, the positive range of s(m) is represented by five levels, we may have

$$s(m) = \begin{cases} = 0 & \text{for } 0 \le s(m) < 0.125 \\ = 0.25 & \text{for } 0.125 \le s(m) < 0.375 \\ = 0.5 & \text{for } 0.375 \le s(m) < 0.625 \\ = 0.75 & \text{for } 0.625 \le s(m) < 0.875 \\ = 1 & \text{for } 0.875 \le s(m) \le 1 \end{cases}$$

which is illustrated in Fig. 7-1. Similarly, the negative range of s(m) can also be so represented, yielding a 9-value representation scheme. Given a fixed sequence length N, the arc-sine of the quantization levels in s(m) can be evaluated to find the corresponding value in n. For N = 1,024, we obtain for the above scheme :

$$s(n) = \begin{cases} = -1 & \text{for } 682 < n < 850 \\ = -0.75 & \text{for } 662 < n \le 682 \text{ or } 850 \le n < 914 \\ = -0.5 & \text{for } 575 < n \le 622 \text{ or } 914 \le n < 961 \\ = -0.25 & \text{for } 532 < n \le 575 \text{ or } 961 \le n < 1004 \\ = 0 & \text{for } 0 \le n < 20 \text{ or } 492 < n \le 532 \text{ or } 1004 \le n < 1024 \\ = 0.25 & \text{for } 20 \le n < 63 \text{ or } 449 < n \le 492 \\ = 0.5 & \text{for } 63 \le n < 110 \text{ or } 402 < n \le 449 \\ = 0.75 & \text{for } 110 \le n < 174 \text{ or } 338 < n \le 402 \\ = 1 & \text{for } 174 \le n \le 338 \end{cases}$$

(7-2)

It is obvious that c(n) can be represented in a similar way. When X(1) is to be evaluated, each element of {x(n)} is sorted according to the values of s(n) and c(n) it is to be multiplied, which can be achieved simply by looking at the values of n as

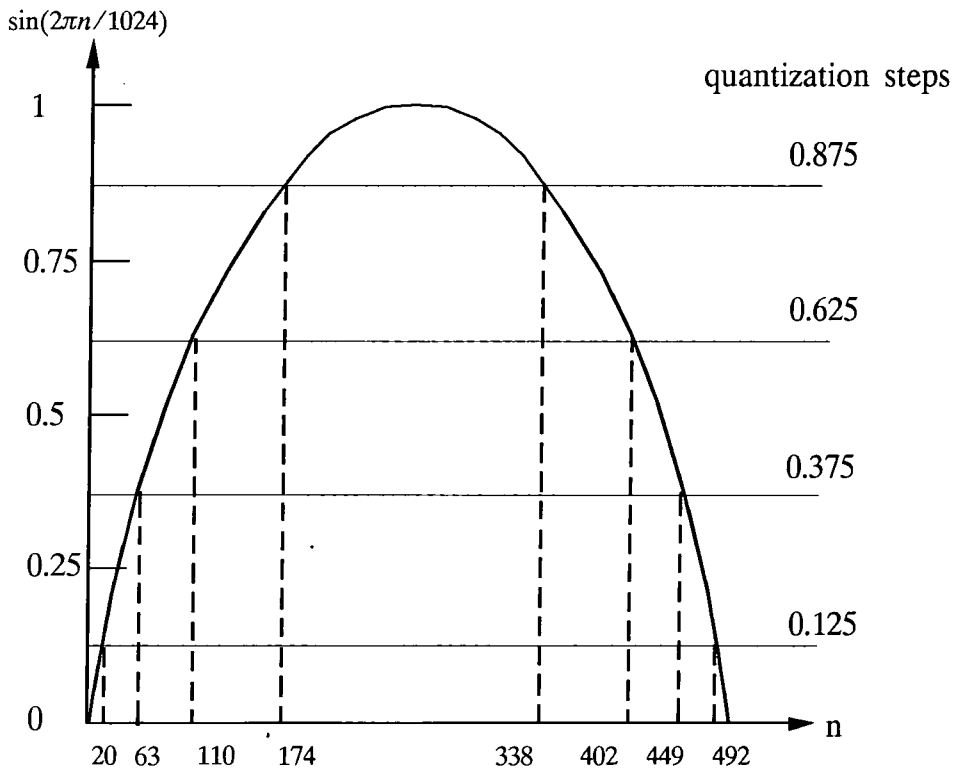


Fig. 7-1 Representing the magnitude of the positive half cycle of a sine function by 5 values, namely, 0, 0.25, 0.5, 0.75 and 1.

specified in eqn(7-2) for s(n) and a similar equation for c(n). The real and imaginary parts of X(1) are respectively given by :

$$Real\{X(1)\} = \sum_{i \in n} \left(\sum_{n \in d_i} x(n) \right) c(i)$$

and

$$Imag\{X(1)\} = \sum_{i \in n} \left(\sum_{n \in e_i} x(n) \right) s(i)$$

where i is one of the levels specified by n, and d_i and e_i are the groups formed according to c(i) and s(i) respectively. Fig. 7-2 shows the block diagram for the computation, in which only 2 groups of x(n) are depicted. As multiplications by 0, 1 and -1 are trivial, there are only 6 non-trivial multiplications required for a 9-level kernel. If the sums of x(n) corresponding to s(n) = -m and c(n) = -m are first subtracted from those corresponding to s(n) = m and c(n) = m respectively before doing the multiplications, there are only 3 multiplications required for computing X(1).

To compute X(k) other than k = 1, the same sorting procedure can apply with k regarded as a "scaling factor". Recall that sin (2πnk/N), with a period of N/k in the time domain, is a compressed version of sin (2πn/N). Hence for a kernel W_{1024}^{nk} :

$$s(nk) = \begin{cases} = -1 & \text{for } 682 < nk < 850 \\ = -0.75 & \text{for } 662 < nk \le 682 \text{ or } 850 \le nk < 914 \\ = -0.5 & \text{for } 575 < nk \le 622 \text{ or } 914 \le nk < 961 \\ = -0.25 & \text{for } 532 < nk \le 575 \text{ or } 961 \le nk < 1004 \\ = 0 & \text{for } 0 \le nk < 20 \text{ or } 492 < n \le 532 \text{ or } 1004 \le n < 1024 \\ = 0.25 & \text{for } 20 \le nk < 63 \text{ or } 449 < nk \le 492 \\ = 0.5 & \text{for } 63 \le nk < 110 \text{ or } 402 < nk \le 449 \\ = 0.75 & \text{for } 110 \le nk < 174 \text{ or } 338 < nk \le 402 \\ = 1 & \text{for } 174 \le nk \le 338 \end{cases}$$

(7-3)

where nk ≡ nk modulo N because the period of s(nk) in eqn (7-3) is now scaled to the sequence length N, which is 1,024 in this example.

7.2.1.2 Computational Complexity : Consider that the data sequence x(n) is sampled regularly in time. With the approximate method, there are only a few complex multiplications for evaluating one frequency component. In the above example of a 9-level kernel, the actual multiplications required are only 3 per frequency component. In general, therefore, the total number of multiplications required is u.N, where u is a small integer. As compared to the FFT whose complexity is $N\log_2 N$, both algorithms have a linear complexity, but the number of multiplications required by the approximate method can even be smaller by making $u << \log_2 N$. The kernel may also be rounded off to the nearest integer power of 2, thus making the multiplications merely binary shifts, which can be effectively implemented by hardware or low-level languages.

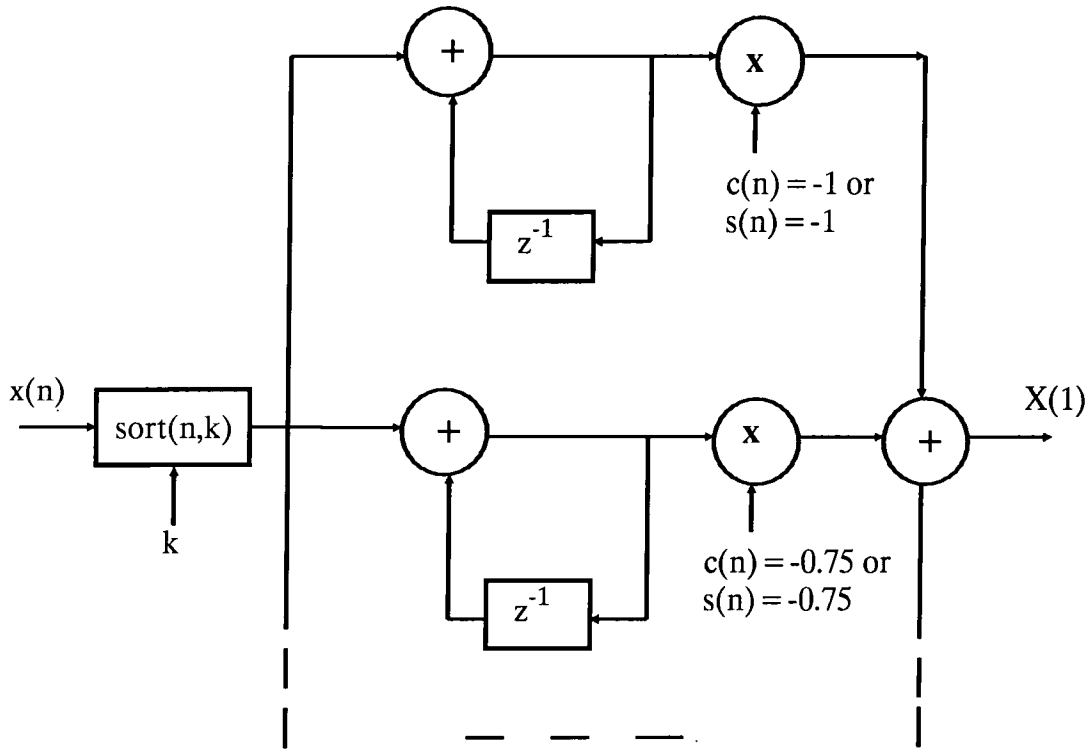


Fig. 7 - 2 Block diagram of the approximate evaluation of X(k) by grouping x(n) according to the values of c(n) and s(n).

In principle N-1 additions are required for computing each frequency component, although those data points to be multiplied by 0 need not be added at all. Thus the complexity for addition is N^2, which is significantly higher than that of the FFT. This approximate method could be faster than the FFT if both programs were run on a computer which took a much longer interval in executing multiplication than addition.

Suppose the data sequence is not uniformly sampled, then the FFT is not applicable and the computational complexity is N^2 for both multiplication and addition. It is obvious that, with the approximate method being used, computational effort will be significantly saved in multiplication, and even for addition, it is also slightly lessened. The problem remaining to be considered is the trade-off in the accuracy of the recovered spectrum.

7.2.2 Three-level Kernel

7.2.2.1 Basic Principle : In the previous section, the kernel of the transform is rounded off to a few levels. Obviously, the computational speed increases as the number of levels in the kernel decreases. This rounding off can get coarser and coarser until ultimately two levels, i.e. + 1 and -1 are left, which is equivalent to a set of rectangular waves being used as the kernel [4]. Multiplications by ± 1 are trivial; hence only additions and subtractions are required for the computation. A penultimate round off to 3 levels (+ 1, -1 and 0) corresponding to addition, subtraction and 'do-nothing' was proposed by Mason[5]. The three-level round-off was reported to have less leakage than the two-level truncation while maintaining the computational effort substantially the same as the latter. In fact it could even be marginally less because

there is a "do-nothing" that requires no operation at all. Fig. 7-3 shows how a sine function is quantized to three levels.

7.2.2.2 Leakage and Amplitude Error : Fig. 7-4 shows a typical truncated sine function, sal, and its corresponding cosine counterpart, cal. (The terms sal and cal are borrowed from Hughes and Herron [47].) The truncation level to generate these functions is $\sqrt{2} = 0.707$, which corresponds to a "cut-in angle" of 0.25π. Note that in this case, the duration for the occurrence of 1 is equal to that of 0. If the truncation level is lowered, the duration for the 1 will be lengthened.

An interesting property of these sal and cal functions is that they keep the geometric symmetry of the sine and cosine functions, which can be clearly seen in Fig. 7-4. Hence a set of "almost orthogonal"[1] basis functions can be generated by sal(kθ) and cal(kθ), where k is an integer. Consequently, when these functions are used as a transform kernel, the results retain the frequency characteristics of the DFT except

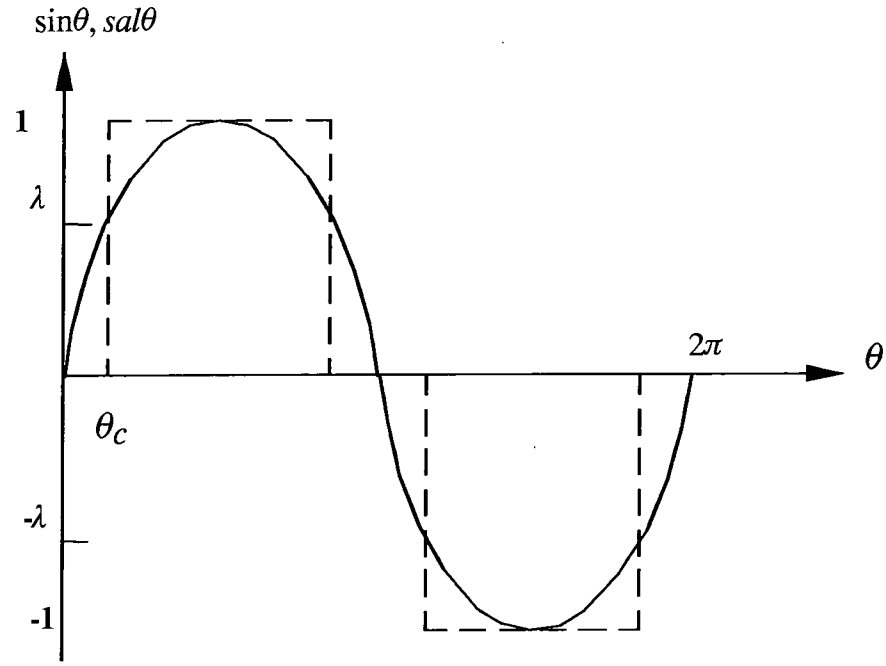


Fig. 7-3 Quantization of the sine function. λ is the truncation level and θ_C is the cut-in angle.

1. The functions sal(kθ) and cal(kθ) are orthogonal in its own right. When they are, however, used as substitutes for sin(kθ) and cos(kθ), orthogonality is lost.

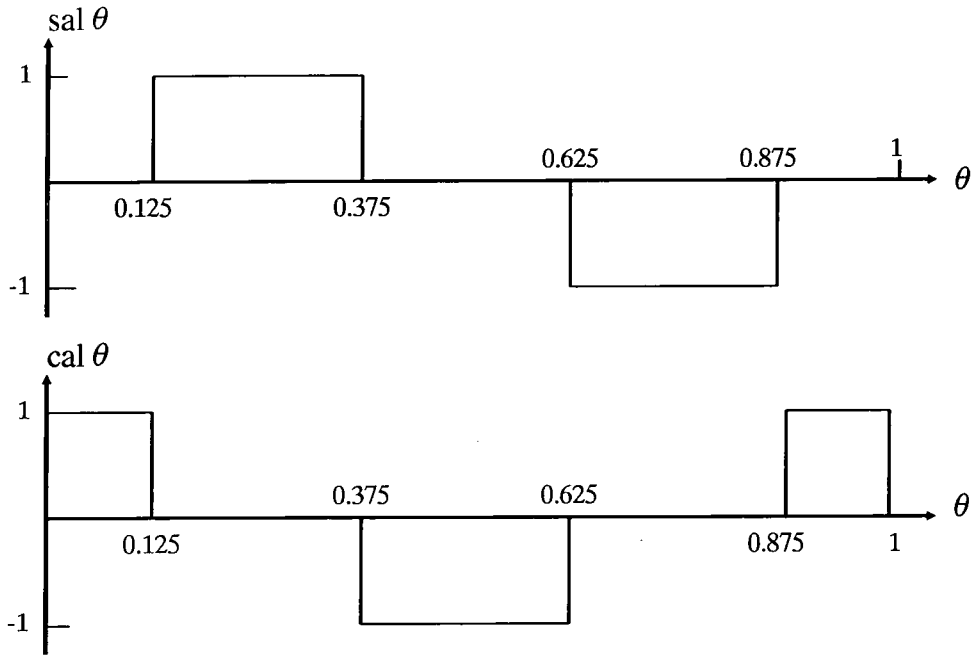


Fig. 7-4 The truncated sine function, sal θ, and cosine function, cal θ [46]. The truncation level is at $\sqrt{2}$ (=0.707) and the period of the function, 2π, is scaled to 1.

that leakage occurs and generates a background noise in the spectrum. The Fourier series of the cal and sal functions can be written as :

$$cal\,\theta = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos k\theta \qquad \text{and} \qquad sal\,\theta = \sum_{k=1}^{\infty} b_k \sin k\theta$$

where

$$a_0 = \frac{2}{\pi}\left(\int_0^{\pi/2-\theta_c} d\theta - \int_{\pi/2+\theta_c}^{\pi} d\theta\right) = 0$$

$$a_k = \frac{2}{\pi}\left(\int_0^{\pi/2-\theta_c} \cos k\theta d\theta - \int_{\pi/2+\theta_c}^{\pi} \cos k\theta d\theta\right)$$

$$= \frac{2}{k\pi}\left\{\sin\left[k\left(\frac{\pi}{2}-\theta_c\right)\right] + \sin\left[k\left(\frac{\pi}{2}+\theta_c\right)\right]\right\} \qquad \textbf{(7-4)}$$

and

$$b_k = \frac{2}{\pi}\int_{\theta_c}^{\pi-\theta_c} \sin k\theta d\theta = \frac{2}{k\pi}\left\{\cos k\theta_c - \cos\left[k\left(\pi-\theta_c\right)\right]\right\} \qquad \textbf{(7-5)}$$

Hence for the functions in Fig. 7-4 where $\theta_c = 0.25\pi$:

$$cal\,\theta = \begin{cases} \left(\frac{4}{k\pi}\sin\frac{k\pi}{4}\right)\cos k\theta & \text{for } odd\ k \\ 0 & \text{for } even\ k \end{cases} \tag{7-6}$$

and

$$sal\,\theta = \begin{cases} \left(\frac{4}{k\pi}\cos\frac{k\pi}{4}\right)\sin k\theta & \text{for } odd\ k \\ 0 & \text{for } even\ k \end{cases} \tag{7-7}$$

which are shown in Fig. 7-5. Since sal θ and cal θ correspond to sin θ and cos θ respectively, the coefficients other than k = 1 represent leakage terms. The desired coefficients at k = 1 contain about 44% of the total energy of the series. A different θ_c, of course, yields different series for the sal and cal functions, but the desired coefficients will still be the most significant terms. This property is illustrated in Hughes and Herron [47], where the distributions of the Fourier coefficients for four different θ_c are shown.

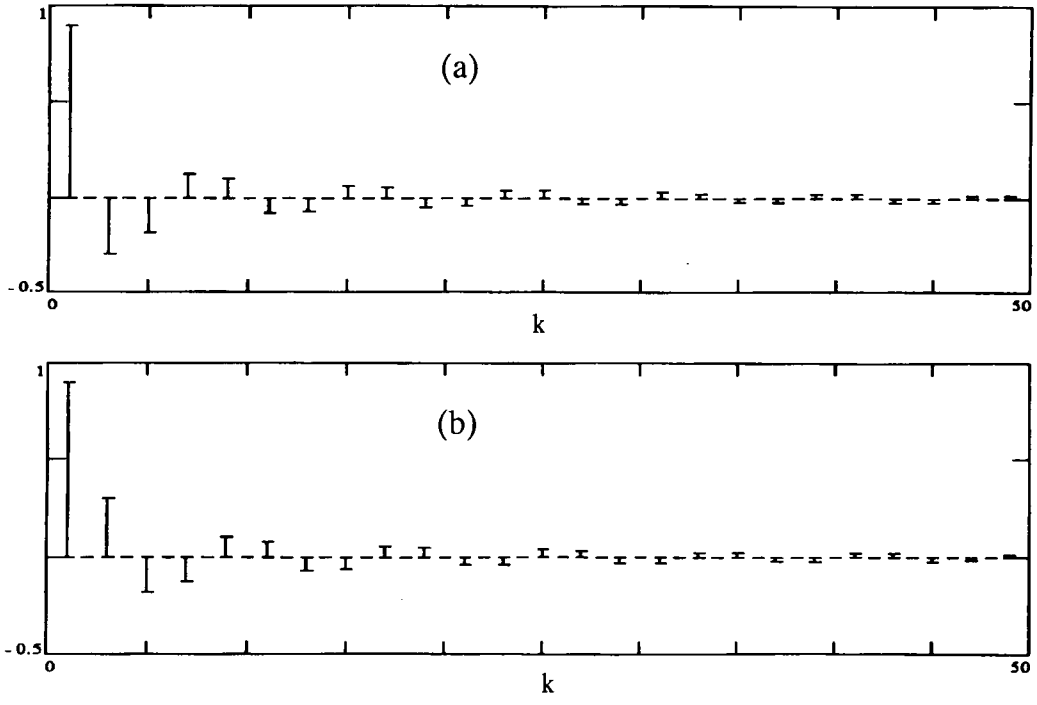


Fig. 7-5 The Fourier coefficients of (a) sal θ and (b) cal θ with $\theta_c = 0.25\pi$.

Equations (7-4) and (7-5) also indicate the amount of error in amplitude when the cal and sal functions are used as the kernel. By definition, k = 1 in the above equations corresponds to the desired frequency index and the desired values for a_1 and b_1 are 1. Hence the actual values of a_1 and b_1 are related to the amplitude of the components being recovered. For example, in Fig. 7-5 where $\theta_c = 0.25\pi$, $a_1 = b_1 = 0.9003$. Thus the real and imaginary parts of the computed components are expected to be 10% smaller in amplitude. Referring to Table 7-1, the relative accuracy of the 3-level kernel is 92%, which is about 8% smaller than expected. The set of amplitudes in (a) are used as reference for computing the error incurred in (b) because the input data x(n) is the same for column (a) and (b). More numerical examples will be given in Table 7-3 of section 7-4.

Numerically, θ_c can vary between 0 to 0.5π. When $\theta_c = 0$, we have a two-level kernel of values + 1 and -1, which gives an estimated error of 27.3 % (larger than the expected value) in the amplitude of a real component or imaginary component. As θ_c approaches 0.5π, the truncation level approaches 1, which means that the all input data will be discarded and the evaluation fails. Hence a practical value of θ_c falls within 0 to 0.25π.

7.2.2.3 Computational Complexity : Since no multiplications are required by this algorithm, the only computational load derives from additions (including subtractions). Fig. 7-6 shows a block diagram of the computation using the three-level kernel. Assuming that all input data x(n) in Fig. 7-6 are to be added, there are N-1 additions per X(k) and the overall complexity is thus N^2. The actual numbers required will be a fraction of N^2 since the "duty cycle" of the rectangular waves forming the kernel is always less than 100% (see Fig. 7-4).

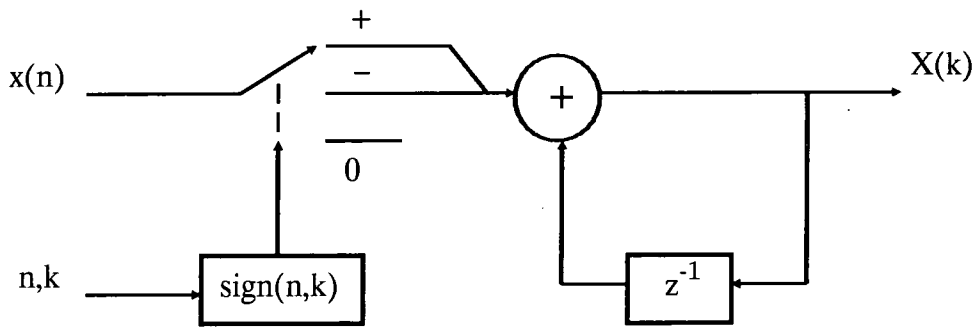


Fig.7-6 Block diagram of the evaluation of a frequency spectrum by a three-level truncation of the kernel values.

7.3 Estimation of Randomly Sampled Sequences

As pointed out in section 7.2.2, the approximate Fourier transform using the truncated sine and cosine functions requires no multiplications at all, which is very attractive for evaluating randomly sampled sequences. Fig. 7-7 shows two amplitude spectra of the same signal, $x(t) = 1.5\cos(2\pi.50t) + 2\sin(2\pi.506t) + \cos(2\pi.1{,}400t)$, sampled for 1,024 points and evaluated to k = 2,047. It can be seen that the spectrum recovered by the three-level kernel retains the anti-alias property of the additive random sampling (a.r.s.) although the amplitudes of the components are slightly less than those obtained by direct evaluation. Table 7-1 summarizes the results of the spectral estimation. Note that the ratios of the component amplitudes and the signal-to-noise ratios for both evaluation methods are virtually the same, which shows that the key dimensions of the spectrum are preserved. Taking the set of component amplitudes computed by the DFT as the reference, the accuracy of the three-level method is about 92%. The method also reduces the time required for computation by 57% because of the saving in multiplications.

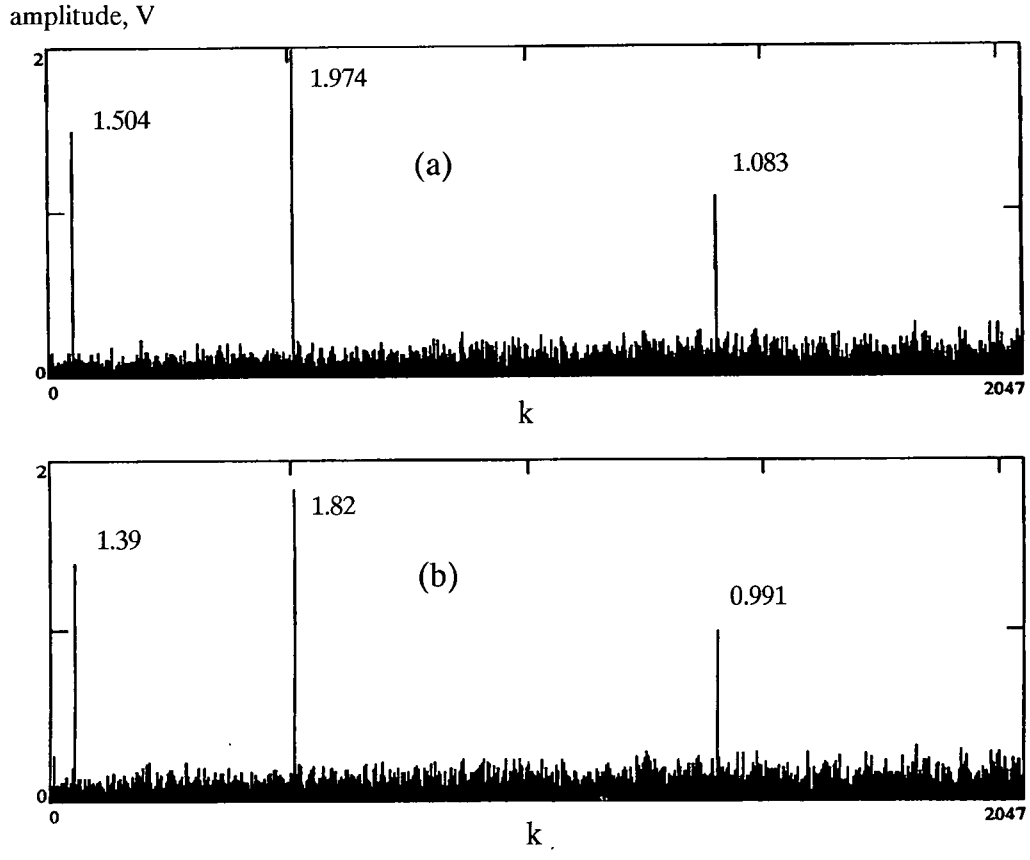


Fig. 7-7 Amplitude spectra of a signal x(t) = 1.5 cos(2π .50t) + 2 sin (2π .506t) + cos(2π .1,400t) sampled for 1,024 points by a.r.s. and evaluated by :

(a) direct computation, and

(b) the rapid method with a 3-level kernel shown in Fig. 7-4.

Table 7-1 : Data of the spectra shown in Fig. 7-5.

	Computed by :	
	(a) DFT	(b) 3-level kernel
Signal amplitude (V)	1.504, 1.974, 1.083	1.39, 1.82, 0.991
Amplitude ratio	1.39 : 1.82 : 1	1.40 : 1.84 : 1
Average accuracy (%)	96.7	94.3
Rel. accuracy (average)	1	0.92
S/N ratio (dB)	28.3	27.4
Peak noise (mV)	311	317
Computation time (sec)	78.55	33.61

7.4 Three-level Method with Parallel a.r.s. and Hybrid a.r.s

7.4.1 Accuracy and Saving : The algorithms for parallel a.r.s. and hybrid a.r.s., which are covered in Chapters 4 and 5 respectively, aim at reducing the number of multiplications required for evaluating the frequency components of a randomly sampled sequence. Both of these algorithms save at least 75% of the multiplications when compared to direct evaluation. It will be shown that data sequences sampled by parallel a.r.s. and hybrid a.r.s. can also be reconstructed with the three-level kernel. The most obvious advantage is that even the remaining 15% of multiplications can be saved as well.

The amplitude spectra of a signal $x(t) = 1.5 \cos(2\pi .50t) + 2 \sin(2\pi .506t) + \cos(2\pi .1{,}400t)$ sampled for 1,024 points by parallel a.r.s. and hybrid a.r.s. are shown in Fig. 7-8 (a) and (c) respectively. In (d) and (e), the sequences sampled by the above methods respectively are reconstructed using the three-level kernel with the truncation level at 0.707. It can be seen that the spectra reconstructed by using a three-level kernel maintain the anti-alias property. Table 7-2 lists the results of the computation. Since the principle of applying the three-level kernel to both of the above sampling methods is the same, hybrid a.r.s. will be chosen as the representative for further discussion.

To illustrate the effect of the cut-in angle on the relative accuracy, Table 7-3 lists a set of results when the above signal x(t) is sampled by hybrid a.r.s. and computed by three-level kernels of different θ_c. The first row (DFT) is the result of direct evaluation used as the set of references. The column of expected relative accuracy is in fact the magnitude of a_1 or b_1 given by eqn 7-4 or 7-5. It is obvious that the values of relative accuracy obtained by computation match closely with the expected values.

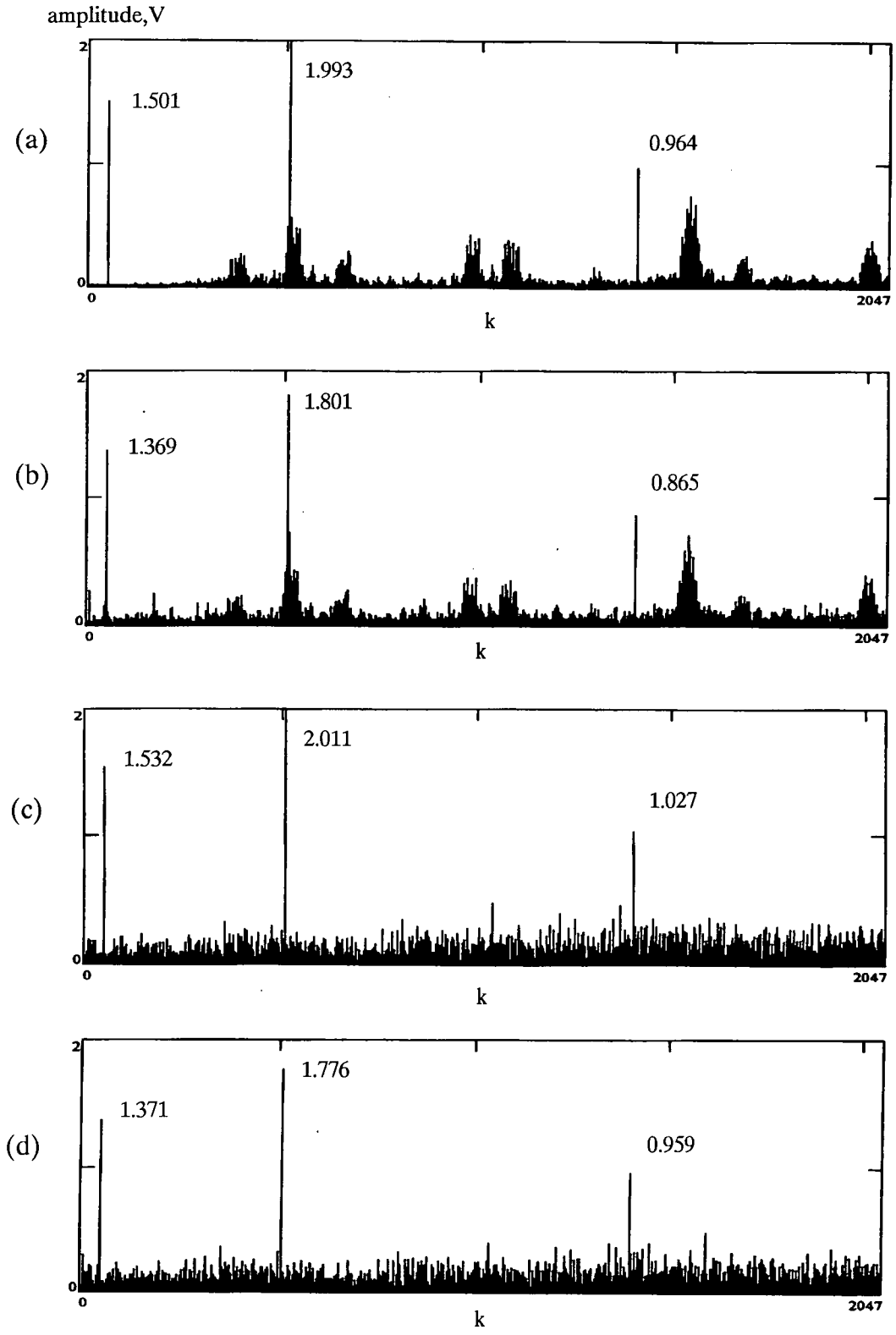


Fig. 7-8 Amplitude spectra of a signal x(t) = 1.5 cos(2π .50t) + 2 sin (2π .506t) + cos(2π .1,400t) sampled for 1024 points by :

(a) parallel a.r.s. and reconstructed by the DFT,

(b) parallel a.r.s. and reconstructed by a three-level kernel with $\theta_c = 0.25\pi$,

(c) hybrid a.r.s. and reconstructed by the DFT, and

(d) hybrid a.r.s. and reconstructed by a three-level kernel with $\theta_c = 0.25\pi$.

Table 7-2 : Data of the spectra from parallel a.r.s. and hybrid a.r.s. shown in Fig. 7-8.

	Parallel a.r.s.		Hybrid a.r.s.	
	(a) DFT	(b) 3-level kernel	(c) DFT	(d) 3-level kernel
Signal amplitude (V)	1.501,1.993, 0.964	1.369, 1.801, 0.865	1.532, 2.011, 1.027	1.371, 1.776, 0.959
Amplitude ratio	1.56 : 2.07 : 1	1.58 : 2.08 : 1	1.49 : 1.98 : 1	1.43 : 1.85 : 1
Rel. accuracy (average)	1	0.91	1	0.90
S/N ratio (dB)	28.2	27.1	28.4	27.4
Peak noise (mV)	754	734	478	484
Computation time (sec)	77.9	31.2	77.4	30.9

Since these expected values are readily available, they can be used as *scaling factors* to adjust the amplitudes computed by a three-level kernel closer to those by direct evaluation. For example, in the second row of Table 7-3 where $\theta_c = 0.25\pi$, the expected relative accuracy = 0.9. When the computed components are divided by 0.9, their values become 1.52, 1.96 and 1.06, which are close to the reference values in the first row and also to the exact values of 1.5, 2 and 1.

The three-level kernel is devised to save multiplications, and so are the algorithms of parallel a.r.s. and hybrid a.r.s. When this kernel is applied to either of the two algorithms, the effort in determining the sal and cal values will also be reduced. Referring to the signal flow diagram for computing the hybrid a.r.s. shown in Fig. 5-6, for a sequence length of 16, only 4 instead of 16 cal and sal values are needed per data point. This results in a 75% saving in the load of determining the cal and sal values, which is exactly the same percentage of saving in multiplications by the cosine and

Table 7-3 : Effect of the cut-in angle θ_C on the relative accuracy of the component amplitude. The input sequence is sampled by hybrid a.r.s.

	expected rel. accuracy	computed amplitude in V (relative accuracy)		
DFT	1	1.532 (1)	2.011 (1)	1.027 (1)
$\theta_C=0.25\pi, \lambda=\sqrt{2}$	0.90	1.371 (0.90)	1.766 (0.88)	0.959 (0.93)
$\theta_C=0.166\pi, \lambda=0.5$	1.10	1.689 (1.10)	2.214 (1.10)	1.123 (1.09)
$\theta_C=0.096\pi, \lambda=0.3$	1.22	1.853 (1.21)	2.451(1.22)	1.202(1.17)

sine values. In practice, the saving of cal and sal in terms of computation time is hardly observable because the time for determining such a value by a 486 system is so short that this amount becomes an insignificant factor. Referring to column (d) of Table 7-2, from a total of 30.9 seconds of computation time, about 0.83 second can be saved .

7.4.2 Bandwidth : In principle, the substitution of the sine and cosine by the sal and cal does not affect the anti-alias property of a random sampling algorithm so that the resulting Nyquist frequency is also infinite. However, similar to the case discussed in section 3.3.3, the bandwidth of the spectrum will be limited by the characteristics of the host system that implements the reconstruction.

The situation is clear when we refer to the block diagram of the evaluation shown in Fig. 7-6. Assuming that the evaluation is realized by hardware, the sal and cal are generated by a switching action depending on the input indices n and k. The upper bound in frequency is obviously determined by how fast the switching can be. When realization is by software, the switching rate is reflected by the resolution of the word-length representing the sampling times, which is equivalent to the limiting frequency of the system. The limiting frequency index k_m occurs when the product $k_m t_n$ becomes an integer for all n, where $\{t_n\}$ is the set of sampling times. If, for

example, t_n is in seconds expressed in fixed point decimal number which holds values to 6 digits past the decimal point, then $k_m = 10^6$ Hz and the fold-over frequency is at $\frac{1}{2} k_m = 5 \times 10^5$ Hz. Proofs are omitted here since the analysis parallels that in section 3.3.3.

7.5 Concluding Remarks

The rapid evaluation method using a three-level kernel requires no multiplications at all for estimating a spectrum. Given a randomly sampled sequence that has a computation complexity of N^2 in both multiplication and addition, such a method is certainly an attractive solution to speed up the computation. In Tables 7-1and 7-2, we can see that this method saves about 60% of the original computation time.

The speed of computation is gained at the expense of the accuracy in the amplitude of the resulting spectrum. From the major coefficients of the Fourier series of sal θ and cal θ, nevertheless, a scaling factor can be obtained to adjust the computed amplitude. Another drawback is the leakage which occurs in the form of a broadband noise, but with random sampling, background noise exists anyway. By observation, these two types of noise are random in nature and exhibit no reinforcement to each other. Tables 7-1 and 7-2 show that the signal-to-noise ratios are not degraded by the rapid evaluation method.

From the frequency-domain representation in Fig. 7-5, it can be seen that the major components of the sal and cal functions map to the frequency indices (or wave numbers) of the sine and cosine functions. Consequently, the anti-alias property of random sampling is unaffected by using the sal and cal as substitutes for the sine and

cosine respectively. The rapid evaluation method can also be applied to parallel a.r.s. and hybrid a.r.s. to eliminate all the multiplications required by these two algorithms. In fact, with either of these algorithms, the load in determining the sal and cal values can also be saved. This load, however, is insignificant when realized in a modern computer and the saving will not enhance the speed of computation significantly.

If speed is a crucial factor to consider, the rapid evaluation method is the best candidate in computing the spectrum from a randomly sampled sequence. This evaluation method can also be exploited to reduce the hardware cost of implementation. As neither trigonometric functions nor multiplications are involved, a micro-processor having no floating-point operation (e.g. Intel 8086), or even a binary adder, is also suitable for computing the spectrum.

CHAPTER 8

APPLICATION EXAMPLES OF RANDOM AND PSEUDO-RANDOM SAMPLING

8.1 Introduction

Random sampling, also known as randomized sampling, irregular sampling, or time dithering [48, 49], is one of the sub-Nyquist sampling methods that can recover without aliasing a spectrum which is not band-limited. As discussed in Chapter 3, the use of random sampling may facilitate the adoption of slower hardware and save memory storage in some cases, but in general the loading in computation is heavier than regular sampling. Owing to this reason, specially designed hardware can be more efficient than a general purpose computer for computing a spectrum from a randomly sampled sequence. Besides the loading in computation, there are other costs associated with random sampling as well. Hence this sampling method finds its applications where the alias-free property or the reduction in the number of samples outweighs other considerations.

Random or irregular sampling applies naturally when signals being observed occur irregularly in time, such as those in astrophysics and space science. Another area of application is in instrumentation where anti-alias filtering is not desired or a speed higher than the normal operational speed of the available hardware is required. Typical applications in instrumentation have been introduced in Chapter 3. Two applications in digital signal processing, which are motion detection in images and the correlation detector, will be suggested in this chapter.

8.2 Motion Detection in Images

8.2.1 Segmentation by motion : For our visual perception, motion is a powerful cue to extract objects of interest from a background. In imaging applications, motion arises from a relative displacement between the sensing system and the object being viewed. The use of motion in segmentation can be achieved in both spatial and frequency domain [50,51]. The basic approach of the spatial domain technique is to compute the difference between the images taken at different instants whilst the frequency domain technique uses the Fourier transform to detect objects moving at a constant speed. In the following discussion, we shall focus on the latter approach equipped with random or pseudo-random sampling.

8.2.2 Frequency Domain Technique : Assuming an object moves at a constant speed, a sequence of T digital images of the scene of size MxN pixels per frame may be obtained as shown in Fig. 8-1. The projections of the object onto the x-axis and y-axis are given by $\sum_{y=0}^{N-1} f(x,y,t)$ and $\sum_{x=0}^{M-1} f(x,y,t)$ respectively, where $t = 0,1,...,T-1$. A complex

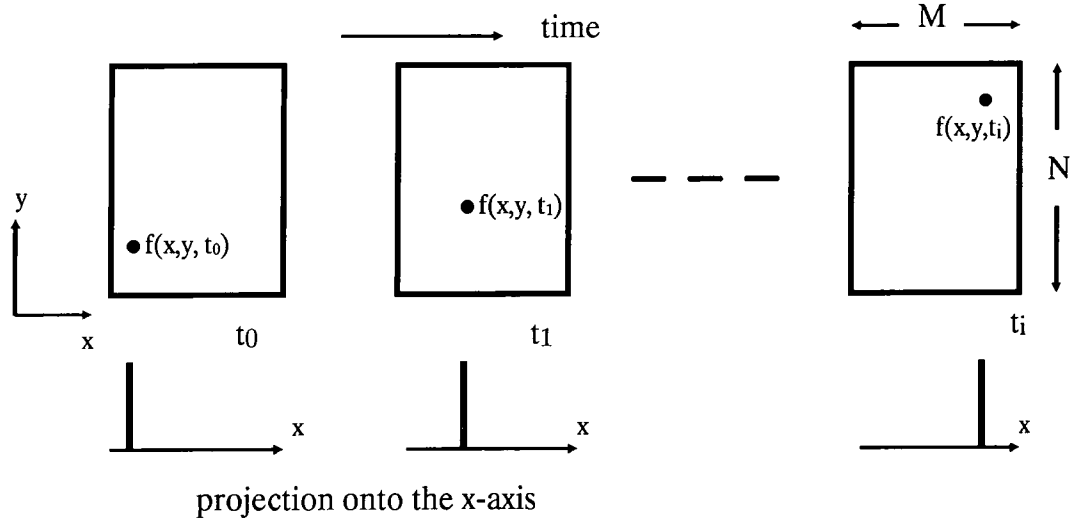


Fig. 8-1 A sequence of image frames.

sinusoid of frequency k_1 can be multiplied to the sums of pixel values added along the y direction. The sum of the weighted projections onto the x-axis at time t is [52]:

$$g_x(t,k_1) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y,t)\, e^{j2\pi k_1 x \Delta t} \qquad \textbf{(8-1)}$$

where t = 0,1,...,T-1, and k_1 is a positive integer.

Similarly, the sum of the weighted projections onto the y- axis is :

$$g_y(t,k_2) = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} f(x,y,t)\, e^{j2\pi k_2 y \Delta t} \qquad \textbf{(8-2)}$$

where t = 0,1,...,T-1, and k_2 is a positive integer.

Let v_1 and v_2 be the velocities of the motion in the x and y directions respectively. The 1-D Fourier transforms of eqns (8-1) and (8-2) respectively become:

$$G_x(u_1,k_1) = \frac{1}{T} \sum_{t=0}^{T-1} g_x(t,k_1)\, e^{-j2\pi u_1 t/T} \qquad \textbf{(8-3)}$$

where $u_1 = 0,1,...,T-1$ and

$$G_y(u_2,k_2) = \frac{1}{T} \sum_{t=0}^{T-1} g_y(t,k_2)\, e^{-j2\pi u_2 t/T} \qquad \textbf{(8-4)}$$

where $u_2 = 0,1,...,T-1$. Obviously $u_1 = k_1 v_1$ and $u_2 = k_2 v_2$. Here v_1 and v_2 are in pixels per total frame time and the actual physical speeds depend on the frame rate.

Let us consider a particular case that a point object is moving in the x-direction in a background with a high but constant level. Hence eqn(8-1) applies. Note that the multiplication of the complex sinusoid, $e^{j2\pi k_1 x \Delta t}$, is orthogonal to the projection of the image. The result is that any moving point will be characterized by a complex sinusoid and any static level, no matter how large, will be averaged to zero or nearly so.

Referring to eqn (8-1), let $S(x,t) = \sum_{y=0}^{N-1} f(x,y,t)$. The plot of S versus x at two particular times t_1 and t_2 is shown in Fig. 8-2 (b), where A is the background level and $A >> (B - A)$, the projection of the object above the background. Then eqn (8-1) can be written as :

$$g_x(t,k_1) = \sum_{x=0}^{M-1} S(x,t)\, e^{j2\pi k_1 x \Delta t}$$

$$= \sum_{x=0}^{M-1} \left[B(x,t) - A\right] e^{j2\pi k_1 x \Delta t} + \sum_{x=0}^{M-1} A\, e^{j2\pi k_1 x \Delta t} \qquad \textbf{(8-1a)}$$

For the first term in the above equation, at each Δt there exists only 1 value of B(x,t) at a particular x. Hence the summation gives a complex sinusoid :

$$\sum_{x=0}^{M-1} \left[B(x,t) - A\right] e^{j2\pi k_1 x \Delta t} = C\, e^{j2\pi k_1 x \Delta t}, \qquad C \text{ a constant}$$

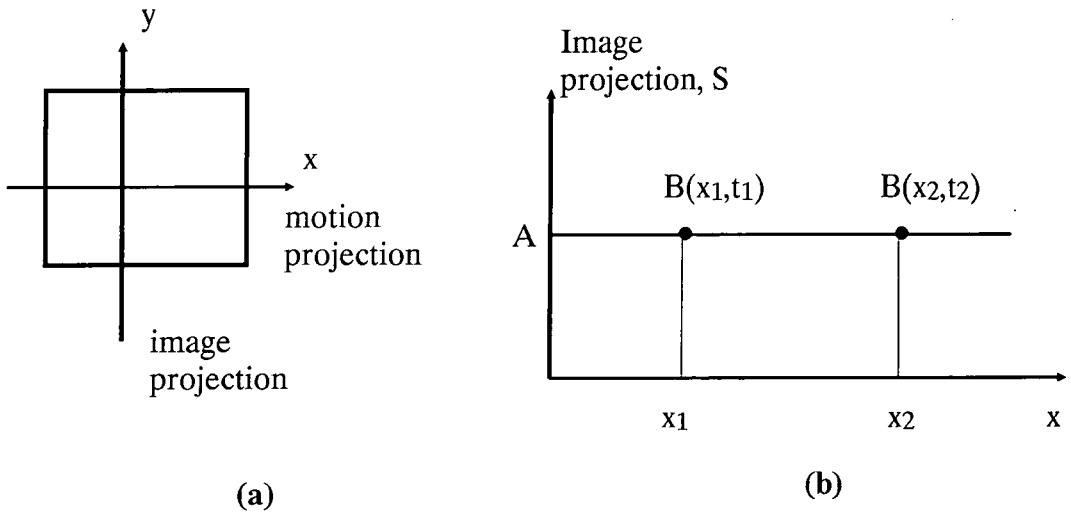


Fig. 8-2 (a) The orthogonality of the projections. (b) The image projections of a high background level A and a moving point B at different times t_1 and t_2..

To have a proper DFT with no smearing, $k_1\Delta t$ must be chosen such that $e^{j2\pi k_1 x \Delta t}$ traverses a number of complete cycles in the M frames. Hence the second term of eqn (8-1a) can be written as :

$$A \sum_{x=0}^{M-1} e^{j2\pi\, k_1 x \Delta t} = 0 .$$

Therefore $g_x(t,k_1) = C\, e^{j2\pi k_1 x \Delta t}$ and its DFT should look similar to those in Fig. 8-4. From the above example, we can generalize that a moving object is characterized by a complex sinusoid, hence contributes to a frequency component in the DFT. Any static background, however, will be suppressed to zero or a very small value in the spectrum.

A practical example is taken from p.474 to 477 of [52]. Fig. 8-3 (a) shows one of a 32-frame sequence of LANDSAT images with white noise added to it. There is an object moving at 0.5 pixel per frame in the x direction and 1 pixel per frame in the y direction. The target, which is circled in Fig. 8-3 (b), has a Gaussian intensity distribution spread over a small area and is hardly discernible. The spectra G_x and G_y, computed according to eqns (8-3) and (8-4), are shown in Fig. 8-4. The indices k_1 and k_2 are chosen to be 6 and 4 respectively. Taking G_x as example, there should be two peaks, one at frequency index 3 (= $k_1 v_1$) and the other at 29 (= T - $k_1 v_1$). From the location of the first peak and the frame rate, we can deduce the speed of the object. This method is especially effective for checking an object moving slowly in a stable background scene corrupted by white noise, e.g. satellite images. Because of aliasing, there is limitation in choosing the values of k_1 and k_2. Suppose k_1 is chosen to be 34 instead of 6 , there will be two peaks at frequency indices 15 and 17 in the spectrum. If 15 is taken as the solution, the result will be incorrect.

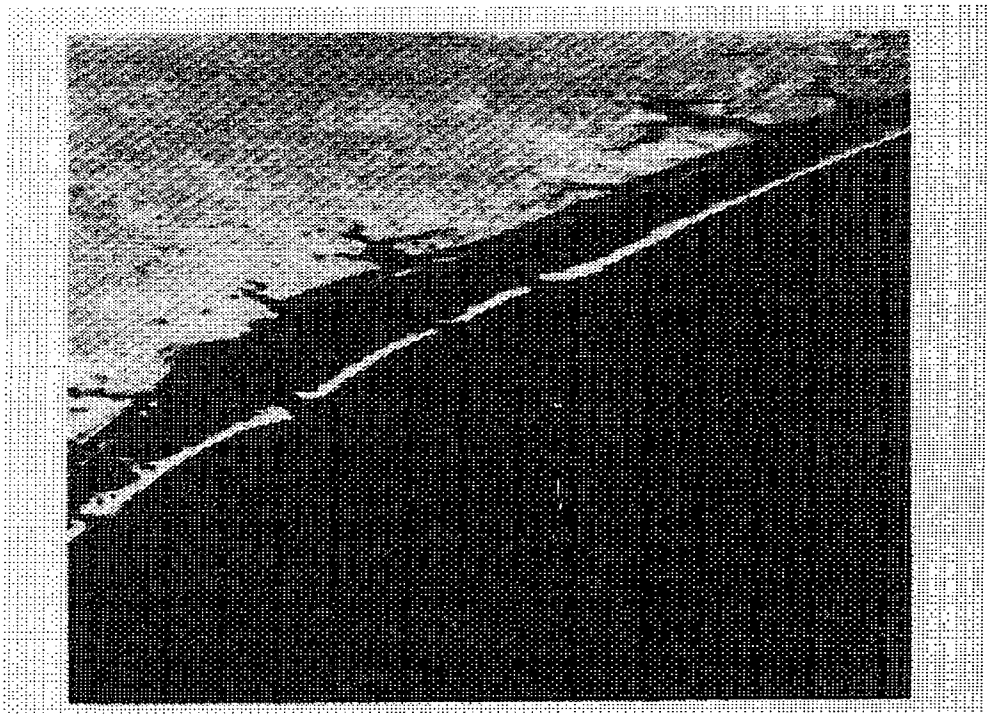


Fig. 8-3(a) A LANSAT frame (From Cowart, Snyder and Ruedger [53])

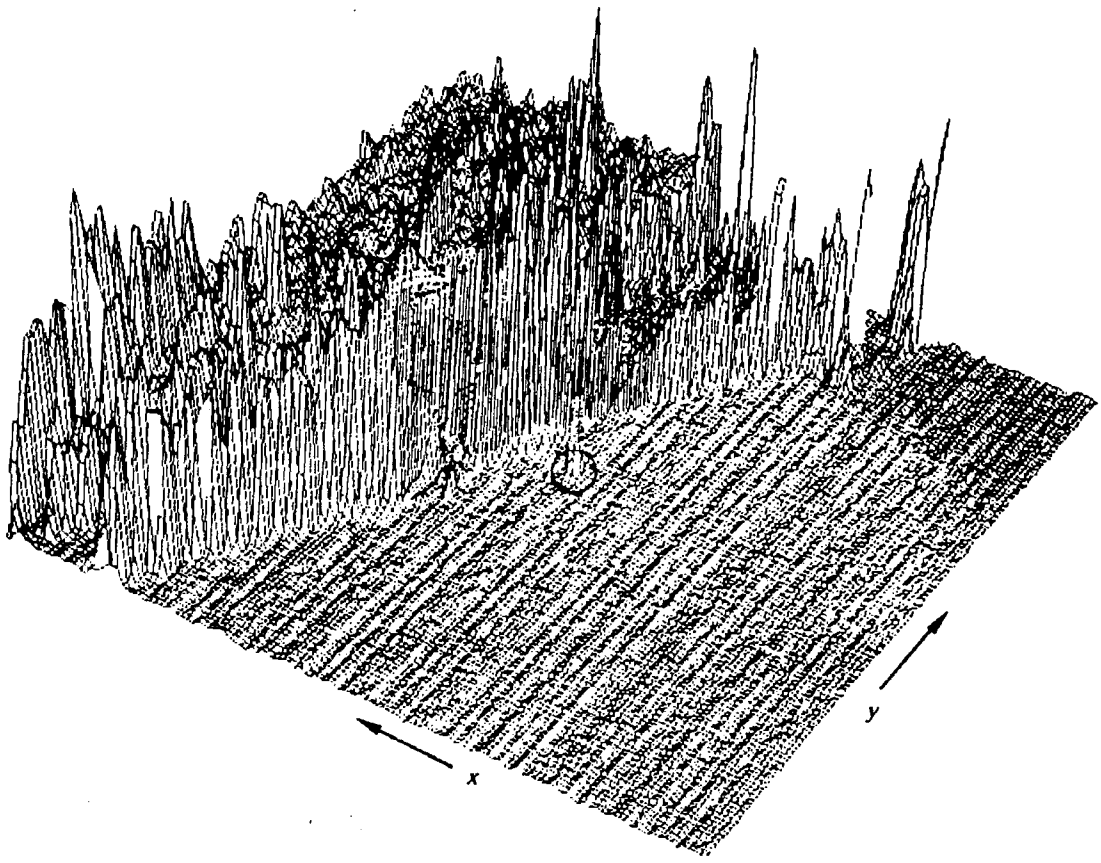


Fig. 8-3 (b) Intensity plot of the above frame with target circled. (From Rajala, Riddle, and Snyder [51])

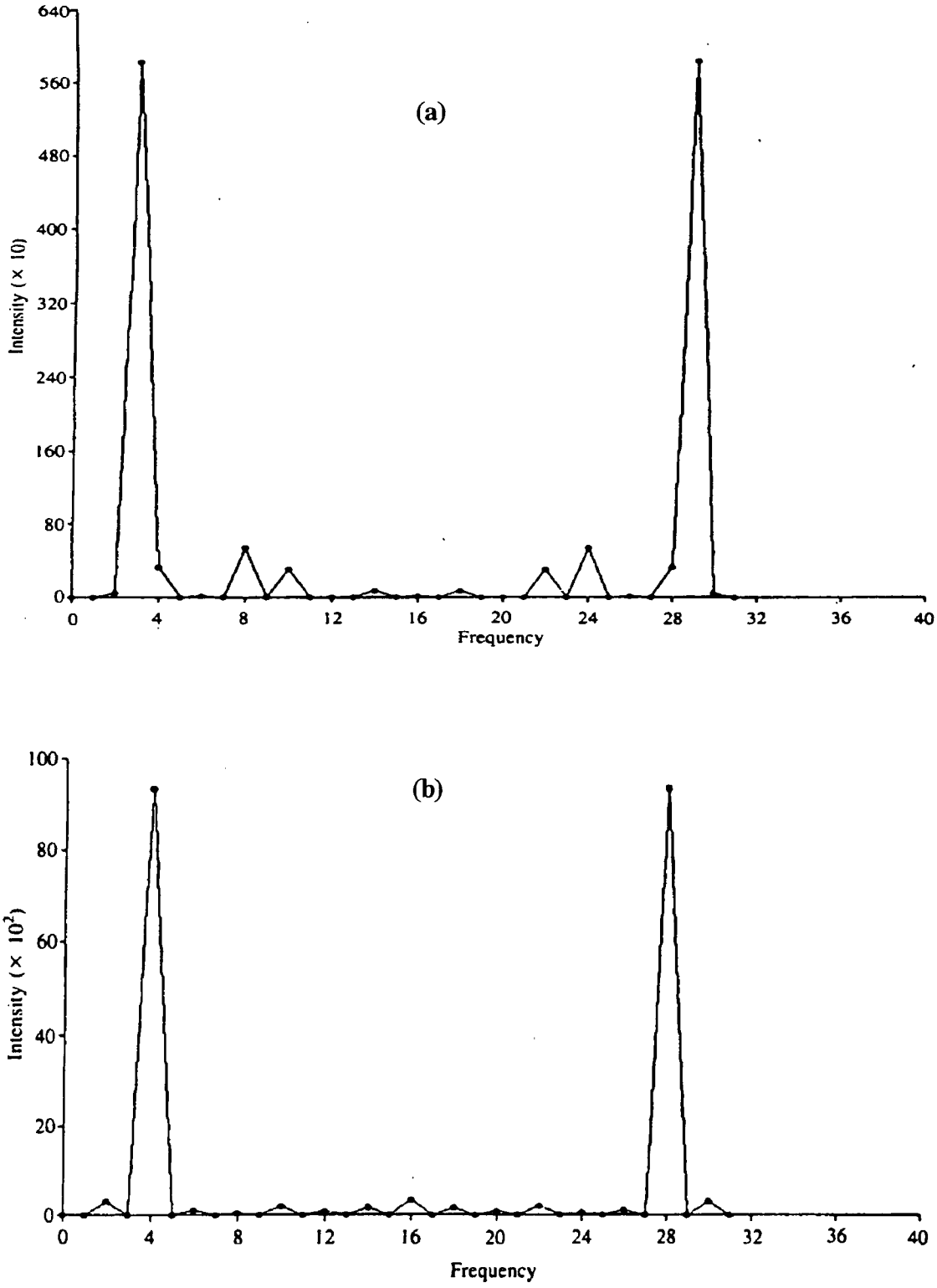


Fig. 8-4 Spectra of a moving object in an image :
(a) G_x with 2 peaks at frequency indices 3 and 29;
(b) G_y with 2 peaks at frequency indices 4 and 28.
(From Rajala, Riddle and Snyder [51])

8.2.3 Applying Random Sampling : Referring to the frequency domain technique discussed above, if the timing of the input frame is a continuous variable, i.e. an image frame can be taken at any moment, random sampling can be applied so as to gain the advantage of being alias-free. One of the consequences is that the range of the multipliers k_1 and k_2 described above can be extended [54].

Let us return to the previous numerical example that k_1 is chosen to be 34. Assume that 32 frames are taken in a duration of one unit of time, the timing for each frame is given by $t_i = i/32 + \tau_i$, where τ is a uniformly distributed random variable. Fig. 8-5 (a) shows the power spectrum of a simulated sequence of length 32 with random noise added. A peak is seen only at frequency index 17, hence no ambiguity arises in choosing the solution. Two more examples are shown in Fig 8-5 (b) and (c) where a sequence is sampled by two random sampling methods, namely jittered random sampling and additive random sampling respectively for 64 points. Note that the input frequency is 41 and the Nyquist limit is 32, but no alias occurs in both cases.

8.2.4 Pseudo-random Sampling : In many practical cases, frames of images are recorded uniformly in time, hence genuine random sampling may not be applicable. We can, however, under-sample the input sequence of images by adopting a scheme which selects the samples $f(x,y,t_n)$ from T frames recorded regularly at an interval of Δt between frames at :

$$t_n = (nP + \sigma_n)\,\Delta t \qquad \textbf{(8-5)}$$

where n = 0,1,2,...,Q-1, Q is an integer which divides T, P = T/Q and σ is a random variable with integer values distributing uniformly between $\pm P/2$.

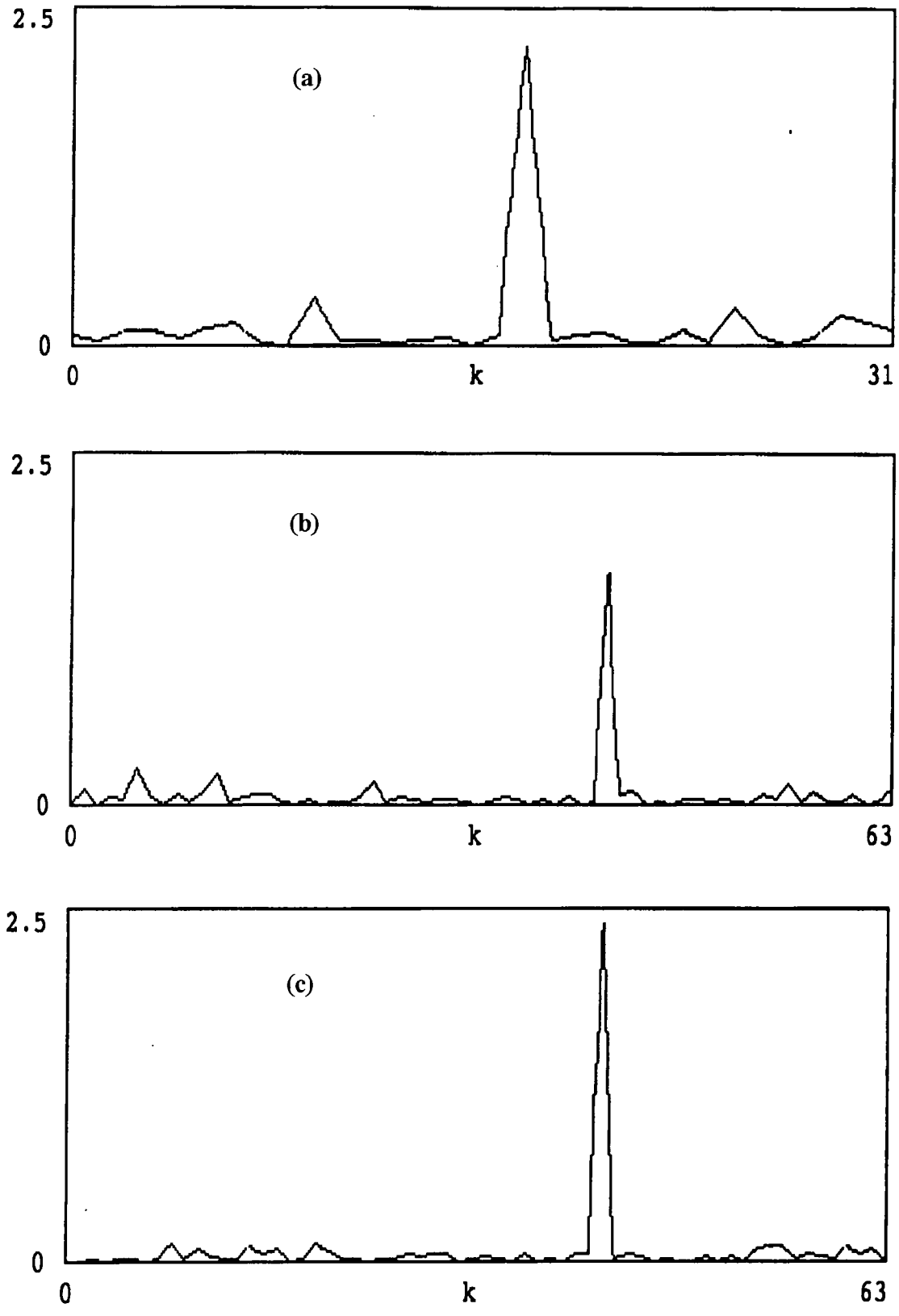


Fig. 8-5 Power spectra of a sequence (with random noise added) sampled by :
(a) jittered random sampling for 32 points. The input frequency (peak) is at k = 17.
(b) jittered random sampling for 64 points. The input frequency (peak) is at k = 41.
(c) additive random sampling for 64 points. The input frequency (peak) is at k =41.

8.2.4.1 Advantages : The spectrum of the above sequence selected by this pseudo-random sampling scheme differs from that of a sequence of samples obtained by down-sampling the input sequence regularly with zeros padded in between. The former method retains the original fold-over frequency whilst the latter does not. For example, if the total number of input frames available (T) is 128 and the number of frames selected (Q) is 32, i.e. a down-sampling of 4, for the pseudo-random sampling method , the fold-over frequency is at 64 (= T/2), whereas for the regular case, the fold-over frequency is effectively at 16 (= Q/2) only. By regularly skipping the data, only a repetition of a compressed version of the original spectrum is obtained (see Fig. 8-6(a)). Fig 8-6 (b) and (c) show the spectra of two sequences with 32 points selected by the pseudo-random scheme from 128 and 256 points respectively. Clearly the frequencies at k = 60 and k = 120 of the two input sequences are in place. Hence with the pseudo- random scheme, the ranges for the multipliers k_1 and k_2 are extended comparing to the regular down-sampling.

In order to detect a slow movement of a target, a long period of observation, hence a large number of frames may be required for the analysis. In doing so, more memory space is required. One solution is that frames are skipped regularly, which effectively lowers the sampling rate and may generate alias. If the pseudo-random sampling is adopted, not only the fold-over frequency remains as if no skipping is done, but also the memory space required is less than storing the original input sequence. Suppose a sequence of T words is obtained and down-sampled by the pseudo-random sampling method by a factor of 4, at most T/2 words, which includes T/4 words of the sampled data and T/4 words denoting the corresponding timing, will be stored. If the timing sequence is not stored, the data sequence should contain

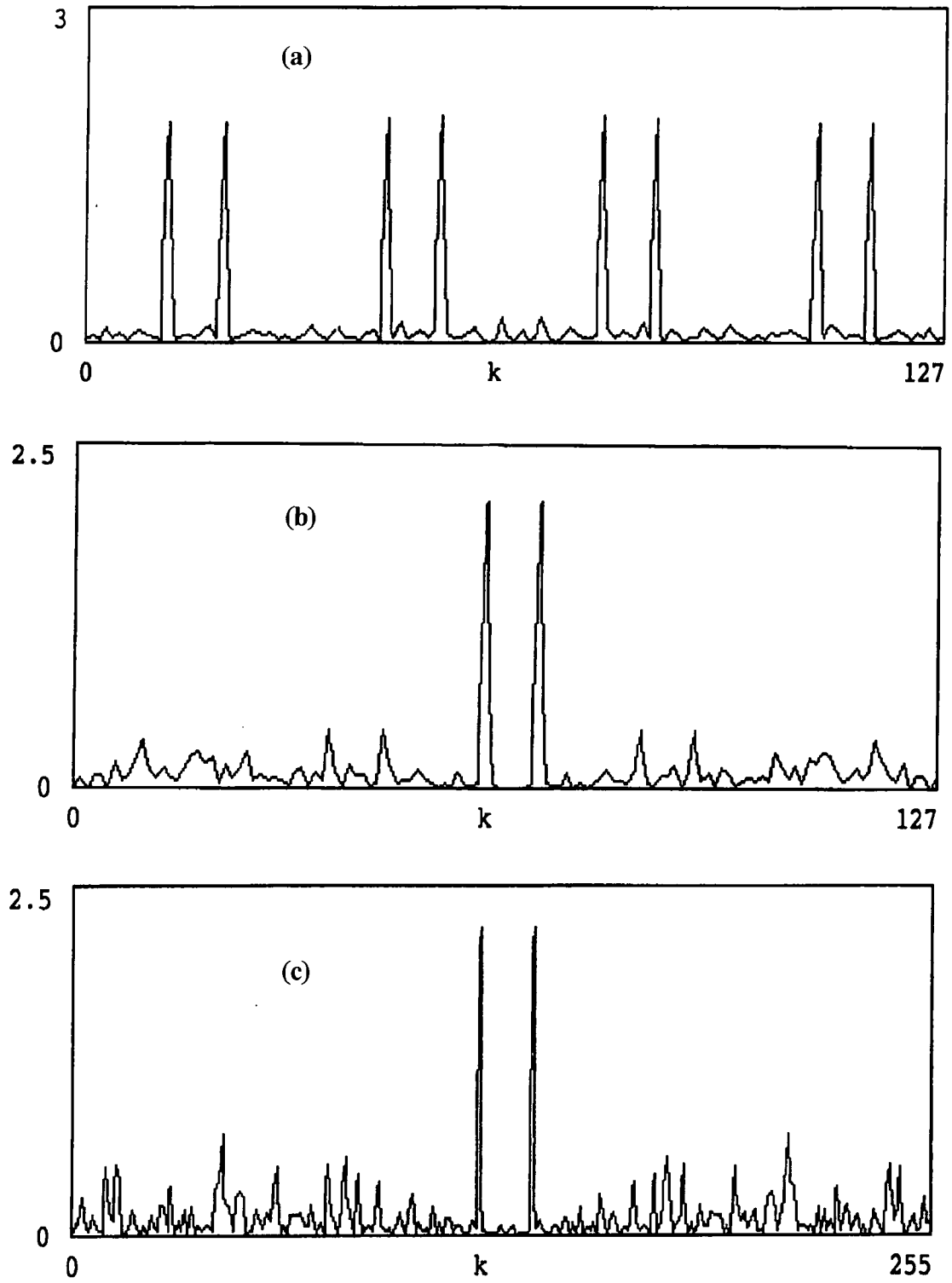


Fig. 8-6 Power spectra of a sequence (with random noise added)

(a) down-sampled regularly by a factor of 4 with zeros padded. Original length is 128 points.

(b) down-sampled with pseudo-random sampling by a factor of 4. Original length is 128 points and input frequency is at k = 60.

(c) down-sampled with pseudo-random sampling by a factor of 8. Original length is 256 points and input frequency is at k = 120.

zeros at all instants where samples are discarded. Let u(x,y,t_i) be the values of the sampled sequence, then

$$\begin{cases} u(x, y, t_i) = f(x, y, t_i), & \text{if } t_i \in t_n \\ u(x, y, t_i) = 0, & \text{if } t_i \notin t_n \end{cases} \quad \textbf{(8-6)}$$

Since these zeros appear consecutively in the form of short sequences, encoding methods, e.g. run-length encoding, may be used to reduce the amount of memory storage.

8.2.5 Simulation results : When random or pseudo-random sampling is adopted to sample a signal, the amplitude of the signal recovered cannot be exact since randomness is introduced in the timing. Table 8-1 summarizes the percentage errors in the signal amplitude from the spectrum estimated by the above methods described. We can see that all errors are below 10 %. Since different sets of random variables are used, we cannot judge their relative performance by directly comparing the numerical values of these errors.

Random sampling and pseudo-random sampling methods can be applied in detecting moving objects recorded by a sequence of image frames, e.g. satellite images. As random sampling is alias free, there is no ambiguity in choosing the correct peak of frequency in the spectrum for calculating the speed of the object. Pseudo-random sampling effectively retains the usable frequency range of a long sequence although data points are in fact discarded. This scheme is especially suitable for detecting a slow moving object as discussed in section 8.2.4.1.

With either random or pseudo-random sampling, background noise is generated and error is introduced in the amplitude spectrum. From Fig. 8-5 and Fig.

8-6, however, we can see that the peak frequencies are strong enough to be extracted even with random noise deliberately added to the input sequences. Table 8-1 shows that the signal amplitude estimated by the above methods are at least 90% accurate. For the pseudo-random method, if zeros are inserted in the sequence where samples are discarded, the evaluation of the spectrum can be performed with the FFT provided that the original sequence length is a power of two.

Table 8-1 : Percentage error of the signal amplitude recovered from different random sampling methods

Sampling Method	% error
jitter, 32 points, Fig. 8-5 (a)	4.95
jitter, 64 points, Fig. 8-5 (b)	9.47
additive random sampling, 64 points , Fig. 8-5 (c)	9.19
pseudo-random, 128 points*, Fig. 8-6 (b)	1.98
pseudo-random, 256 points*, Fig. 8-6 (c)	5.30

*32 points of the original sequences are selected

8.3 Correlation Detector

The cross-correlation of two sequences x(n) and s(n) can be defined as :

$$R_{sx}(l) = \sum_{l=0}^{N-1} x(n)\, s(n-l) \qquad \textbf{(8-7)}$$

where n = 0,1,2,. . . , N-1 and N is the sequence length. This function measures how similar these two sequences are and it can be used for signal comparison such as in the detection of a signal in white noise or a one-dimensional template matching in image processing. The block diagrams for both of the mentioned applications would be similar to the one shown in Fig. 8-7, where x(n) is the input signal to be compared with a replica signal s(n). Although the operation is performed in the time domain, with regular sampling, the problem of aliasing still exists if the input is sinusoidal or periodic. Suppose s(n) = A cos (2π f n/N), then the system will not be able to distinguish x(n) = A cos (2π f n/N) from x(n) = A cos [2π f(N + n)/N]. Another example is illustrated in Fig. 8-8, where two patterns of grating are sampled regularly Both of them will yield a sequence {1,1,1,0,0,1,1,1,0, . . .}. If either of them is taken as the template, both patterns will be announced the same. One approach is to increase the sampling rate, which will elongate the sequences and increase the number of

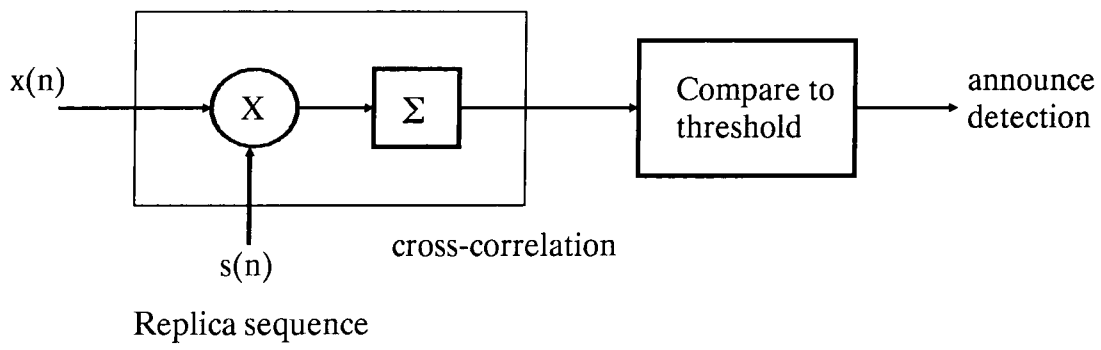


Fig. 8-7 Detection of a deterministic signal in white noise [42].

operations. Another approach is to adopt random or irregular sampling. Suppose the sampling instants 1,2 and 5 are shifted slightly to the left, the sequence from the first pattern may become {1,0,0,0,0,0,1,1,0, . . . } while the sequence from the second pattern will remain unchanged. The two patterns can now be differentiated.

8.3.1 Cross-correlation of Randomly Sampled Sequences : Let us define, according to the format in section 6.4, the *linear* cross-correlation of two signals as :

$$R_{sx}(l) = \sum_{l=0}^{N-1} x(t_n)\, s(t_n - l.t_s) \qquad \textbf{(8-8)}$$

where n = 0,1,2, . . . and t_s is step size of the correlation. This equation can be used to demonstrate the anti-alias property and noise immunity of random sampling.

Assume that, in Fig. 8-7, the replica signal is s(t) = 1.5 cos (2π.50 t) and it is sampled by additive random sampling for 512 points to yield a replica sequence $s(t_n) = 1.5 \cos(2\pi.50\, t_n)$. The input signal x(t) will also be sampled by the same timing

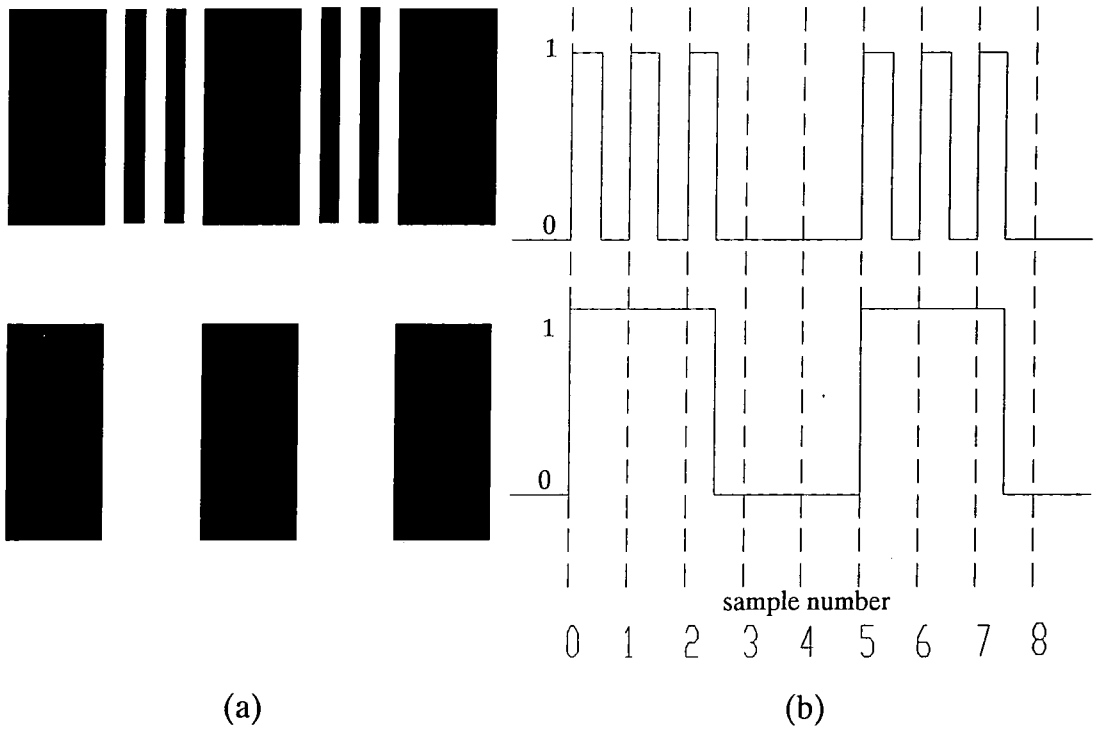


Fig. 8-8 (a) Two patterns of grating and (b) their corresponding intensity profiles obtained by regular sampling such that the resulting sequences are the same.

sequence to give $x(t_n)$. Fig. 8-9 shows the results of cross-correlations of $s(t_n)$ with different $x(t_n)$ using a step size of 1/512 and a window size of $\pm 10\%$ of the step size (section 6.3.1). In Fig. 8-9(a), $x(t_n) = s(t_n)$, and the matching is reflected by a large value in $R_{sx}(0) = 580.7$. In Fig. 8-9(b), $x(t_n) = s(t_n) + e(t_n)$, where $e(t_n)$ is a random noise of 0.58 V r.m.s., giving a signal-to-noise ratio of 8.25 dB. In spite of such strong noise, $R_{sx}(0) = 587.5$. In Fig. 8-9(c), $x(t_n) = 1.5 \cos (2\pi.562\ t_n)$, which would be an alias of $s(t_n)$ if sampling were performed regularly. With random sampling, however, aliasing is suppressed and $R_{sx}(0)$ is only -29.2.

The correlation detector works not only with monotonic sinusoids, but also with signals having several frequency components. Table 8-2 shows another set of results of a template $s(t_n) = 1.5 \cos (2\pi.50\ t_n) + 2 \sin (2\pi.506\ t_n) + \cos (2\pi.1400\ t_n)$ correlating with different inputs. Only the first terms of the correlation, i.e. $R_{sx}(0)$, are listed because these are the most significant terms. The first two rows of Table 8-2 show the results of a perfect match and a perfect match corrupted by noise. It can be seen that $R_{sx}(0)$ are large. In row three to row six, matches are partial so that the values of $R_{sx}(0)$ are smaller than the previous cases. In row seven, the input has components which are aliases of the template $s(t_n)$; and in the last row, the input frequencies are close, but not equal, to those of the template. For these cases of mis-matches, the results are small in value and negative in sign. Note that in this example the Nyquist frequency is 256, but an input frequency more than 5 times this value (i.e. 1,400) can also be handled.

8.3.2 Cross-correlation with Delayed Signal : In regular sampling, an input being a delayed version of a template can be detected by cross-correlation using eqn (8-7). Suppose the input $x(n) = s(n-m)$, then the maximum value $R_{sx}(m)$ in the resulting

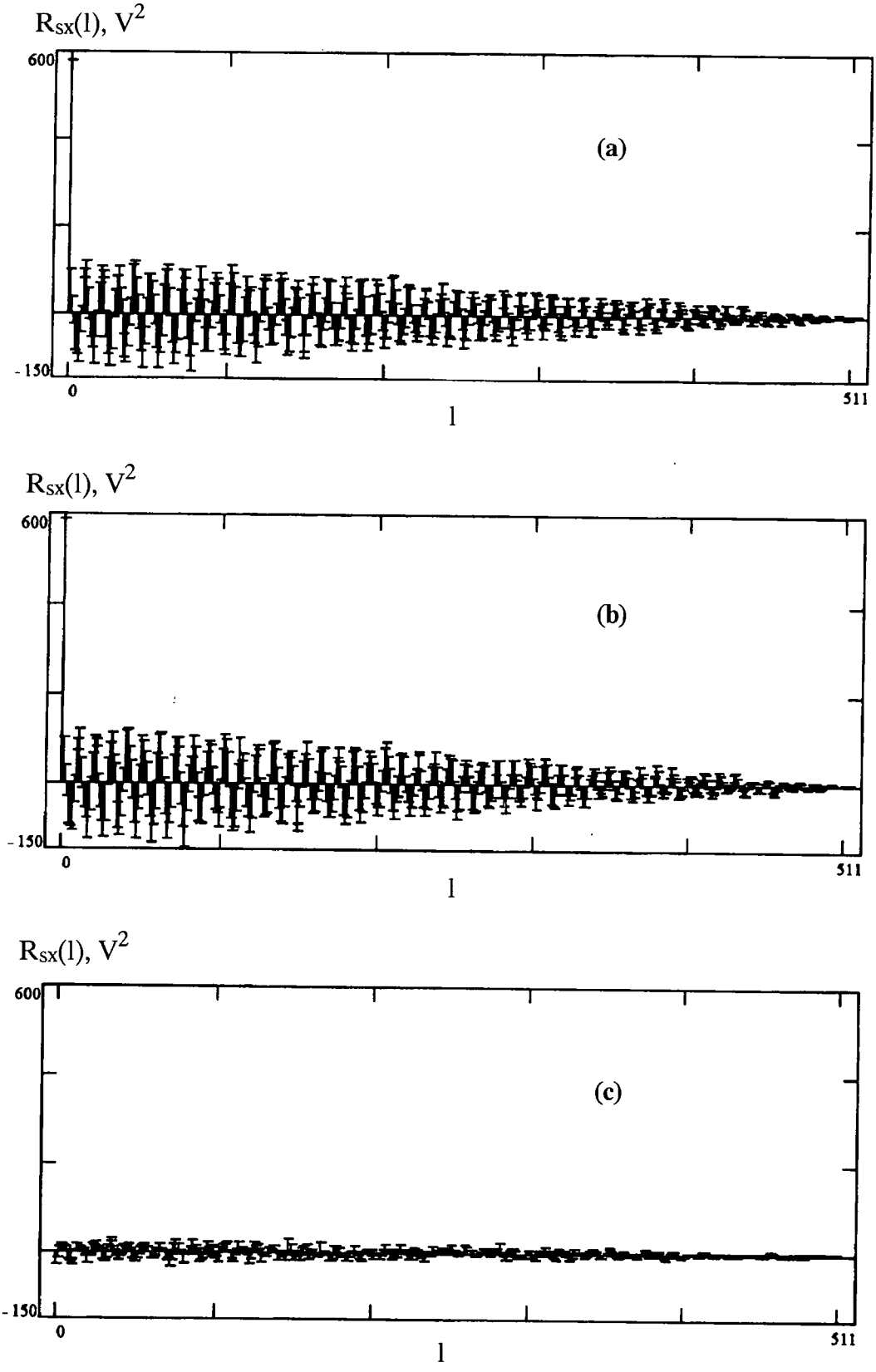


Fig. 8-9 Cross-correlation of a template $s(t_n) = 1.5 \cos(2\pi . 50\, t_n)$ with :

(a) $x(t_n) = s(t_n)$. A large $R_{sx}(0)$ indicates a match at $l = 0$.

(b) $x(t_n) = s(t_n)$ + noise. $R_{sx}(0)$ is also large.

(c) $x(t_n) = 1.5 \cos(2\pi . 562\, t_n)$. This alias is rejected since $R_{sx}(0)$ is small.

sequence will indicate a match at l = m. With random sampling, however, a linear shift in the time or spatial domain cannot align two randomly sampled sequences. A different approach, which uses a delayed version of the template for detecting a delayed signal, is suggested :

$$R_{sx}(l) = \sum_{n=0}^{N-1} x(t_n)\, s_l(t_n) \qquad \textbf{(8-9)}$$

where $s_l(t_n) = s(t_n - l.d)$, l is an integer and d is a fixed delay interval. Obviously d is the resolution of the cross-correlation. Fig. 8-10 shows three cases of correlation using eqn(8-9). The input sequences in (a) and (b) are sampled regularly. In (b) the input signal is delayed by 14 sampling periods with reference to (a) and in (c) the input signal is a randomized version of (b). From these examples, we can see that the maximum values in $R_{sx}(l)$, which are $R_{sx}(0)$ in (a) and $R_{sx}(14)$ in (b) and (c), are not much larger than some other correlation values, making the subsequent thresholding very unreliable, especially when the input sequence is randomized. Despite a strong correlation, an input sequence may be close, but not necessarily identical, to a particular template. For example, in Fig. 8-10 (c), $R_{sx}(14)$ = 3075, $R_{sx}(18)$ = 2883 and $R_{sx}(73)$ = 2967. The corresponding templates are $s_{14}(t_n)$ = {-6.35, 2.829, -0.385,

Table 8-2 : Cross-correlation of $s(t_n) = 1.5 \cos(2\pi.50\, t_n) + 2 \sin(2\pi.506\, t_n) + \cos(2\pi.1400\, t_n)$ with different inputs $x(t_n)$.

$x(t_n)$	$R_{sx}(0)$	matching
same as $s(t_n)$	1863.9	yes
same as $s(t_n)$ + noise	1906.5	yes
2 sin ($2\pi.506\, t_n$) + cos ($2\pi.1400\, t_n$) + noise	1332.4	partial
2 sin ($2\pi.506\, t_n$) + noise	1060.2	partial
1.5 cos ($2\pi.50\, t_n$) + noise	616.6	partial
cos ($2\pi.1400\, t_n$) + noise	314.8	partial
2 cos ($2\pi.562 t_n$) - 2 sin ($2\pi.6\, t_n$) + 3cos ($2\pi.376\, t_n$)	-135.4	no
2 cos ($2\pi.51 t_n$) + 2 sin ($2\pi.505\, t_n$) + 3sin ($2\pi.1400\, t_n$)	-230.5	no

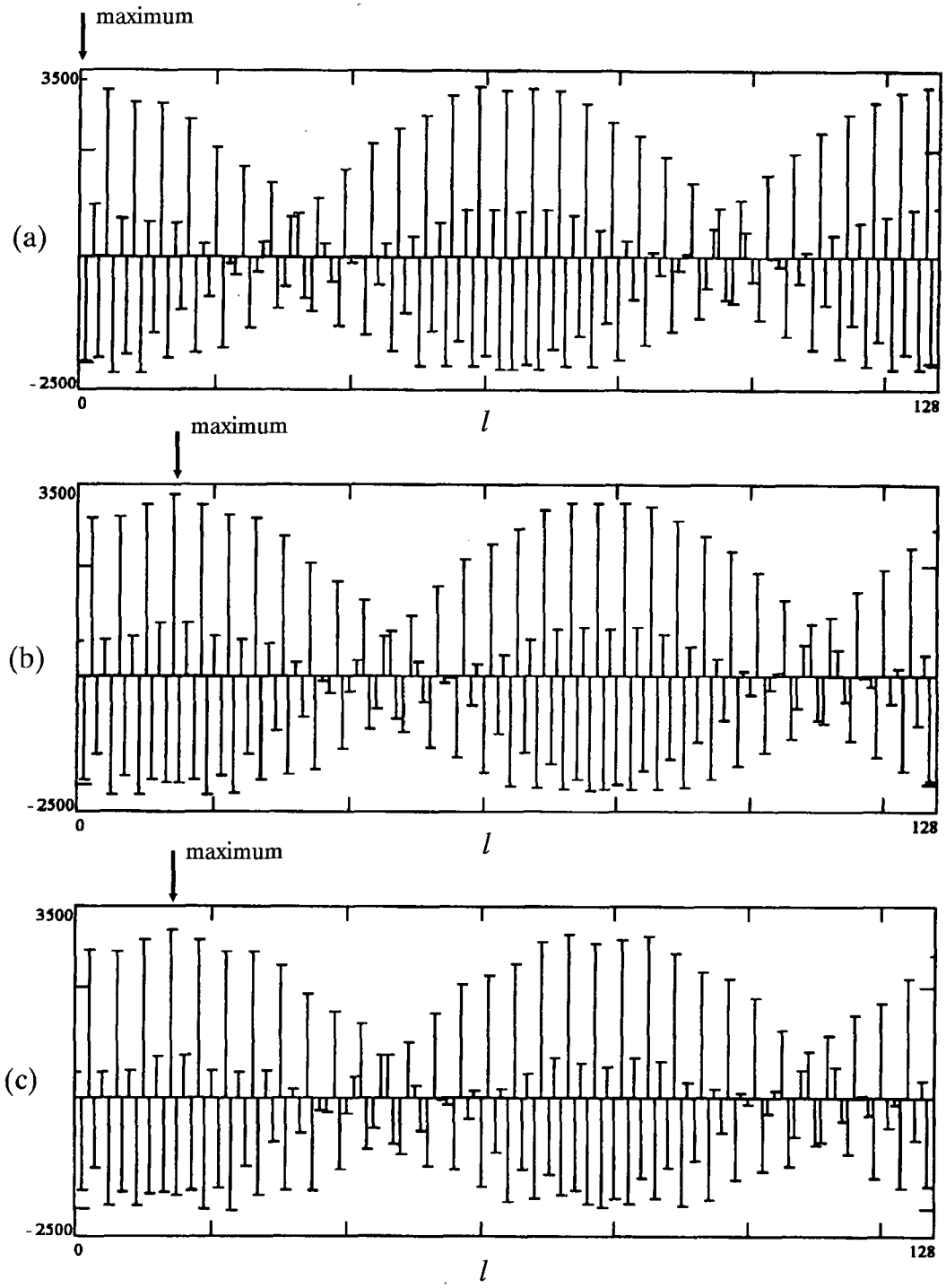


Fig. 8-10 Correlation results of :

(a) 2 regularly sampled sequences x(n) and s(n) with x(n) = s(n) = $\sin(2\pi.18n/256) + 3\cos(2\pi.65n/256) - 4\cos(2\pi.126n/256)$. $R_{sx}(0)$ *is the maximum value in the resulting sequence.*

(b) 2 regularly sampled sequences x(n) and s(n) with x(n) = s(n-14). $R_{sx}(14)$ *is the maximum value in the resulting sequence.*

(c) 2 sequences same as in (b) but sampled by a.r.s. with $x(t_n) = s(t_n - 14\, t_s)$. *Eqn (8-9) is used for the evaluation with* $d = t_s$.

5.475, -5.685, 3.434, . . . }, $s_{18}(t_n)$ = {-6.916, 0.6181, -1.802, 3.997, -6.17, 2.797, . . . } and $s_{73}(t_n)$ = {-7.507, 2.713, -1.173, 4.921, -6.134, 3.799, . . . }. To make the result of a matched template distinct, instead of performing the correlation, we can compare the magnitude of each data point between the input and the template, i.e. checking whether $|input\ [i] - template\ [i]| < \delta$ where δ is a preset threshold, and count the number of pairs satisfying the requirement. Fig. 8-11 shows the block diagram of the scheme and Fig. 8-12 shows the counts obtained from masking three inputs with the same set of templates. The delay interval d is equal to one averaging sampling period t_s. As the correlation is performed in regular intervals, the count will repeat after $l = N/2$, where $N = 1/t_s$. From Fig. 8-12 (a) and (b), it is clear that the delay in a randomly sampled sequence can be determined uniquely up to N/2. A set of counts is tabulated in Table 8-3. In each of the first three rows of the table, a large count indicates a detection of the signal at a delayed interval. Row three illustrates how resistant to noise the scheme is. Adding a random noise of 0.58 V r.m.s. to the input signal in row two decreases the maximum count from 218 to 187, but raises the second

Table 8-3 : Masking $s_l(t_n) = 1.5 \cos [2\pi.50\ (t_n - l.t_s)] + 2 \sin [2\pi.506\ (t_n - l.t_s)] + \cos [2\pi.1400\ (t_n - l.t_s)]$ with different inputs $x(t_n)$. δ = 5% of $|s_l(t_n)|$.

Input $x(t_n)$ =	max. count	l
$s_{14}(t_n)$	239	14
$s_{200}(t_n)$	218	200
$s_{200}(t_n)$ + noise	187	200
$2 \sin (2\pi.506\ t_n) + \cos (2\pi.1400\ t_n)$ + noise	32*	253
$1.5 \cos (2\pi.50\ t_n)$ + noise	18*	245
$\cos (2\pi.1400\ t_n)$ + noise	19*	83
$1.5\cos(2\pi.5t_n) + 2\sin(2\pi.501t_n) + \cos(2\pi.1402t_n)$	17*	136

*low counts which will be rejected by thresholding.

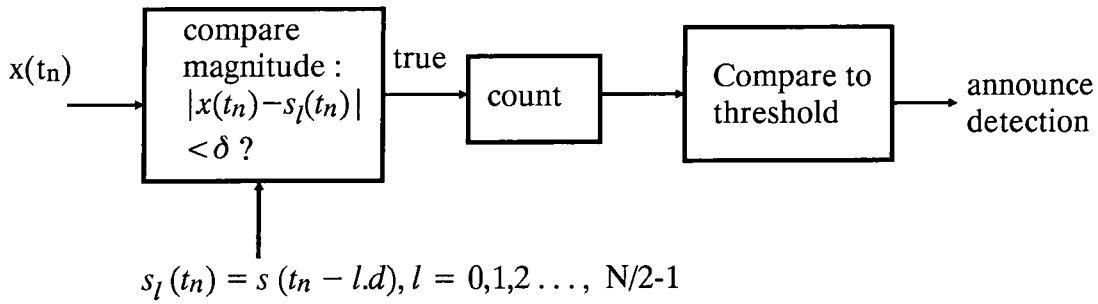


Fig. 8-11 Correlation by comparing the magnitude of input with a set of templates.

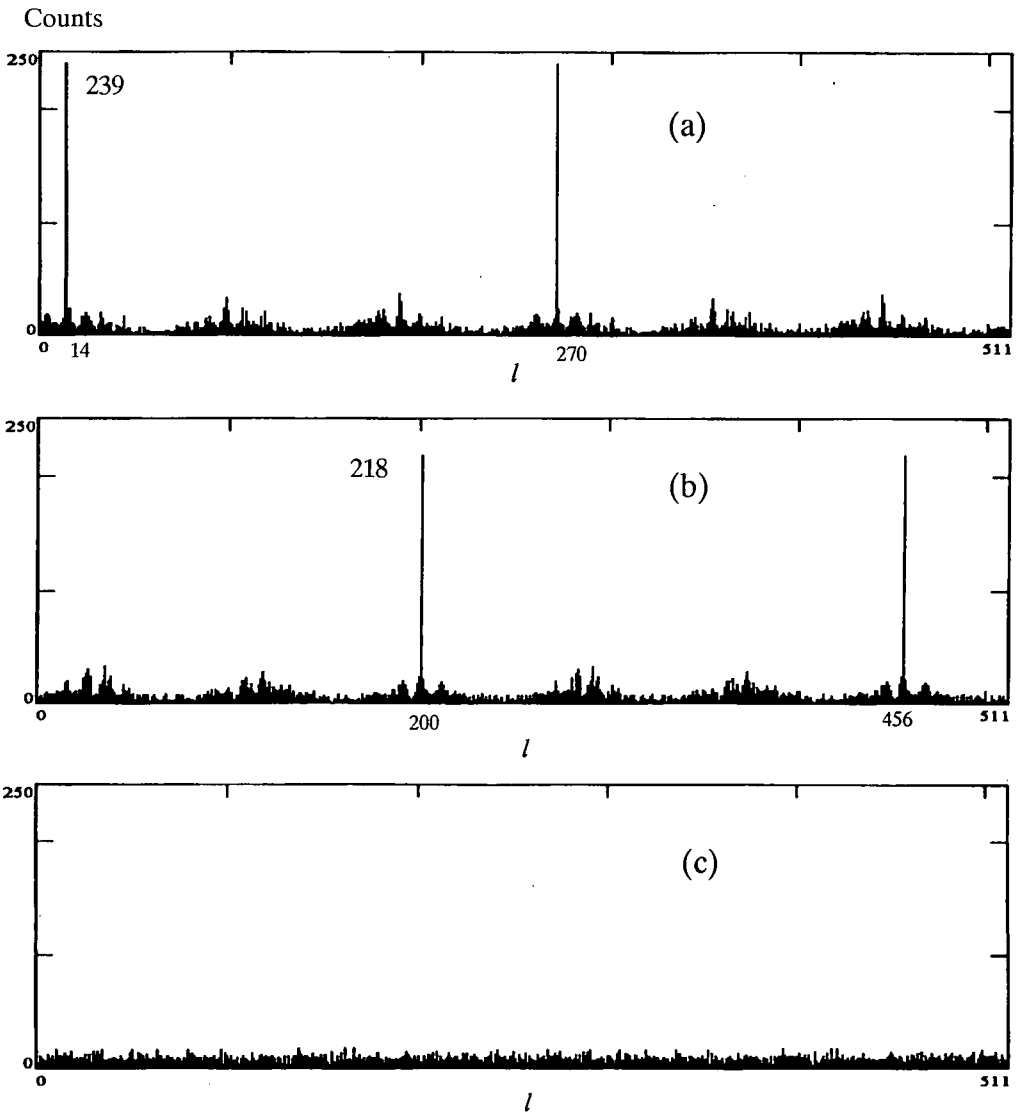


Fig. 8-12 Masking $s(t_n - l.d)$, *where* $s(t_n) = 1.5\cos(2\pi.50t_n) + 2\sin(2\pi.506t_n) + \cos(2\pi.1400t_n)$, $l = 0,1,2,\ldots 511$, $d = 1/512$ *and* $\delta = 0.05\,|s_l(t_n)|$, *with :*

(a) $x(t_n) = s(t_n - 14d)$ *sampled by jittered random sampling.*

(b) $x(t_n) = s(t_n - 200d)$ *sampled by additive random sampling.*

(c) $x(t_n) = 1.5\cos(2\pi.5t_n) + 2\sin(2\pi.501t_n) + \cos(2\pi.1402t_n)$ *sampled by a.r.s. Low counts are obtained.*

largest count (not tabulated here) slightly from 32 to 33 only. The last four rows show cases of mis-matches where the counts are low.

8.3.3 Advantages and Disadvantages : Random sampling is one of the sub-Nyquist sampling methods. Its anti-alias property in the cross-correlation of two similar signals is shown in Table 8-2 and Fig. 8-9. As fewer samples require fewer operations for a correlator, random sampling has an advantage over regular sampling in detecting periodic signals, such as the image of a texture pattern or grating.

Alignment of timing between the input and the template is necessary whether sampling is performed regularly or irregularly. To obtain useful cross-correlation results, the initial sampling times of both signals must be aligned to a suitable point, for example, the beginning of a periodic waveform. If the input and the template are continuous signals that can be synchronized and sampled simultaneously, alignment is achieved. Without synchronization, alignment is also feasible by trial and error for signals coming from a source that can be re-sampled. Assuming that the the input is in phase with the template, i.e. no delay between the two signals, only one template is required. In this case, alignment should be accurate within one average sampling period since the correlation result is still acceptable within such a phase error. For example, in row 1 of Table 8-2, if the input signal is delayed by one average sampling period (= 1/512), the result will become 1,452.1, which is 78% of the value of a perfect match.

If delays in terms of a whole multiple of the average sampling period are introduced in the input, the scheme depicted in Fig. 8-11 may be used to detect the delays. In order to save computation time, the set of templates required can be generated beforehand but a memory space must then be assigned to store them.

Performing magnitude comparison takes approximately the same amount of time as computing correlation. Simulation results for 512-point sequences matching with 512 templates indicate that computation time is increased slightly by 4% as compared to correlation. Simulation also reveals that the phase error in time alignment must be less than 4% of a sampling period, which is true for both randomly and regularly sampled sequences.

8.4 Concluding Remarks

In this chapter, two applications in digital signal processing which exploit the anti-alias property of random sampling have been proposed. The first application, which is to detect a moving object in a sequence of images, requires spectral reconstruction. As discussed previously in Chapter 3, random sampling loses the advantage to apply the FFT, which will slow down the process when a general purpose computer is used. Fortunately, the sequence length in this application is usually so short, say a few tens of frames, that computation will not become a prohibitive factor. The second application, which is the correlation detector, operates in the spatial domain and requires no extra work-load for computing a spectrum, although for the sampling process itself, a more complicated hardware, such as a synchronized circuit, may be needed for taking sample values from the input signals.

A detailed discussion on the criteria to choose random sampling for an engineering application will be found in Chapter 10, the conclusions.

CHAPTER 9

RECONSTRUCTING RANDOMLY SAMPLED SIGNALS BY THE FFT

9.1 Introduction

In regular sampling, the DFT is a reversible operation since the transform kernel is a set of orthogonal functions. When a sequence is randomly sampled, the randomized timing destroys the orthogonality of the transform kernel for estimating the frequency spectrum, which can be seen in eqn (3-17) and repeated below :

$$X(k) = \frac{2}{N}\sum_{n=0}^{N-1} x(t_n)\, exp(-j2\pi k t_n), \qquad k = 1,2,3,\ldots \tag{9-1}$$

Hence given a set of frequency components X(k) and sampling times t_n, we cannot recover the time sequence $x(t_n)$ from an inverse DFT corresponding to eqn (9-1). Bilinskis and Mikelsons [28] propose an unorthogonal transform method which is basically a minimization of the mean square error of a reconstructed time sequence. Applying this method to reconstruct a randomly sampled sequence will involve heavy computation in manipulating the transform kernel, but the reconstructed signal may not keep the details of the original waveform even though a minimization process is performed.

When a spectrum is estimated by eqn (9-1), the resulting spectrum X(k) is in fact mapped to a regular grid in the frequency domain. Therefore, when X(k) is eventually transformed to a time sequence x(n), the result will correspond to a

regularly sampled version of $x(t_n)$. In this reconstruction, there are three conditions to be satisfied :

(1) The reconstructed signal x(n) must be a real sequence since the input $x(t_n)$ is real.

(2) The sequence length of the reconstructed signal x(n) must be long enough to convey the required frequency information although the original sequence $x(t_n)$ may be sampled at a sub-Nyquist rate.

(3) The broadband noise in the frequency spectrum introduced by random sampling must be eliminated.

9.2 Reconstruction on a regular time grid

9.2.1 Length of Sequence : In eqn(9-1), N is the number of samples taken in the time domain and k corresponds to the index of the frequency component being evaluated. In regular sampling, the spectrum can be evaluated uniquely to only N/2 - 1 when the length of the input time sequence is N. Nevertheless, with random sampling, the frequency index k should go beyond this limit in order to take advantage of the sub-Nyquist sampling.

In general, we have a randomly sampled real sequence $x(t_n)$ of length N and we may evaluate its frequency spectrum X(k) to a length M such that $M > N$. The spectrum X(k) is now on a regular grid but its value is unique up to $k = M-1$. If the input sequence $x(t_n)$ were regularly sampled, to obtain M unique frequency components, we should have sampled 2M points and the fold-over frequency in its spectrum would be at $k = M$. To exploit this symmetry property of a real signal, we can extend X(k) to 2M points by putting :

$$X(M) = 0$$

$$Real\left\{X(2M-k)\right\} = Real\left\{X(k)\right\}$$

$$Imag\left\{X(2M-k)\right\} = -Imag\left\{X(k)\right\} \qquad \textbf{(9-2)}$$

where k = 1,2,3,. . . M-1, and *Real* and *Imag* stands for the real and imaginary parts of X(k) respectively. Then we can perform an inverse DFT (or FFT if M is a power of 2) on the extended spectrum and the result x(n) of length 2M will be a regular time sequence containing all the information carried by X(k).

9.2.2 Filtering of Noise : Filtering in the frequency domain can be implemented by choosing a suitable window function. The simplest window is, of course, the rectangular window, but it will create an adverse effect, which is the well-known Gibb's phenomenon, in the recovered time sequence [55,56]. There are many different windows available to solve this problem, such as the Hanning window, the Hamming window, the Blackman window, etc., and the choice is not crucial for this application. On the other hand, we need some knowledge of the characteristics of the spectrum so as to determine which type of filter, e.g. low-pass or band-pass, we should apply to recover the desired information.

9.2.3 Examples : Three examples are to be given : (i) a rectangular wave, (ii) a triangular wave and (iii) a triangular wave mixed with two sinusoidal waves. The first two waves are reconstructed by low-pass filtering below the Nyquist frequency. The purpose is to show the effects of windowing. The third example demonstrates the alias-free property of random sampling by extracting components above the Nyquist frequency.

Fig. 9-1 shows the process of recovering a rectangular wave from a randomly sampled sequence of length 256. The sampling sequence is a jittered sampling

sequence, of which the ratio of the standard deviation of the random variable to the mean sampling period (σ/μ) is equal to 33%. Fig. 9-1 (b) depicts the amplitude spectrum of the randomly sampled sequence. Since the information of the input signal lies mainly in the low-frequency band, a low-pass FIR filter using the Hamming window may be used to filter out the wide-band noise generated by random sampling. The Hamming window is defined as :

$$w_h(k) = \begin{cases} 0.54 + 0.46\cos(2\pi k/m), & -(m-1)/2 \le k \le (m-1)/2 \\ 0, & \textit{elsewhere} \end{cases} \qquad \textbf{(9-3)}$$

where $w_h(k)$ is the window coefficient and m, an odd integer, is the length of the filter. The spectrum, after modified by a Hamming window with m = 31 (cut-off frequency k_c = 15) and extended according to eqn(9-2), is shown in Fig. 9-1(c). Its inverse DFT, i.e. the recovered signal in the time domain, is shown in Fig. 9-1(d). In this example, although the length of the resulting sequence is doubled to 512 points, it can actually be down-sampled to 256 points without losing any details in the waveform.

The effects of windowing are illustrated in Fig. 9-2, where Hamming windows of k_c = 15, 30 and 50 are used to filter the spectrum in Fig. 9-1 (b). The respective mean square errors of the reconstructed signals are 6.15×10^{-3}, 1.82×10^{-3} and 1.00×10^{-3} . It is obvious that a wider window returns sharper corners and edges in the recovered waveforms since these features correspond to the high frequency components in the spectrum. However, passing more high frequency content into the output signal will generate larger ripples on the top part of the waveform which should be flat. Another example with a triangular wave as input is shown in Fig. 9-3. The same effects are also visible.

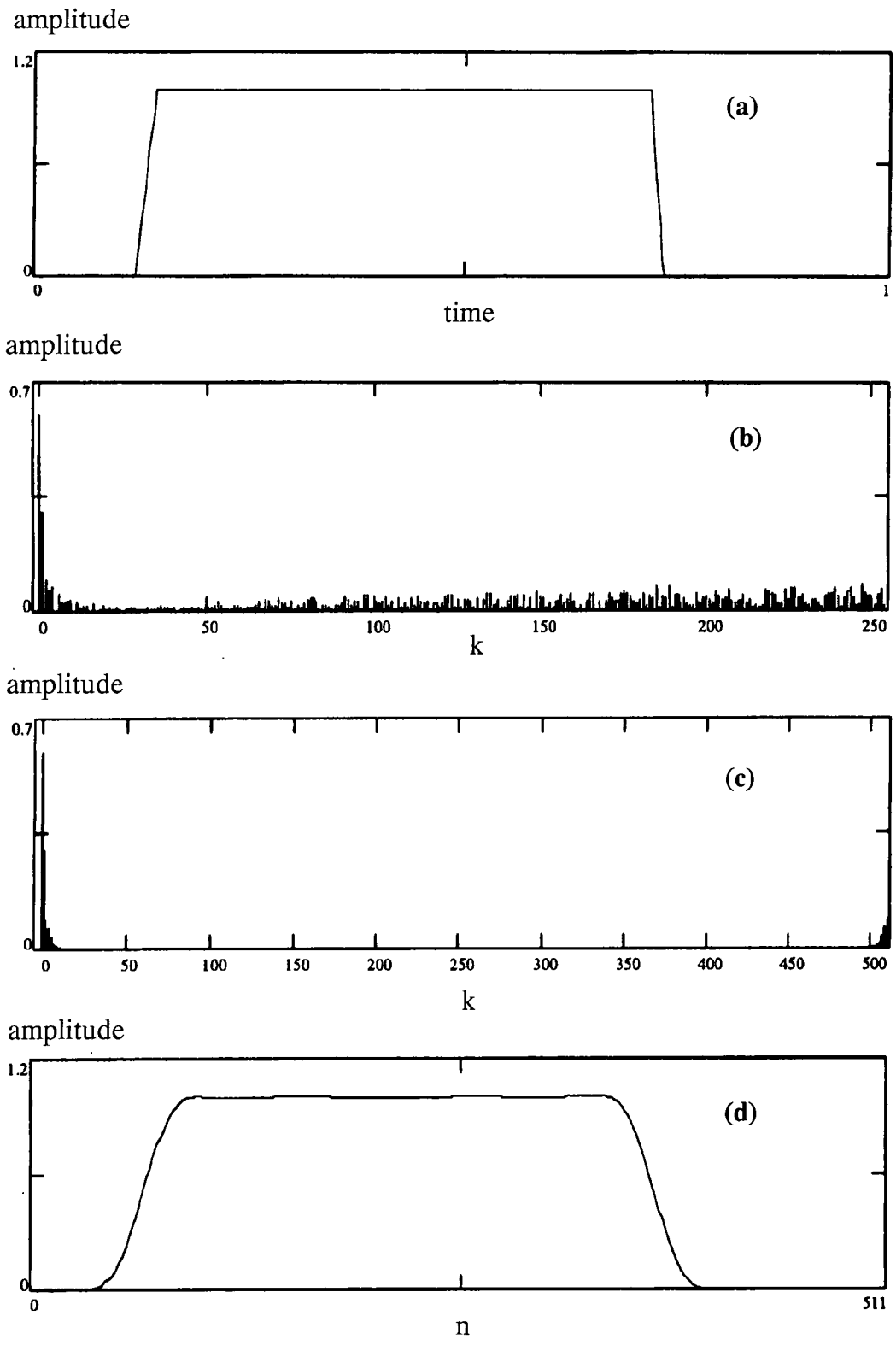


Fig. 9-1 Reconstructing a rectangular wave :

(a) the input signal,

(b) the amplitude spectrum from a 256-point sequence sampled from (a) by jittered random sampling;

(c) the amplitude spectrum in (b) after filtering and extending to 512 points;

(d) result after performing the inverse DFT on (c).

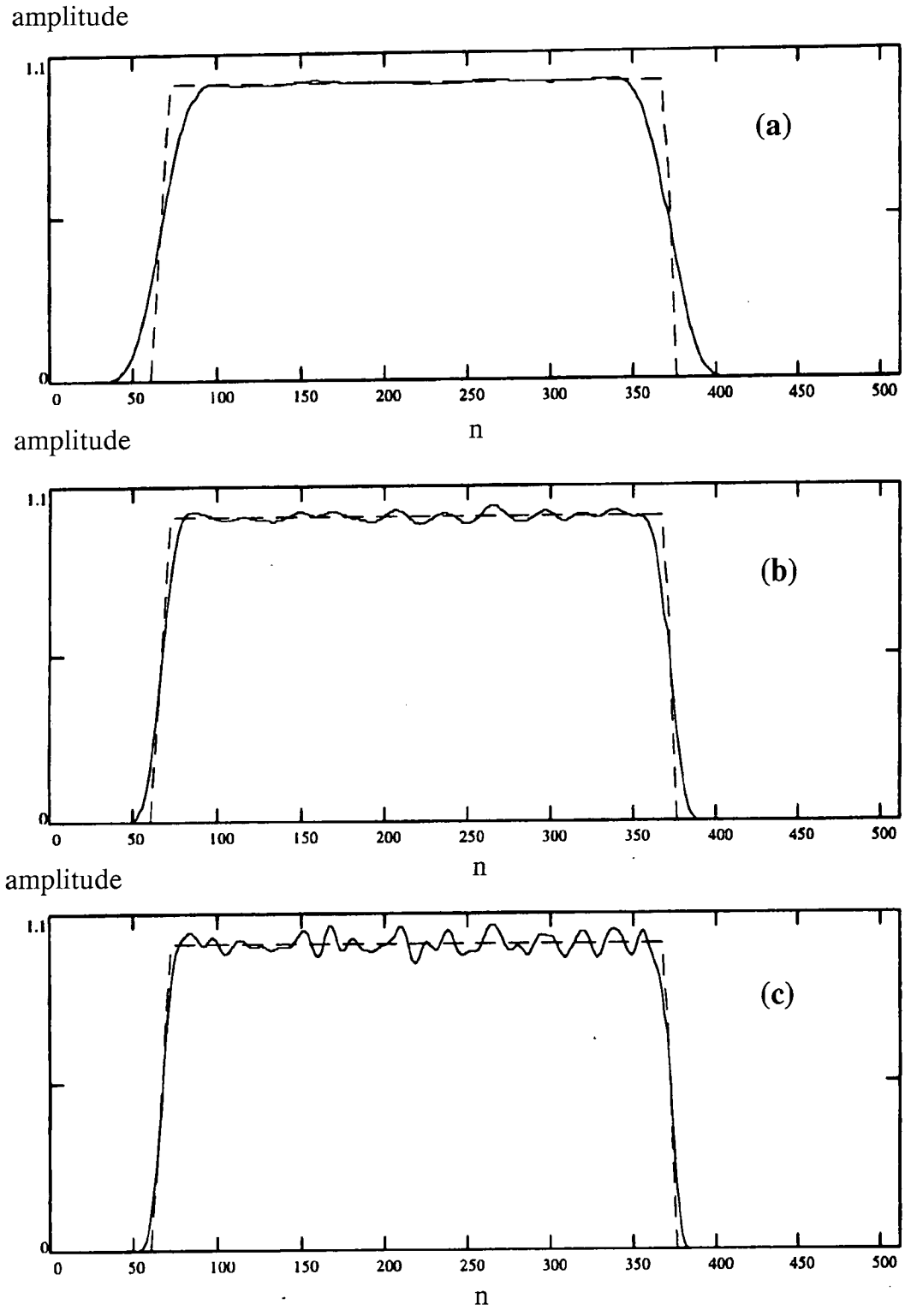


Fig. 9-2 Results of reconstructing a rectangular wave by random sampling and filtering with Hamming windows of a cut-off frequency :

(a) k_c = 15; mean square error = 6.15×10^{-3};

(b) k_c = 30; mean square error = 1.82×10^{-3};

(b) k_c = 50; mean square error = 1.00×10^{-3}.

In each plot, the dash line represents the original signal.

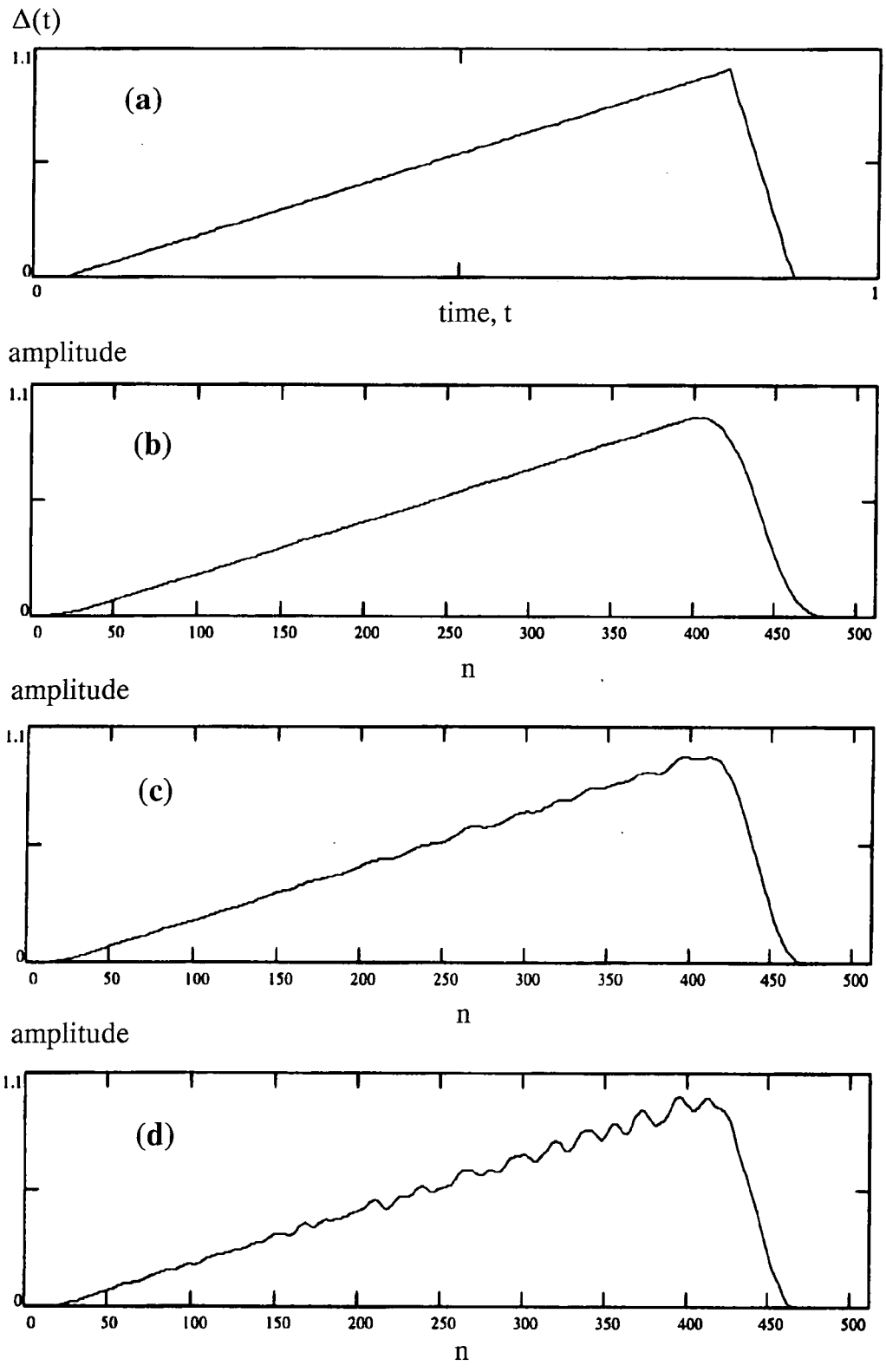


Fig. 9-3 (a) A triangular wave, Δ(t), and its reconstruction by random sampling and filtering with different cut-off frequencies :

(b) k_C = 15; mean square error = 5.10 x 10^{-4};

(c) k_C = 30; mean square error = 1.18 x 10^{-4};

(d) k_C = 50; mean square error = 2.97 x 10^{-4}.

The third example shown in Fig. 9-4 utilizes the alias-free property of random sampling in reconstructing a time sequence. The reference signal, depicted in Fig. 9-4 (a), consists of the triangular wave $\Delta(t)$ plotted in Fig. 9-3 (a) mixed with two sinusoids, $0.5 \sin(2\pi.150t)$ and $0.4 \cos(2\pi.154t)$. To exploit the sub-Nyquist sampling, the signal is sampled randomly for 256 points so that the frequency indices of the two sinusoids are above the Nyquist limit at 128. The resulting spectrum is plotted in Fig. 9-4 (b) and it is filtered and extended to 512 points as shown in Fig. 9-4 (c). The filtering consists of three Hamming windows, which are a low-pass filter centred at $k = 0$ with a length of 31 ($k_c = 15$) and two band-pass filters centred at $k = 150$ and 154, both having a length of 5. Even if we do not know beforehand the characteristics of the input, the choices of such filters can also be made basing on the spectrum in Fig. 9-4 (b), which clearly indicates low frequency content near the d.c. region and two sinusoids at $k = 150$ and 154.

Since the reconstructed sequence will be regularly spaced, the Nyquist criterion applies. Having performed the inverse DFT, the resulting sequence shown in Fig. 9-4 (d) has a length of 512, which is a suitable length for conveying the frequency information. If it is down-sampled to 256 points, the result will be identical to the samples of the triangular wave $\Delta(n)$ plus the aliases of the two input sinusoids, which are $0.4 \cos(2\pi.\frac{102n}{256}) - 0.5 \sin(2\pi \times \frac{106n}{256})$. As the highest frequency content is 154, the fold-over of the spectrum can in fact be made at any point above that value, say 160, thus producing a sequence of length 320. The result of this reconstruction can be seen in Fig. 9-5 and the mean square error is found to be the same as with length 512. However, if the FFT is adopted for computing the inverse, 512 is a reasonable length

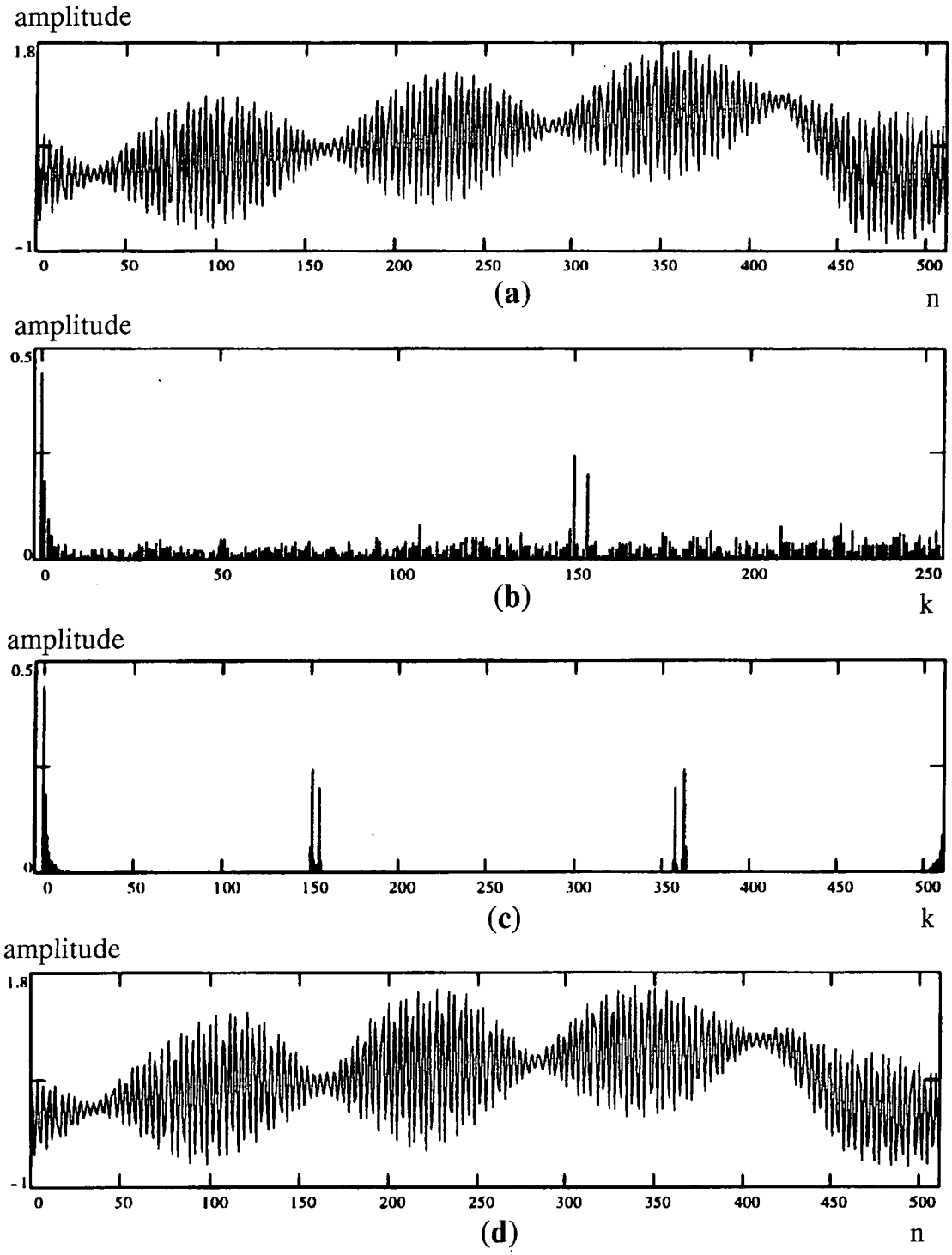


Fig. 9-4 Recovering a signal sampled at a sub-Nyquist rate :

(a) the reference signal : the triangular wave $\Delta(t)$ in Fig. 9-3 (a) + 0.5 sin ($2\pi.150t$) + 0.4 cos ($2\pi.154t$) sampled regularly for 512 points;

(b) the amplitude spectrum of the reference signal sampled randomly for 256 points; the frequencies of the two sinusoids are above the Nyquist limit at 128;

(c) the amplitude spectrum in (b) after filtering and extending to 512 points;

(d) the signal recovered by applying the inverse DFT to (c); mean square error = 1.53×10^{-2} .

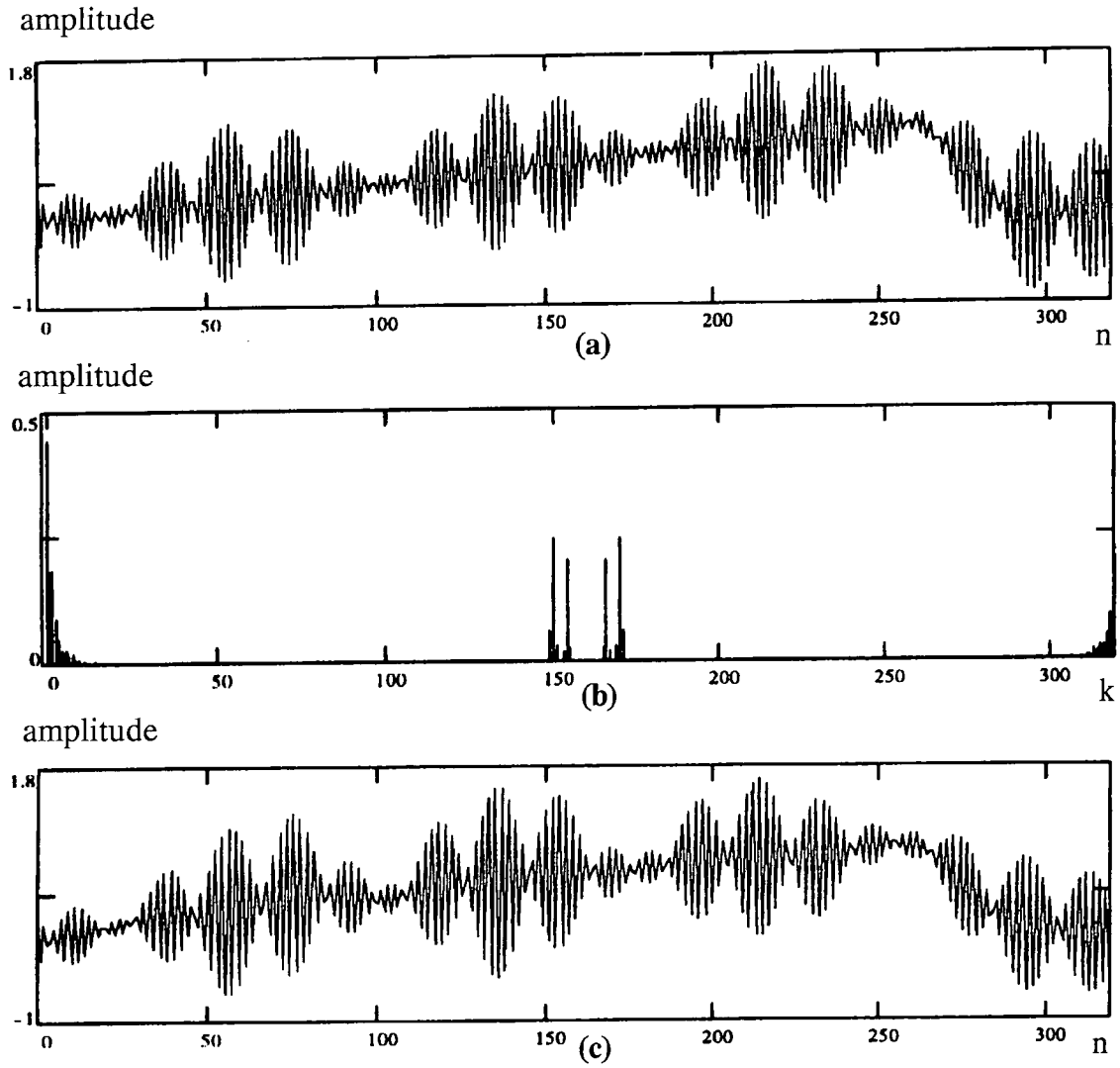


Fig. 9-5 Reconstruction of the composite wave $\Delta(t) + 0.5 \sin(2\pi.150t) + 0.4 \cos(2\pi.154t)$ using a sequence length of 320 :

(a) the input sampled regularly for 320 points and taken as the reference signal;

(b) the amplitude spectrum in Fig. 9-4 (b) after filtering and extending to 320 points;

(c) the signal recovered from the inverse DFT of Fig.9-5 (b); mean square error = 1.53 x 10^{-2}.

Table 9-1 : Mean square errors of the reconstructed signals.

Waveform	Shown in	Low-pass cut-off	Mean square error
Rectangular	Fig. 9-2(a)	$k_c = 15$	6.15×10^{-3}
Rectangular	Fig. 9-2(b)	$k_c = 30$	1.82×10^{-3}
Rectangular	Fig. 9-2(c)	$k_c = 50$	1.00×10^{-3}
Triangular	Fig. 9-3(b)	$k_c = 15$	5.10×10^{-4}
Triangular	Fig. 9-3(c)	$k_c = 30$	1.18×10^{-4}
Triangular	Fig. 9-3(d)	$k_c = 50$	2.97×10^{-4}
Composite	Fig. 9-4(d)	$k_c = 15$	1.53×10^{-2}
Composite	Fig. 9-5(c)	$k_c = 15$	1.53×10^{-2}

since it produces a Nyquist frequency of 256, which is the smallest power of 2 greater than 154.

9.3 Concluding Remarks

The previous three examples illustrate how a time sequence may be recovered from the spectrum of a randomly sampled signal by filtering, extending the spectrum and performing the inverse DFT (or FFT). For a regularly sampled sequence, a direct inverse of its frequency spectrum leads to a perfect reconstruction of the time sequence, but for random sampling the two extra processes mentioned above are essential because filtering preserves the recovered signal from distortion and the extension of the spectrum guarantees the existence of a real sequence after the inverse transform. With the inversion process made possible, a two-way transform between the time and the frequency domains for random sampling is complete. The usefulness of this sampling method can be broadened, such as to reconstruct one-dimensional patterns, and is no longer limited to spectral estimation. Computational complexity is not a concern as the FFT can be used for finding the inverse provided that a suitable sequence length is chosen.

The advantages of adopting random sampling are obviously its alias-free property and the reduction in the number of samples. Tomography is one example that might benefit from adopting random sampling since data from a patient are acquired by x-ray scanning, in which reducing the number of samples is highly desirable. Although it is straightforward to extend random sampling from one dimension to two dimensions, the effects of random sampling to the Radon transform, which is involved in computing tomography, are yet to be studied.

CHAPTER 10

CONCLUSIONS

10.1 A Brief Review

In various fields of engineering involving the use of digital signal processing, input signals may be processed by algorithms which require heavy computation. Nowadays, with the development in fabrication technology that produces fast chips (microprocessors, memory, etc.) at a low cost, computing power is no longer a problem in most cases, although computing speed may still be a concern in real-time applications. In the ages when only vacuum tubes and transistors were available, it was obviously important to develop methods that could simplify or speed up the computation. Some examples of those methods are the Monte Carlo simulation, statistical estimation and stochastic computation. Randomized signal processing (including sampling, quantization, correlation, etc.), which is also a statistical approach, was proposed and developed during that time. Apart from being a mathematical topic, random sampling finds its applications in engineering. By using this technique to extend the input frequency range in measuring a signal, Hewlett-Packard produced and marketed successfully high-quality instruments such as the sampling voltmeter HP 3406 A (in 1985) and the high-speed digitizing oscilloscope HP 54100 A/D. According to Bilinskis and Mikelsons [28], "Hewlett-Packard is the only company which has enough knowledge of, and confidence in, this approach to apply it over and over again with remarkable results".

10.2 Relationship between Random and Regular Sampling

The conventional sampling method assumes that all sampling instants occur precisely at regular intervals, but in practice this may not be true as the sampling time might be affected by noise or the inaccuracy of the hardware. While one might simply ignore this minor deviation, there were researchers who showed interest in studying how the inexact timing would affect the frequency spectrum [24,29]. When the deviation is considered to be a random variable in time being added to each occurrence of a regular sampling impulse, the result is identical to a randomized sampling sequence. Hence randomized sampling may be related to regular sampling through this field of study.

Mathematical proofs of the alias-free property of random sampling can be dated back to 1960 [24]. Conceptually, random sampling may be considered a pseudo-continuous sampling process as a sampling impulse can occur, with a certain probability, at any instant throughout a sampling period, which implies that the separation between two successive sampling periods can be infinitely small. In other words, the effective sampling rate is very high. This topic is discussed in Chapter 6 with the aid of auto-correlation which converts a randomly sampled sequence to its power spectrum on a regular grid.

The major motivation of using randomized sampling is its alias-free property which enables an input signal to be sampled at a sub-Nyquist rate. In fact, sub-Nyquist sampling can also be achieved by another approach - manipulating several regular sampling sequences to obtain more information from the input signal. Underhill et. al. [3] proposed to sample the input independently by three sampling sequences of a slightly different frequency. Kohlenberg[57], and recently Coulson [58], discussed an

Nth order nonuniform sampling where N uniform sample streams of separation T were interleaved with time offsets $k_i < T$ between successive streams. In principle, the above methods resolve the ambiguities in the frequency domain by the properties of the aliases themselves. However, when random sampling is adopted, aliases are turned to a broadband noise and thus no ambiguities arise in the spectrum.

10.3 Contributions of This Thesis to Random Sampling

10.3.1 Estimation of Noise Spectral Density and Bandwidth : When a sequence is randomly sampled, an alias-free spectrum can be recovered from a sequence sampled below the Nyquist rate. In exchange, we have to pay a much higher cost in computation and to tolerate a background noise in the whole spectrum. In section 3.3.1, it is pointed out that the computational complexity is N^2, and in section 3.3.2 , the noise spectral density per watt input in the spectrum is shown to be approximately $1/N^2$ watt per frequency resolution, where N is the length of the data sequence. Theoretically the bandwidth of a randomly sampled sequence is infinite, but in practice it is limited by the word-length of the host system. Detailed discussion is found in section 3.3.3.

10.3.2 Sampling Methods and Computational Algorithms : Two methods of speeding up the calculation of a spectrum, which are based on the principle of inserting limited regularities into random sampling sequences, are proposed and described in Chapters 4 and 5. In Chapter 4, parallel additive random sampling is exposed to show how it can save 87% of the multiplications required. Since this scheme forms a naturally parallel process, several processors may be employed to compute the spectrum simultaneously, which can further speed up the evaluation. When examining the resulting spectrum, it is seen that aliases are turned into bursts of noise. Should these

bursts be found undesirable, an adaptive thresholding is suggested to filter them out. Another sampling procedure without generating these bursts of noise is hybrid additive random sampling, which is described in Chapter 5. Saving in multiplications is between 75% to 87% depending on the format chosen.

10.3.3 Study of Auto-correlation Sequences : In Chapter 6, a technique of determining the auto-correlation of a randomized sequence using a regular step size is proposed and elaborated. With the aid of auto-correlation, the anti-alias characteristics of parallel additive random sampling and hybrid additive random sampling are further examined and verified. This technique may also be used as a means to convert a randomly sampled sequence to an equivalent regularly spaced sequence having a desired Nyquist frequency.

10.3.4 Rapid Spectral Estimation using a Coarse Kernel : Since most fast algorithms were based on the principle of reducing multiplications[1], researchers would consider simplifying the transform kernel as such to obtain a similar effect. Walsh and Hadamard transforms are examples that use only 1 and -1 in their transform matrices. Parallel to this line of thought, one might look for a simplified representation of the kernel of the Fourier transform. For example, we may approximate the sine and cosine functions by binary fractions (½ , ¼ , . . ., etc.), as in computer arithmetic these multiplications can be realized by a linear shift of the multiplicand. To push this idea further, a two-level representation by 1 and -1 [4] and a three-level representation by 1, 0 and -1 [5] of the sine and cosine functions were proposed. The motive in so doing is, of course, that all multiplications become additions and subtractions. (This method

1. With the advent of microprocessor having a built-in mathematical co-processor capable of performing floating point multiplications, e.g. Intel 80486 and Pentium, the cycle time of a multiplication is in the same order of magnitude to an addition. When designing a fast algorithm for these processors, one should consider minimizing the total operations in multiplications and additions rather than replacing multiplications by additions.

is known by various names. It is cited as the Poorman's transform by Lamoureux [59].) Although this approximation can be applied to uniformly sampled sequences as seen in the examples of the above mentioned papers, the computation process is slower than the FFT unless special hardware (but not a general purpose computer) is used for the evaluation because its complexity for additions is N^2. Therefore the approximation method is particularly suitable for estimating the spectrum of a randomly sampled sequence to which the FFT cannot be applied. In Chapter 7, the three-level kernel is analysed and recommended for random sampling.

10.3.5 Applications in Digital Signal Processing : Application examples in the literature of random sampling are mostly related to instrumentation for spectral estimation. In Chapter 8, two applications in the field of digital signal processing are proposed. They are a method for detecting a moving object in an image sequence and a correlation detector.

10.3.6 Inverse Transformation : In regular sampling, the Fourier transform is reversible. An inverse transform always exists as the kernel is orthogonal and transform between the spatial (or time) domain and the frequency domain is straightforward. Random sampling, however, destroys the orthogonality and makes the inverse transform impossible. In Chapter 9, a reconstruction process which converts the spectrum of a randomly sampled sequence to a uniformly spaced time sequence is suggested. Although the process is rigorously not an inverse transform, it serves as a means to channel data from the frequency domain to the spatial domain, which is required by many applications. The conversion process may now be regarded as two-way and complete.

10.4 Usefulness of Random Sampling

The advantage of using random sampling is apparently the possibility of recovering an alias-free spectrum from a sequence sampled at a sub-Nyquist rate, which is definitely a feature superior to regular sampling. Before jumping on the bandwagon, however, one should consider the whole picture of a certain application when adopting random sampling. Typical aspects for consideration are the mode of operation, the computation load and the type of hardware available.

Whenever the situation of an application permits, computation should be avoided or minimized. Computation arises when a spectrum is evaluated. Hence it is more effective to manipulate the data in the data domain without taking any transforms. An example illustrated in section 3.3.5.3 is a sampling oscilloscope, which reconstructs a periodic waveform in the time domain. Another efficient use of random sampling includes those applications requiring minimal computation. Examples are digital r.m.s. voltmeters and wattmeters by which the quantity to be measured is made proportional to the d.c. term of the spectrum [2]. As the d.c. term is obtained by averaging, no multiplications are needed.

In terms of efficiency, the radix-2 FFT is the best algorithm for computing the DFT when a general purpose computer is used as the host system. Unfortunately, with a randomly sampled sequence, the FFT cannot be applied. A simple (but expensive) solution is to build a concurrent structure composing of many fast multipliers and adders. (To the extreme, there are as many multipliers and adders as the number of data points. A result will be obtained after the cycle of one multiplication and one addition.) A more cost-effective approach is to replace multiplications by linear shifts, which can be realized if the values of the sine and cosine functions of the transform

kernel are all represented by binary fractions. Suppose the accuracy of computation can be traded for the speed, one may adopt a two- or three-level kernel so as to eliminate all operations related to multiplication. If only general purpose computers are available, one may choose the parallel or hybrid additive random sampling to speed up the calculation.

Although the FFT is in general the most efficient algorithm for computing a spectrum, there is a particular case that a random sampling algorithm may win. An example given at the end of section 5.2.3 shows how fewer multiplications than regular sampling can be achieved with random sampling when a narrow frequency band is computed.

In a word, one must be cautious in recommending random sampling without qualification. There is much speculation about the usefulness of random sampling and the complexity which a nonuniform time axis introduces is esoteric. In general it is always better to use uniform sampling with the FFT except for the specific cases listed above, or when one is dealing with a very broadband single frequency signal such as radio jammer transmissions, or when the cost of sampling is more expensive than computation, etc.

The crux of applying this technique as widely as uniform sampling to spectral estimation is the loading in computation, which is related to the choice of the transform kernel. The insertion of a random variable to the kernel comprising sinusoidal functions destroys the condition for performing the inverse transform as well. Ideally another set of basis functions suitable for random sampling should be generated, but it seems to be too ambitious a task in mathematics. Failing that, fast computational methods could be devised to make random sampling more

competitive. In the time domain, a systematic representation or modelling of a time sequence could be helpful. In spectral analysis, efficient algorithms or effective hardware structures for computation are desirable. As a challenge in mathematics or engineering application, this area is worth studying and will be elaborated in the following section.

10.5 Further Development

10.5.1. Mathematical Representation : Besides using the Fourier transform, the frequency content of a signal sampled uniformly can also be manipulated by operations in the time domain. The most usual case is to convolve an input sequence with a mask or template. Many discrete-time systems, such as recursive filters, linear systems, control systems etc., can be modelled by linear shift invariant (LSI) constant coefficient difference equations and the z transform is a mathematical operator for solving these equations. If a mathematical operator similar to the z transform could be devised for randomly sampled sequences, then a productive study parallel to the discrete-time system would be developed. Unfortunately, a genuine randomized sampling sequence will create a shift *variant* system. Consider the impulse response function of the non-recursive FIR filter shown in section 3.3.5.3. Since the coefficients vary with each set of sampling times, one needs as many equations as the number of the sets of sampling times to fully describe the function. Such a representation is very complicated, if not impossible, to handle.

In case that the analytical approach is still preferred for a non-recursive system, one may accept an approximation using pseudo-random sampling as a substitute. To make this approach successful, a suitable scheme of mapping a set of randomized timing to a uniform reference grid must be designed. Using the concept about

auto-correlation discussed in Chapter 6, the interval T of the reference grid can be determined by the highest frequency of interest f_m according to the Nyquist criterion, i.e. $T \geq \frac{1}{2fm}$. The randomized sampling instants are mapped to the nearest points on the reference scale (see Fig. 10-1). The sample values form accordingly a new sequence which contains many zero values as there should be fewer points in the original grid than the reference. Since the resulting sequence is now equally spaced, all mathematical tools tailored for regular sampling can be applied. Convolution or correlation with other regular sequences is easily performed. However, if the sequence represents a system function and its frequency response is evaluated, noise reflected by those zero coefficients will mar the result. The case is similar to the discussion on the inverse DFT in Chapter 9. The zero coefficients may, of course, be filled up with values by interpolation, but the result will become dubious when too many fictitious values are inserted. Since the pseudo-random sampling is also a

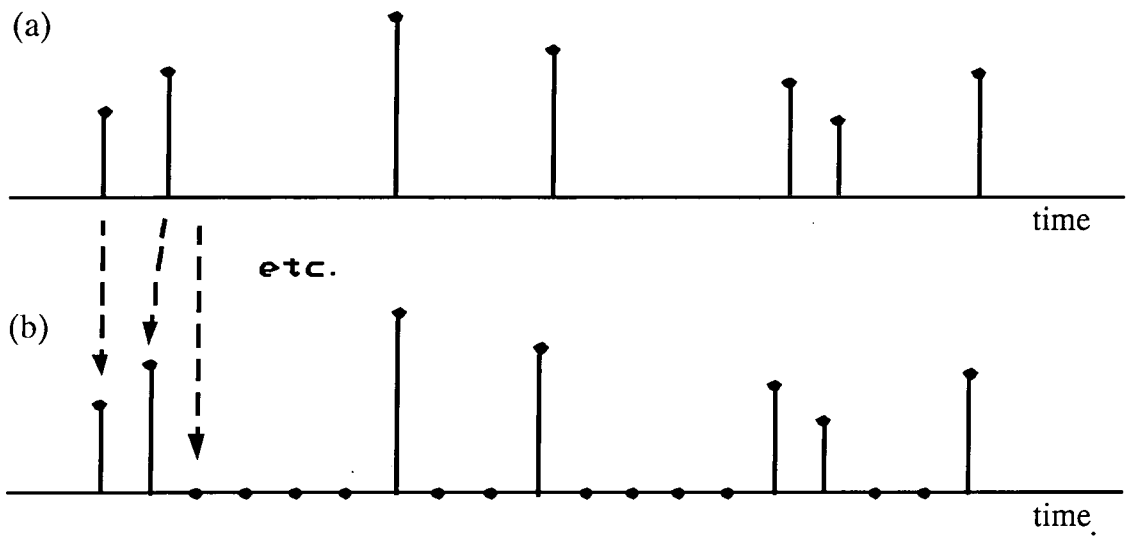


Fig. 10-1 Mapping a sequence with randomized timing to a regular time grid :
(a) The original sequence.
(b) The pseudo-random sequence formed by mapping sample values in (a) to the nearest regular grid points.

random process, the resulting sequence mapped from random sampling retains the original characteristics except the infinite bandwidth. If aliasing occurs after the mapping, a denser grid may be chosen to remove it.

10.5.2 Computational Algorithms : Intrinsically the complexity for computing a spectrum from random sampling is N^2. Any attempt to trim the complexity to a linear class will be in vain. When methods are devised to reduce the number of multiplications, a loss in another aspect is expected. Considering a random sequence of N points being down-sampled to N/2 points and N components being evaluated in the spectrum, the number of multiplications (as well as additions) is $N^2/2$, whereas it would be N^2 for a length-N sequence. Being alias-free, the spectrum will appear the same as the one without down-sampling except that the signal-to-noise ratio is lower. Thus in this case the saving in computation is achieved at the expense of a higher noise level. In other cases, the trade-off may be a lower frequency resolution or a higher hardware cost. Hence the optimal use of random sampling in terms of the above parameters for certain applications forms a topic to study.

One of the motives for designing computational algorithms is to reduce the cost of a hardware structure built for spectral evaluation. For example, when performing a multiplication is much more expensive than an addition, a strategy is to minimize the number of multiplications even if more additions are thus required. Since the cost of integrated circuits is going down, one may eventually opt for the massively parallel structure comprising many processors as described in the previous section. Meanwhile one may adopt a moderate approach of concurrent processing, which requires clever algorithms to break the computation into several independent parts (like the parallel additive random sampling). The concept of pipe-line may also

be exploited whenever possible. Even though the complexity may remain unchanged, the process time is still shortened considerably. The direction of designing algorithms intended for parallel computing should be more rewarding than for a single processor.

10.5.3 Multirate Systems : Returning to the example of down-sampling in section 10.4, one may have noticed the difference caused by the sampling method. For random sampling, only the noise level is affected, whereas for regular sampling, the Nyquist frequency will be lowered. Multirate system with regular sampling is a very fruitful topic, and wavelet transform can also be realized by a collection of multirate filter banks. It is interesting to see whether random sampling can be applied to this area. Can multi-resolution analysis or polyphase representation be established with random sampling? How will the property of perfect reconstruction of filter banks be affected? Can mother wavelets survive randomized timing? Vaidyanathan [44] and Benedetto [60] may be referred to as a starting point for the study.

10.5.4 Two-dimensional Applications : In the spatial domain, a two-dimensional sampling function usually consists of impulses lying on a uniform square lattice. Randomization could be introduced in either one or both of the axes to form a random sampling scheme. Assuming the separability of the transform kernel, the row-column decomposition can be used for estimating the DFT of randomly sampled sequences. Although it is not difficult to realize a two-dimensional random or pseudo-random sampling, it may not be easy to find an application justifiable for using such a process. One would spontaneously connect two-dimensional signal processing with image processing, but to one's disappointment, image processing in general does not benefit from random sampling. Making things even worse, the associated noise will degrade

the quality of an image composed of grey-level or colour pixels. Despite the above disadvantage, at least two successful cases can be quoted. Random and pseudo-random sampling can be adopted for tracking a moving object in a sequence of images as described in Chapter 8. Dithering improves a half-tone picture by removing the blocking effect left behind by compression using transform coding or by any process which manipulates data in blocks.

Another possible candidate to adopt random sampling is tomography. Computed tomography refers to the cross-sectional imaging of an object from either transmission or reflection data collected by illuminating the object from many different directions. With sufficient cross-sectional images (called slices), a three-dimensional model of the object can be established. This method is extremely helpful for exploring nondestructively internal organs of the human body. Sub-Nyquist sampling is an advantage here because, for medical ground, one may prefer to reduce samples which are taken with an x-ray scanner. The mathematical tools utilized in the computation, i.e. the Radon transform and Fourier slice theorem, are related to the Fourier transform [61, 62]. To reconstruct a slice from the projections requires the inverse Radon transform. Hence the effects of random sampling on both the forward and the inverse Radon transform must be carefully studied. A more efficient algorithm for finding an inverse transform is preferred even if the techniques proposed in Chapter 9 are applicable. Adding an extra filtering process prior to finding the inverse, which is already computationally intensive, will be too much a burden. The accuracy of the reconstructed image is a decisive factor affecting the suitability of the application. For locating the position of an internal organ, a high accuracy in the physical dimension of a reconstructed image is required. Suppose it is intended for

examining the function of an organ, then the exactness in the physical dimension may not be a concern. Apparently random sampling is more suitable for the latter.

Sampling may also be performed in the transform domain. For example, in designing an FIR filter, the frequency sampling method is one of the techniques often used. Angelidis proposes to sample nonuniformly in the frequency plane when designing zero-phase FIR filters for acquiring flexibility in the placement of the frequency samples [63]. The computation involves recursive polynomial interpolation, which is claimed to be fast and simple. Pseudo-random sampling is obviously feasible in the case of one-dimensional frequency sampling and its viability in a two-dimensional procedure is yet to be studied. Randomized sampling in the spatial domain gains a wider bandwidth in its frequency spectrum, but this advantage does not exist (or has no meaning) in frequency sampling since the corresponding spatial sequence, such as the impulse response of a low-pass filter, should be band-limited. Instead one has to hunt for other advantages in terms of saving in the computation, the convergence in the solutions, the accuracy of the results, etc. to justify the adoption of such an unconventional process.

References :

[1] Oppenheim, A.V. and Schafer, R.W. : Discrete-time Signal Processing, Prentice Hall, 1989.

[2] Filicori, F., Iuculano,G., Menchetti, A. and Mirri, D. : "Random Asynchronous Sampling Strategy for Measurement Instruments based on Non-linear Signal Conversion", IEE Proc., vol. 136, pt A, no.3, 1989, pp. 114-150.

[3] Underhill,M.J., Sarhadi,M. and Aitchison,C.S. : " Fast Sampling Frequency Meter", Electronics Letters, vol. 14, no.12, 1978, pp. 366-367.

[4] Gaster, M. and Roberts, J.B. : " Rapid Method of Forming Spectra from Rectangular Wave Transforms", IEE Proc., vol. 126, no. 7, 1979, pp. 658-663.

[5] Mason, J.S.D. : " Power Spectra from Approximate Transforms", IEE Proc., pt A, vol. 127, no.4, 1980, pp. 250-25.

[6] Cooley, J.W. and Tukey, J.W. : " An Algorithm for the Machine Computation of Complex Fourier Series", Mathematics of Computation, vol 19, Apr. 1965, pp. 297-301.

[7] Rader, C.M. : " Discrete Transforms When the Number of Data Samples is Prime", the IEEE Proceedings, vol. 56 no. 6, June 1968, pp. 1107-1108.

[8] Kolba, D.P. and Parks, T.W. : " A Prime Factor FFT Algorithm using High-speed Convolution ", IEEE Trans. on ASSP, vol. ASSP-25, no. 4, Aug. 1977, pp. 281-294.

[9] Reed, I.S. and Truong, T.K. : " The Use of Finite Fields to Compute Convolutions ", IEEE Trans. on Information Theory, vol. IT-21, no. 2, Mar. 1975, pp. 208-213.

[10] Proakis, J.G. and Manolakis, D.M. : Introduction to Digital Signal Processing, Macmillan, 1988.

[11] Good, I.J. : " The Interaction Algorithm and Practical Fourier Analysis", J. Roy. Stat. Soc., vol. B-20, 1958, pp 361-372; vol. 22, 1960, pp. 372-375.

[12] Winograd, S. : " On Computing the Discrete Fourier Transform", Proc. National Academy of Science, U.S.A., vol. 73, 1976, pp. 1005-1006.

[13] Nussbaumer, H.J. : Fast Fourier Transform and Convolution Algorithms, Springer- Verlag, 1981.

[14] Siu, W.C. and Lo, K.C. : " A New Hardware-based Realization of Prime Radix Discrete Fourier Transform", Proc., The IEEE Asian Electronics Conference, 1987, Hong Kong, pp. 77-82.

[15] Burrus, C.S. and Eschenbacher, P.W. : " An In-place, In-order Prime Factor FFT Algorithm", IEEE Trans. on ASSP, vol. ASSP-29, no. 4 , August 1981, pp. 806-817.

[16] McClellan, J.H. and Rader, C.M. : Number Theory in Digital Signal Processing, Prentice Hall, 1979.

[17] Burrus, C.S. : " Index Mapping for Multidimensional Formulation of the DFT and Convolution", IEEE Trans., ASSP vol. 25, 1977, pp. 239-242.

[18] Arambepola, B. : "Discrete Fourier Transform Processor based on the Prime Factor Algorithm", IEE Proc., vol. 130, pt. G, 1983, pp. 138-144.

[19] Aloisio, G., Fox, F.C., Kim, J.S. and Veneziani, N. : "A Concurrent Implementation of the Prime Factor Algorithm on Hypercube", IEEE Trans. on Signal Processing, vol 39, 1991, pp. 160-170.

[20] Lo, K.C., Siu W.C., Lun, DPK and Purvis, A. :" Address Generation of Prime Factor Algorithm in a Multiprocessor System", Proc. China 1991 International Conference on Circuits and Systems, vol 1, pp 133-136, June 1991, Shenzhen, China.

[21] Lun, D.P. and Siu, W.C.: "Heterogenous Computing System - SUPERLINK ", Dept. of Electronic Engineering, Hong Kong Polytechnic, 1991.

[22] INMOS Ltd. : Transputer Technical Notes, Prentice Hall 1989.

[23] Marks, R.J. II : Introduction to Shannon Sampling and Interpolation Theory, Springer-Verlag, 1991.

[24] Shapiro, H.S. and Silverman, R.A. : " Alias-free Sampling of Random Noise", J. SIAM, vol. 8, 1960, pp. 225-248.

[25] Beutler, F.J. : " Alias-free Randomly Timed Sampling of Stochastic Processes", IEEE Trans. on Information Theory, vol. IT-16, no. 2, 1970, pp. 147-152.

[26] Masry, E. : " Alias-free Sampling : an Alternative Conceptualization and its Applications ", IEEE Trans. on Information Theory, vol. IT-24, no. 3, 1978, pp. 317-324.

[27] Rudin, W. : Functional Analysis, 2nd Edition, McGraw Hill, 1991.

[28] Bilinskis, I. and Mikelsons, A. : Randomized Signal Processing, Prentice Hall, 1992.

[29] Berkovitz, A. and Rusnak, I. : " FFT Processing of Randomly Sampled Harmonic Signals", IEEE Trans. on Signal Processing, vol. 40, no.11, 1992, pp.2816-2819.

[30] Helfrick, A.D. and Cooper, D.W. : Modern Electronic Instrumentation and Measurement Techniques, Prentice Hall, 1990.

[31] Sarhadi, M. : " Spectral Analysis at High Frequencies using a Modified FFT ", Proc. International Symposium on Computer Architecture and Digital Signal Processing, vol.1, pp. 242-246, Oct. 1989, Hong Kong.

[32] Sarhadi, M. and Aitchison, C.S. : " Sub-Nyquist Sampling Process in the Presence of Noisy Sampling Signal", Electronics Letters, vol. 16, no. 10, May 1980.

[33] Sarhadi, M. and Catchpole, J.L. : " Progress towards a Discrete Sub-Nyquist Fourier Transform", Proc. IEE Colloqium on Digital Signal Processing, University of Durham, 14th Sept., 1989.

[34] Baier, J. and Fürst, H.W. : " A Novel Method for Detection of Aliased Frequency Components in FFT-based Spectrum Analysers and Digital Oscilloscopes", Proc. IEEE International Symposium of Circuits and Systems, vol. 1, pp. 770-773, May 1993, Chicago, U.S.A.

[35] Lo, K.C. and Purvis, A. :" Fast Computational Algorithm in Parallel Random Sampling ", Electronics Letters, vol.28, no.12. June 1992, pp. 1115-1117.

[36] Lo, K.C. and Purvis, A. :" Parallel Random Sampling with Multiprocessor System", Proc. IEEE International Symposium on Circuits and Systems, vol. 3, pp. 1979-1982, May 1993, Chicago, U.S.A.

[37] Bilinsky, I.Y., Vystavkin, A.N. and Mikelson, A.K. : " Processing of Randomly-sampled Signals", Proc. 3rd European Signal Processing Conference, vol. 1, pp. 109-112, Sept. 1986, The Hague, The Netherlands.

[38] INMOS : Transputer Reference Manual, Prentice Hall, 1988.

[39] INMOS : 3L Parallel C User Guide, INMOS Ltd., 1989.

[40] Brigham, E.O. : The Fast Fourier Transform and Its Applications, Prentice Hall, 1988.

[41] Lo, K.C. and Purvis, A. : " Hybrid Additive Random Sampling and its Realization", Proc. IEEE International Symposium on Circuits and Systems, vol. 2, pp. 105-108, June, 1994, London, U.K.

[42] Lo, K.C. and Purvis, A. : " On Interlacing and Concatenating Additive Random Sampling Sequences ", Proc. International Conference on Signal Processing, vol. 1, pp. 48 - 51, Oct. 1993, Beijing, China.

[43] Therrien, C.W. : Discrete Random Signals and Statistical Signal Processing, Prentice Hall, 1992.

[44] Vaidyanathan, P.P. : Multirate Systems and Filter Banks, Prentice Hall, 1993.

[45] Press, W.H., Teukolsky, S.A., Vetterling,W.T. and Flannery B.P. : Numerical Recipes in C, Cambridge University Press, 1992.

[46] Lomb, N.R. : Astrophysics and Space Science, vol. 39, pp. 447-462, 1976.

[47] Hughes, R.D. and Heron, M.L. : " Approximate Fourier Transform using Square Waves", IEE Proc., vol. 136, pt A, no.4 , 1989, pp. 223-228.

[48] Bilinsky, I. and Mikelsons, A. : "Applications of Randomized and Irregular Sampling as an anti-aliasing technique", Proc., European Signal Processing Conference 1990, Barcelona.

[49] US Patent 5115189 : Anti-aliasing Dithering Method and Apparatus for Low-frequency Signal Sampling, Hewlett-Packard Company, Palo Alto 1992.

[50] Jain, J.R. and Jain, A.K. : "Displacement Measurement and Its Application in Interframe Image Coding", IEEE Trans. Comm. vol. COM-29, 1981, pp. 1799 - 1808.

[51] Rajala, S.A., Riddle, A.N. and Snyder, W.E. : "Application of the One-dimensional Fourier Transform for Tracking Moving Objects in Noisy Environment", Comput. Vis. Graph. Image Proc., vol. 21, 1983, pp. 280 - 293.

[52] Gonzalez, R.C. and Woods, R.E. : Digital Image Processing, Prentice Hall, 1992.

[53] Cowart, A.E., Snyder, W.E. and Ruedger, W.H. : "The Detection of Unresolved Targets Using the Hough Transform", Comput. Vis. Graph. Image Proc., vol. 21, 1983, pp. 222 - 238.

[54] Lo, K.C. and Purvis, A. : " Application of Random and Pseudo-random Sampling for Tracking Moving Objects", Proc. IEEE 1994 International Symposium on Speech, Image Processing and Neural Networks, vol. 1, pp. 260 - 263, April 1994, Hong Kong.

[55] Hamming, R.W. : Digital Filters, 2nd Edition, Prentice Hall, 1989.

[56] Antoniou, A. : Digital Filters; Analysis, Design, and Applications, McGraw Hill, 1993.

[57] Kohlenberg, A. : "Exact Interpolation of Band-limited Functions", J. Appl. Phys., vol. 47, no.12, Dec. 1953, pp. 1432-1436.

[58] Coulson, A.J. : "A Generalization of Nonuniform Bandpass Sampling", IEEE Trans. on Signal Processing, vol. 43, no. 3, March 1995, pp.694-704.

[59] Lamoureux, M.P. : "The Poorman's Transform : Approximating the Fourier Transform without Multiplication", IEEE Trans. on Signal Processing, vol. 41, no. 3, March 1993, pp. 1413-1415.

[60] Benedetto, J.J. :"Irregular Sampling and Frames", Wavelet Analysis and Its Applications, vol. 2, pp.445 - 505, Edited by C.K. Chui, Academic Press, 1992.

[61] Macovski, A. : Medical Imaging Systems, Prentice Hall, 1983.

[62] Cho, Z.H., Jones, J.P. and Singh, M. : Foundations of Medical Imaging, John Wiley, 1993.

[63] Angelidis, E. : "A Recursive Frequency-Sampling Method for Designing Zero-Phase FIR Filters by Nonuniform Samples", IEEE Trans. on Signal Processing, vol. 43, no. 6, June 1995, pp.1461-1467.

Publications :

[1] Lo, K.C., Siu W.C., Lun, DPK and Purvis, A. : " Address Generation of Prime Factor Algorithm in a Multiprocessor System", Proc. China 1991 International Conference on Circuits and Systems, vol 1, pp 133-136, June 1991, Shenzhen, China.

[2] Lo, K.C. and Purvis, A. : " Fast Computational Algorithm in Parallel Random Sampling ", Electronics Letters, vol.28, no.12. June 1992, pp. 1115-1117.

[3] Lo, K.C. and Purvis, A. : " Parallel Random Sampling with Multiprocessor System", Proc. IEEE International Symposium on Circuits and Systems, vol. 3, pp. 1979-1982, May 1993, Chicago, U.S.A.

[4] Lo, K.C. and Purvis, A. : " On Interlacing and Concatenating Additive Random Sampling Sequences ", Proc. International Conference on Signal Processing, vol. 1, pp. 48 - 51, Oct. 1993, Beijing, China.

[5] Lo, K.C. and Purvis, A. : " Application of Random and Pseudo-random Sampling for Tracking Moving Objects", Proc. IEEE 1994 International Symposium on Speech, Image Processing and Neural Networks, vol. 1, pp. 260 - 263, April 1994, Hong Kong.

[6] Lo, K.C. and Purvis, A. : " Hybrid Additive Random Sampling and its Realization", Proc. IEEE International Symposium on Circuits and Systems, vol. 2, pp. 105-108, June, 1994, London, U.K.

[7] Lo, K.C. and Purvis, A. : " Reconstructing Randomly Sampled Signals by the FFT ", Proc. 1996 IEEE International Symposium on Circuits and Systems, vol. 2, pp. 124-127, May, 1996, Atlanta, U.S.A.

[8] Lo, K.C. and Purvis, A. : " A New Approach to Estimate Spectra from Randomly Sampled Sequences ", under review.

Appendix 1: Program Listing

The following programs are written in 3L Parallel C for the realization of parallel a.r.s. using 5 transputers.

Configuration file :

```
processor host
processor root
processor T1
processor T2
processor T3
processor T4

wire jumper root[0] host[0]
wire ? root[2] T1[1]
wire ? T1[2] T2[1]
wire ? T2[2] T3[1]
wire ? T3[2] T4[1]
wire ? T3[3] root[3]

task master file = "getroott.b4" ins = 3 outs = 3
task s1 file = "get1t.b4" ins = 2 outs = 2
task s2 file = "get2t.b4" ins = 2 outs = 2
task s3 file = "get3t.b4" ins = 3 outs = 3
task s4 file = "get4t.b4" ins = 2 outs = 2
task iserver ins = 1 outs = 1
task filter ins = 2 outs = 2 data = 30k

place iserver host
place master root
place s1 T1
place s2 T2
place s3 T3
place s4 T4
place filter root

connect ? filter[0] iserver[0]
connect ? iserver[0] filter[0]

connect ? master[1] filter[1]
connect ? filter[1] master[1]

connect ? master[0] s1[0]
connect ? s1[0] master[0]

connect ? s1[1] s2[0]
connect ? s2[0] s1[1]

connect ? s2[1] s3[0]
connect ? s3[0] s2[1]

connect ? s3[1] s4[0]
connect ? s4[0] s3[1]

connect ? s3[2] master[2]
connect ? master[2] s3[2]
```

Root transputer

```
/* program name: getroott.c

  description: parallel ARS 32x32 points taking for 1 second under
               5 transputers nodes with channel method, include
               high order band

  related progarm: get1t.c
                   get2t.c
                   get3t.c
                   get4t.c
                   gett.cfg

  date : 11/3/1993    */


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <chan.h>

#define pi 3.141592
#define T 1.0
#define f 1.0
#define ESC 27
#define Y 89
#define y 121
#define n 110
#define node 5
#define node_address 0
#define MAXDATA 1024


/************************ Global variable **********************/

float **x;
float *t;
CHAN **in, **out;
int p, m, N;   /* p - row size
                  m - column size
                  N - p*m size of matrix
                */
int order;     /* high order band indicator */


/********************** Function protocol ***********************

 void error_message(int);
 void Initialize(void);
 struct MATRIX *MakeMatrix(void);
 int get_input(int *, float *);
 void get_time(void);
 void gen_sample(int *,float *,int);
 void Insmat(struct MATRIX *, int);
 void Calculate(struct MATRIX *, int, float *, float *);
 void KillMat(void);
 void Process(float *, float *, int, int);
 void output(float *, float *);
 void send_input(int *, float *, int, int);
 void send_matrix_size(int, int, int);
 void get_result(float *, float *, int);
 void main(int, char **, char **, CHAN **, int, CHAN **, int);
*****************************************************************/
```

```
/******************** Matrix Structure ************************/

struct MATRIX
{
 float **real;
 float **imag;
}*Mmat1, *Mmat2;


/******************** error message printout ******************/

void error_message(error_no)
int error_no;
{
 switch (error_no)
 {
  case 1: printf("error allocating memory\n");
          printf("*** program ended abnormally ***\n");
          exit(1);
          break;

  case 2: printf("error open file\n");
          printf("*** program ended abnormally ***\n");
          exit(1);
          break;

  case 3: printf("error reading file\n");
          printf("*** program ended abnormally ***\n");
          exit(1);
          break;

  default: printf("unexpected error\n");
          printf("*** program ended abnormally ***\n");
          exit(1);
          break;
 }
}

/************************* Get matrix size *********************/

void get_matrix_size()
{
 int temp, done, valid = 0;

 printf("Get matrix size parameter\n");
 printf("= = = = = = = = = = = = = = = = = = = = = = = = = = = = = =\n");
 while(!valid)
 {
  printf("Row size? ");
  scanf("%d",&p);
  if(p< =1)
   printf("Invalid input\n");
  else
  {
   done = 0;
   temp = p;
   while(temp1 && done! = 1)
   {
        if (temp%2! = 0) done = 1;
        temp = temp/2;
   }
  }
  if (done) printf("Invalid input\n");
  else valid = 1;
 }
```

```
  valid = 0;
  while(!valid)
  {
   printf("Column size? ");
   scanf("%d",&m);
   if(m < = 1)
    printf("Invalid input\n");
   else
   {
    done = 0;
    temp = m;
    while(temp1 && done! = 1)
    {
          if (temp%2! = 0) done = 1;
          temp = temp/2;
    }
   }
   if (done) printf("Invalid input\n");
   else valid = 1;
  }
  N = p*m;
}

/************************ Memory allocation ********************/

void Initialize()
{
 int i;

 printf("Initialization memory\n");
 printf("= = = = = = = = = = = = = = = = = = = = = = = = = \n");
 x = (float **)malloc(p*sizeof(float *));
 if (x = = NULL) error_message(1);
 for(i = 0;i < p;i + + )
 {
  x[i] = (float *)malloc(m*sizeof(float));
  if (x[i] = = NULL) error_message(1);
 }
 t = malloc(p*sizeof(float));
 if (t = = NULL) error_message(1);
}


/************************ Create Matrix ************************/

struct MATRIX *MakeMatrix()
{
 struct MATRIX *mat;
 int i;

 mat = (struct MATRIX *)malloc(sizeof(struct MATRIX));
 if (mat = = NULL) error_message(1);
 mat- > real = (float **)malloc(p*sizeof(float *));
 if (mat- > real = = NULL) error_message(1);
 mat- > imag = (float **)malloc(p*sizeof(float *));
 if (mat- > imag = = NULL) error_message(1);
 for (i = 0;i < p;i + + )
 {
  mat- > real[i] = (float *)malloc(m*sizeof(float));
  if (mat- > real[i] = = NULL) error_message(1);
  mat- > imag[i] = (float *)malloc(m*sizeof(float));
  if (mat- > imag[i] = = NULL) error_message(1);
 }
 return (mat);
}
```

```
/********************** Get frequency & amplitude ***************/

int get_input(freq,amp)
int *freq;
float *amp;
{
 int count = 0, end_input = 0;
 char input;

 printf("Get input frequency and amplitude\n");
 printf("======================================\n");
 while(end_input! = 1)
 {
  printf("frequency ");
  scanf("%d",freq + +);
  printf("amplitude ");
  scanf("%f",amp + +);
  count + +;
  printf("More (y/n) ");
  scanf("%s",&input);
  if (input = = n) end_input = 1;
  else
   end_input = 0;
 }
 return(count);
}

/*********************** Get variable timing ******************/

void get_time()
{
 FILE *fp;
 int i;
 float input;

 printf("Getting variable timing\n");
 printf("==========================\n");
 fp = fopen("uniform.prn","r");
 if (fp = = NULL) error_message(2);
 t[0] = 0.0;
 i = 1;
 while(i < p)
 {
  if(fscanf(fp,"%f",&input) = = EOF) error_message(3);
  t[i] = (float) (input + 1)/N;
  i + +;
 }
 fclose(fp);
}

/*************** Generate sample matrix element ***************/

void gen_sample(freq,amp,count)
int *freq, count;
float *amp;
{
 int i, j, row, column;
 float tm, comp_value, angle;

 printf("Generating sampling sequence\n");
 printf("================================\n");
 tm = 0.0;
```

```
  for (row=0;row<p;row++)
  for (column=0;column<m;column++)
  {
   if (column==0)
   {
    tm+=t[row];
    t[row]=tm;
   }
   else tm+=(float) 1.0/N;
   comp_value=0.0;
   for (i=0;i<count;i++)
   {
    angle=2.0*pi*freq[i]*tm;
    comp_value=amp[i]*cos(angle)+comp_value;
   }
   x[row][column]=comp_value;
  }
}

/******************** Insert elements to matrix ****************/

void Insmat(mat,i)
int i;
struct MATRIX *mat;
{
 int r, c;
 float angle, cosine, sine;

 if (i==0)
 {
  for(c=0;c<m;c++)
  for(r=0;r<p;r++)
  {
   mat->real[r][c]=x[r][c];
   mat->imag[r][c]=0.0;
  }
 }
 else
 {
  for (c=0;c<m;c++)
  {
   angle=2.0*pi*c*i/N;
   cosine=cos(angle);
   sine=sin(angle);
   for (r=0;r<p;r++)
   {
        mat->real[r][c] = x[r][c]*cosine;
        mat->imag[r][c] = x[r][c]*sine;
   }
  }
 }
}

/****************** Calculate frequency spectrum ***************/

void Calculate(mat,comp,real,imag)
struct MATRIX *mat;
int comp;
float *real, *imag;
{
 float x_real, x_imag, row_sum_real, row_sum_imag;
 float angle, cosine, sine;
 int r, c, i, component;
```

```
  for(i=0;i<order;i++)
  {
   component=comp+i*N;
   x_real = 0.0;
   x_imag = 0.0;
   for (r=0;r<p;r++)
   {
    row_sum_real = mat->real[r][0];
    row_sum_imag = 0.0;
    for (c=1;c<m;c++)
    {
          row_sum_real += mat->real[r][c];
          row_sum_imag += mat->imag[r][c];
    }
    if (r==0)
    {
        x_real=row_sum_real;
        x_imag=row_sum_imag;
    }
    else
    {
        angle = 2.0*pi*component*t[r];
        cosine=cos(angle);
        sine=sin(angle);
        x_real += (row_sum_real*cosine-row_sum_imag*sine);
        x_imag += (row_sum_imag*cosine+row_sum_real*sine);
    }
   }
   real[component] = x_real;
   imag[component] = x_imag;
  }
}

/********************** Free memory allocation *****************/

void KillMat()
{
 int i;

 for (i=0;i<p;i++)
 {
  free(Mmat1->real[i]);
  free(Mmat1->imag[i]);
  free(Mmat2->real[i]);
  free(Mmat2->imag[i]);
  free(x[i]);
 }
 free(Mmat1->real);
 free(Mmat1->imag);
 free(Mmat2->real);
 free(Mmat2->imag);
 free(x);
 free(Mmat1);
 free(Mmat2);
 free(t);
}

/************************ Process matrix ************************/

void Process(real,imag,start,stop)
float *real, *imag;
int start, stop;
```

```
{
 int row, column, i, j, comp;
 float power_index1, power_index2;

 printf("Processing data\n");
 printf("=================\n");
 /* calculate M[0] */

 Mmat1 = MakeMatrix();
 if (Mmat1==NULL)
  error_message(1);

 Mmat2 = MakeMatrix();
 if (Mmat2==NULL)
  error_message(1);

 comp = 0;
 Insmat(Mmat1,comp);
 Calculate(Mmat1,comp,real,imag);

 /* calculate M[N/2] */

 comp = N/2;
 for (column=0;column<m;column+=2)
 for (row=0;row<p;row++)
 {
        Mmat2->real[row][column] = Mmat1->real[row][column];
        Mmat2->imag[row][column] = Mmat1->imag[row][column];
        Mmat2->real[row][column+1] = -1.0 * (Mmat1->real[row][column+1]);
        Mmat2->imag[row][column+1] = -1.0 * (Mmat1->imag[row][column+1]);
 }
 Calculate(Mmat2,comp,real,imag);

 /* calculate M[N/4] */

 comp = N/4;
 for (row=0;row<p;row++)
 {
  Mmat2->real[row][0]=Mmat1->real[row][0];
  Mmat2->imag[row][0]=Mmat1->imag[row][0];
 }
 for (column=1;column<m;column+=2)
 {
  power_index1=pow(-1.0,(float)((column+1)/2));
  power_index2=pow(-1.0,(float)((column+3)/2));
  for (row=0;row<p;row++)
  {
   Mmat2->real[row][column+1] = Mmat1->real[row][column+1]*power_index1;
   Mmat2->imag[row][column+1] = Mmat1->imag[row][column+1]*power_index1*-1.0;
   Mmat2->real[row][column] = Mmat1->imag[row][column]*power_index2;
   Mmat2->imag[row][column] = Mmat1->real[row][column]*power_index2;
  }
 }
 Calculate(Mmat2,comp,real,imag);

 /* calculate M[3N/4]   */

 comp = 3*N/4;
 for (column=0;column<m;column++)
 for (row=0;row<p;row++)
        Mmat2->imag[row][column] = Mmat2->imag[row][column] * -1.0;
 Calculate(Mmat2,comp,real,imag);

 /*   for start to stop calculate Mi   */
```

```
for (i=start;i<stop;i++)
{
        comp = i;
        Insmat(Mmat1,comp);
 Calculate(Mmat1,comp,real,imag);

 /* Mat[N/2+i] */
 comp = N/2 +i;
 for (column=0;column<m;column+=2)
 for (row=0;row<p;row++)
 {
         Mmat2->real[row][column] = Mmat1->real[row][column];
         Mmat2->imag[row][column] = Mmat1->imag[row][column];
         Mmat2->real[row][column+1] = Mmat1->real[row][column+1] * -1.0;
         Mmat2->imag[row][column+1] = Mmat1->imag[row][column+1] * -1.0;
 }
 Calculate(Mmat2,comp,real,imag);

 /* Mat[N-i] */
 comp = N-i;
 for (column=0;column<m;column++)
 for (row=0;row<p;row++)
 {
  Mmat2->real[row][column] = Mmat1->real[row][column];
  Mmat2-i>mag[row][column] = Mmat1->imag[row][column] * -1.0;
 }
 Calculate(Mmat2,comp,real,imag);

 /* Mat[N/2-i] */
 comp = N/2-i;
 for (column=0;column<m;column+=2)
 for (row=0;row<p;row++)
 {
         Mmat2->real[row][column+1] = Mmat1->real[row][column+1] * -1.0;
         Mmat2->imag[row][column+1] = Mmat1->imag[row][column+1];
 }
 Calculate(Mmat2,comp,real,imag);

 /* calculate N/4 - i   */

 j = N/4 - i;
 if (j!=i)
 {
  /* Mmat[j] */
  comp = j;
  for (row=0;row<p;row++)
  {
       Mmat2->real[row][0]=Mmat1->real[row][0];
       Mmat2->imag[row][0]=Mmat1->imag[row][0];
  }
  for (column=1;column<m;column+=2)
  {
       power_index1=pow(-1.0,(float)((column+1)/2));
       power_index2=pow(-1.0,(float)((column+3)/2));
       for (row=0;row<p;row++)
       {
        Mmat2>-real[row][column+1] = Mmat1->real[row][column+1]*power_index1;
        Mmat2->imag[row][column+1] = Mmat1->imag[row][column+1]*power_index1*-1.0;
        Mmat2->real[row][column] = Mmat1->imag[row][column]*power_index2;
        Mmat2-imag[row][column] = Mmat1-real[row][column]*power_index2;
       }
 }
 Calculate(Mmat2,comp,real,imag);

 /* Mat[N/2+j] */
```

```
    comp = N/2 + j;
    for (column=0;column<m;column+=2)
    for (row=0;row<p;row++)
    {
        Mmat1->real[row][column] = Mmat2->real[row][column];
        Mmat1->imag[row][column] = Mmat2->imag[row][column];
        Mmat1->real[row][column+1] = Mmat2->real[row][column+1] * -1.0;
        Mmat1->imag[row][column+1] = Mmat2->imag[row][column+1] * -1.0;
    }
    Calculate(Mmat1,comp,real,imag);
    /* Mmat[N-j] */
    comp = N-j;
    for (column=0;column<m;column++)
    for (row=0;row<p;row++)
    {
        Mmat1->real[row][column] = Mmat2->real[row][column];
        Mmat1->imag[row][column] = Mmat2->imag[row][column] * -1.0;
    }
    Calculate(Mmat1,comp,real,imag);

    /* Mat[N/2-j] */
    comp = N/2 -j;
    for (column=0;column<m;column+=2)
    for (row=0;row<p;row++)
    {
        Mmat1->real[row][column+1] = Mmat2->real[row][column+1] * -1.0;
        Mmat1->imag[row][column+1] = Mmat2->imag[row][column+1];
    }
    Calculate(Mmat1,comp,real,imag);
  }
 }
}


/************************ Output to file ***********************/

void output(real,imag)
float *real, *imag;
{
 FILE *fp;
 int i;

 printf("Output to file\n");
 printf("= = = = = = = = = = = = = = =\n");
 fp = fopen("gett.prn","w");
 if (fp==NULL) error_message(2);
 else
 real[0] = real[0]/N;
 fprintf(fp,"%.10f\t%.10f\n",real[0].imag[0]);
 for (i=1;i<order*N;i++)
 {
  real[i] = real[i]/N*2;
  imag[i] = imag[i]/N*2;
  fprintf(fp,"%.10f\t%.10f\n",real[i],imag[i]);
 }
 fclose(fp);
}


/********* Send message information to other processors *********/

void send_input(freq,amp,freq_count,port)
int *freq, freq_count, port;
float *amp;
{
```

```
 /* chan out message to port */
 chan_out_word(freq_count,out[port]);
 chan_out_message(freq_count*sizeof(int),freq,out[port]);
 chan_out_message(freq_count*sizeof(float),amp,out[port]);
 chan_out_message(p*sizeof(float),t,out[port]);
}


/********** Send matrix size to other processors **************/

void send_matrix_size(port)
int port;
{
 /* chan out message to port */
 chan_out_word(p,out[port]);
 chan_out_word(m,out[port]);
 chan_out_word(order,out[port]);
}


/************** Get results from other processors ***************/

void get_result(real,imag,port)
float *real, *imag;
int port;
{
 int count, cnt[MAXDATA], i, comp;
 float inp_real[2*MAXDATA], inp_imag[2*MAXDATA];

 /* chan in message from port */
 chan_in_word(&count,in[port]);
 chan_in_message(count*sizeof(float),inp_real,in[port]);
 chan_in_message(count*sizeof(float),inp_imag,in[port]);
 chan_in_message(count*sizeof(int),cnt,in[port]);
 for (i=0;i<count;i++)
 {
  comp=cnt[i];
  real[comp]=inp_real[i];
  imag[comp]=inp_imag[i];
 }
}


/************************* Main program ************************/

void main(argc,argv,envp,in_ports,inlen,out_ports,outlen)
int argc,inlen,outlen;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 int freq[MAXDATA], freq_count, count[MAXDATA];
 float amp[MAXDATA];
 int start, stop;
 float y_real[2*MAXDATA], y_imag[2*MAXDATA], real[MAXDATA];
 int tm1, tm2, tn1, tn2;
 int size_int, size_remain;

 if(argc) order=1;
 else
 switch(*argv[1])
 {
  case 'H':  order=2;
             break;
  case 'h':  order=2;
             break;
```

```
  case 'L':  order = 1;
             break;
  case 'l':  order = 1;
             break;
  default:   order = 1;
             break;
}

in = in_ports;
out = out_ports;

get_matrix_size();

Initialize();
send_matrix_size(0);

/* get input frequency component & amplitude */
freq_count = get_input(freq,amp);

/* get random variable */
get_time();

/* send message to workers */
printf("sending packets to workers\n");
printf("= = = = = = = = = = = = = = = = = = = = = = = = = = = = = =\n");

send_input(freq,amp,freq_count,0);

/* generate sampling sequence */
gen_sample(freq,amp,freq_count);

/* generate no. of processing element */
size_int = N/8/node;
size_remain = N/8%node;
switch(node_address) {
  case 0: start = 1;
          stop = start + size_int;
          break;
  case 1: start = size_int + 1;
          stop = start + size_int;
          break;
  case 2: start = 2*size_int + 1;
          stop = start + size_int;
          break;
  case 3: start = 3*size_int + 1;
          stop = start + size_int;
          break;
  case 4: start = 4*size_int + 1;
          stop = start + size_int + size_remain;
          break;
  default: exit(1);
}

tm1 = time(NULL);
tn1 = timer_now();

/* Processing Data */
Process(y_real,y_imag,start,stop);

/* get result from workers */
printf("get result from workers\n");
printf("= = = = = = = = = = = = = = = = = = = = = = = = = =\n");

get_result(y_real,y_imag,0);
get_result(y_real,y_imag,2);
```

```
  tm2=time(NULL);
  tn2=timer_now();

  printf("time needed = %d\n",tm2-tm1);
  printf("tick time = %d\n",tn2-tn1);

  output(y_real,y_imag);

  /* free memory from matrix allocation */
  KillMat();

  printf("program ended normally\n");
  printf("========================\n");
}
```

Node transputer

Since the programs for all nodes are the same execpt the node addresses, only the programs for node 1 are listed below.

```
/************************************************************************
  program name: get1t.c

  description: parallel ARS 32x32 points taking for 1 second under
               5 transputers nodes with channel method, include
               high order band

  related progarm: getroott.c
                   get2t.c
                   get3t.c
                   get4t.c
                   gett.cfg

************************************************************************/

#include <stdlib.h>
#include <math.h>
#include <chan.h>

#define pi 3.141592
#define T 1.0
#define f 1.0
#define node 5
#define node_address 1
#define MAXDATA 1024

/************************ Global variable *********************/

float **x;
float *t;
CHAN **in, **out;
int p, m, N;   /* p - row size
                  m - column size
                  N - p*m size of matrix
                */
int order;     /* high order band indicator */


/********************** Function protocol **********************
 void error_message(int);
 void Initialize(void);
 struct MATRIX *MakeMatrix(void);
```

```
 int get_input(int *, float *);
 void gen_sample(int *,float *,int);
 void Insmat(struct MATRIX *, int);
 int Calculate(struct MATRIX *, int, float *, float *,float *,int);
 void KillMat(void);
 int Process(float *, float *, int, int);
 void send_input(int *, float *, int, int);
 void send_matrix_size(int, int, int);
 void send_result(float *, float *, int *, int, int);
 int get_result(float *, float *, int);
 void main(int, char **, char **, CHAN **, int, CHAN **, int);


/********************* Matrix Structure ************************/

struct MATRIX
{
 float **real;
 float **imag;
}*Mmat1, *Mmat2;


/************************* Memory allocation ********************/

void Initialize()
{
 int i;

 x=(float **)malloc(p*sizeof(float *));
 for(i=0;i<p;i++)
  x[i]=(float *)malloc(m*sizeof(float));
 t=malloc(p*sizeof(float));
}


/************************** Create Matrix ************************/

struct MATRIX *MakeMatrix()
{
 struct MATRIX *mat;
 int i;

 mat=(struct MATRIX *)malloc(sizeof(struct MATRIX));
 mat->real=(float **)malloc(p*sizeof(float *));
 mat->imag=(float **)malloc(p*sizeof(float *));
 for (i=0;i<p;i++)
 {
  mat->real[i]=(float *)malloc(m*sizeof(float));
  mat->imag[i]=(float *)malloc(m*sizeof(float));
 }
 return (mat);
}


/* Get input frequency component & amplitude from other processors */

int get_input(freq,amp,port)
int *freq, port;
float *amp;
{
 int count;
 chan_in_word(&count,in[port]);
 chan_in_message(count*sizeof(int),freq,in[port]);
 chan_in_message(count*sizeof(float),amp,in[port]); chan_in_message(p*sizeof(float),t,in[port]);
```

```
 return(count);
}

/*************** Generate sample matrix element ***************/

void gen_sample(freq,amp,count)
int *freq, count;
float *amp;
{
 int i, j, row, column;
 float tm, comp_value, angle;

 tm = 0.0;
 for (row = 0;row < p;row + +)
 for (column = 0;column < m;column + +)
 {
  if (column = = 0)
  {
   tm + = t[row];
   t[row] = tm;
  }
  else tm + = (float) 1.0/N;
  comp_value = 0.0;
  for (i = 0;i < count;i + +)
  {
   angle = 2.0*pi*freq[i]*tm;
   comp_value = amp[i]*cos(angle) + comp_value;
  }
  x[row][column] = comp_value;
 }
}

/******************** Insert elements to matrix ****************/

void Insmat(mat,i)
int i;
struct MATRIX *mat;
{
 int r, c;
 float angle, cosine, sine;

 if (i = = 0)
 {
  for(c = 0;c < m;c + +)
  for(r = 0;r < p;r + +)
  {
   mat- > real[r][c] = x[r][c];
   mat- > imag[r][c] = 0.0;
  }
 }
 else
 {   for (c = 0;c < m;c + +)
  {
   angle = 2.0*pi*c*i/N;
   cosine = cos(angle);
   sine = sin(angle);
   for (r = 0;r < p;r + +)
   {
        mat- > real[r][c] = x[r][c]*cosine;
        mat- > imag[r][c] = x[r][c]*sine;
   }
  }
 }
}
```

```
/****************** Calculate frequency spectrum ***************/

Calculate(mat,comp,real,imag,cnt,element)
struct MATRIX *mat;
int comp, *cnt, element;
float *real, *imag;
{
 float x_real, x_imag, row_sum_real, row_sum_imag;
 float angle, cosine, sine;
 int r, c, i, key, component;

 key=element;
 for(i=0;i<order;i++)
 {
  component=comp+i*N;
  x_real = 0.0;
  x_imag = 0.0;
  for (r=0;r<p;r++)
  {
   row_sum_real = mat->real[r][0];
   row_sum_imag = 0.0;
   for (c=1;c<m;c++)
   {
        row_sum_real += mat->real[r][c];
        row_sum_imag += mat->imag[r][c];
   }
   if (r==0)
   {
        x_real=row_sum_real;
        x_imag=row_sum_imag;
   }
   else
   {
        angle = 2.0*pi*component*t[r];
        cosine=cos(angle);
        sine=sin(angle);
        x_real += (row_sum_real*cosine-row_sum_imag*sine);
        x_imag += (row_sum_imag*cosine+row_sum_real*sine);
   }
  }  real[key] = x_real;
  imag[key] = x_imag;
  cnt[key]=component;
  key++;
 }
 return(key);
}

/********************** Free memory allocation *****************/

void KillMat()
{
 int i;

 for (i=0;i<p;i++)
 {
  free(Mmat1->real[i]);
  free(Mmat1->imag[i]);
  free(Mmat2->real[i]);
  free(Mmat2->imag[i]);
  free(x[i]);
 }
 free(Mmat1->real);
 free(Mmat1->imag);
 free(Mmat2->real);
 free(Mmat2->imag);
```

```
 free(x);
 free(Mmat1);
 free(Mmat2);
 free(t);
}

/************************ Process matrix ************************/

int Process(real,imag,start,stop,count)
float *real, *imag;
int start, stop, *count;
{
 int row, column, i, j, k, comp;
 float power_index1, power_index2;

 /* for start to stop calculate Mi */

 Mmat1 = MakeMatrix();
 Mmat2 = MakeMatrix();
 k = 0;
 for (i = start;i < stop;i + +)
 {
  comp = i;
  Insmat(Mmat1,comp);
  k = Calculate(Mmat1,comp,real,imag,count,k);

  /* Mat[N/2 + i] */  comp = N/2 + i;
  for (column = 0;column < m;column + = 2)
  for (row = 0;row < p;row + +)
  {
   Mmat2- > real[row][column] = Mmat1- > real[row][column];
   Mmat2- > imag[row][column] = Mmat1- > imag[row][column];
   Mmat2- > real[row][column + 1] = Mmat1- > real[row][column + 1] * -1.0;
   Mmat2- > imag[row][column + 1] = Mmat1- > imag[row][column + 1] * -1.0;
  }
  k = Calculate(Mmat2,comp,real,imag,count,k);

  /* Mat[N-i] */
  comp = N-i;
  for (column = 0;column < m;column + +)
  for (row = 0;row < p;row + +)
  {
   Mmat2- > real[row][column] = Mmat1- > real[row][column];
   Mmat2- > imag[row][column] = Mmat1- > imag[row][column] * -1.0;
  }
  k = Calculate(Mmat2,comp,real,imag,count,k);

  /* Mat[N/2-i] */
  comp = N/2-i;
  for (column = 0;column < m;column + = 2)
  for (row = 0;row < p;row + +)
  {
   Mmat2- > real[row][column + 1] = Mmat1- > real[row][column + 1] * -1.0;
   Mmat2- > imag[row][column + 1] = Mmat1- > imag[row][column + 1];
  }
  k = Calculate(Mmat2,comp,real,imag,count,k);

  /* calculate N/4 - i  */
  j = N/4 - i;
  if (j! = i)
  {
   /* Mmat[j] */
   comp = j;
   for (row = 0;row < p;row + +)
   {
```

```
            Mmat2->real[row][0] = Mmat1->real[row][0];
            Mmat2->imag[row][0] = Mmat1->imag[row][0];
        }
        for (column = 1;column < m;column + = 2)
        {
            power_index1 = pow(-1.0,(float)((column + 1)/2));
            power_index2 = pow(-1.0,(float)((column + 3)/2));
            for (row = 0;row < p;row + + )
            {
             Mmat2->real[row][column + 1] = Mmat1->real[row][column + 1]*power_index1;
             Mmat2->imag[row][column + 1] = Mmat1->imag[row][column + 1]*power_index1*-1.0;
             Mmat2->real[row][column] = Mmat1->imag[row][column]*power_index2;
             Mmat2->imag[row][column] = Mmat1->real[row][column]*power_index2;
            }
        }   k = Calculate(Mmat2,comp,real,imag,count,k);

        /* Mat[N/2 + j] */
        comp = N/2 + j;
        for (column = 0;column < m;column + = 2)
        for (row = 0;row < p;row + + )
        {
            Mmat1->real[row][column] = Mmat2->real[row][column];
            Mmat1->imag[row][column] = Mmat2->imag[row][column];
            Mmat1->real[row][column + 1] = Mmat2->real[row][column + 1] * -1.0;
            Mmat1->imag[row][column + 1] = Mmat2->imag[row][column + 1] * -1.0;
        }
        k = Calculate(Mmat1,comp,real,imag,count,k);

        /* Mmat[N-j] */
        comp = N-j;
        for (column = 0;column < m;column + + )
        for (row = 0;row < p;row + + )
        {
            Mmat1->real[row][column] = Mmat2->real[row][column];
            Mmat1->imag[row][column] = Mmat2->imag[row][column] * -1.0;
        }
        k = Calculate(Mmat1,comp,real,imag,count,k);

        /* Mat[N/2-j] */
        comp = N/2 -j;
        for (column = 0;column < m;column + = 2)
        for (row = 0;row < p;row + + )
        {
            Mmat1->real[row][column + 1] = Mmat2->real[row][column + 1] * -1.0;
            Mmat1->imag[row][column + 1] = Mmat2->imag[row][column + 1];
        }
        k = Calculate(Mmat1,comp,real,imag,count,k);
      }
     }
     return(k);
}


/************ Get matrix size from other processors ***********/

void get_matrix_size(port)
int port;
{
 /* chan in message from port */
 chan_in_word(&p,in[port]);
 chan_in_word(&m,in[port]);
 chan_in_word(&order,in[port]);
 N = p*m;
}
```

```
/******** Send message information to other processors ********/

void send_input(freq,amp,freq_count,port)int *freq, freq_count, port;
float *amp;
{
 chan_out_word(freq_count,out[port]);
 chan_out_message(freq_count*sizeof(int),freq,out[port]);
 chan_out_message(freq_count*sizeof(float),amp,out[port]);
 chan_out_message(p*sizeof(float),t,out[port]);
}

/********** Send matrix size to other processors **************/

void send_matrix_size(port)
int port;
{
 /* chan out message to port */
 chan_out_word(p,out[port]);
 chan_out_word(m,out[port]);
 chan_out_word(order,out[port]);
}


/************* Send result to other processors *****************/

void send_result(real,imag,count,comp_count,port)
float *real, *imag;
int *count, comp_count, port;
{
 chan_out_word(comp_count,out[port]);
 chan_out_message(comp_count*sizeof(float),real,out[port]);
 chan_out_message(comp_count*sizeof(float),imag,out[port]);
 chan_out_message(comp_count*sizeof(int),count,out[port]);
}

/***************** Get result from other processor **************/

int get_result(real,imag,count,comp_count,port)
float *real, *imag;
int comp_count, port, *count;
{
 int inp_cnt[MAXDATA], i, comp, inp_count;
 float inp_real[MAXDATA], inp_imag[MAXDATA];

 /* chan in message from inport 1 */
 chan_in_word(&inp_count,in[port]);
 chan_in_message(inp_count*sizeof(float),inp_real,in[port]);
 chan_in_message(inp_count*sizeof(float),inp_imag,in[port]);
 chan_in_message(inp_count*sizeof(int),inp_cnt,in[port]);
 for (i=0;i<inp_count;i++)
 {
  real[i+comp_count]=inp_real[i];
  imag[i+comp_count]=inp_imag[i];
  count[i+comp_count]=inp_cnt[i];
 } return(inp_count+comp_count);
}

/********************** main program ***************************/

void main(argc,argv,envp,in_ports,inlen,out_ports,outlen)
int argc,inlen,outlen;
char *argv[],*envp[];
CHAN *in_ports[],*out_ports[];
{
 int freq[MAXDATA], freq_count, count[2*MAXDATA], comp_count;
```

```
float amp[MAXDATA];
int start, stop;
float y_real[2*MAXDATA], y_imag[2*MAXDATA];
int size_int, size_remain;

in = in_ports;
out = out_ports;

get_matrix_size(0);

send_matrix_size(1);

Initialize();

/* get frequency component & amplitude */
freq_count = get_input(freq,amp,0);
/* send message to workers */
send_input(freq,amp,freq_count,1);
/* generate sampling sequence */
gen_sample(freq,amp,freq_count);

/* generate no. of processing element */
size_int = N/8/node;
size_remain = N/8%node;

switch(node_address) {
 case 0: start = 1;
         stop = start + size_int;
         break;
 case 1: start = size_int + 1;
         stop = start + size_int;
         break;
 case 2: start = 2*size_int + 1;
         stop = start + size_int;
         break;
 case 3: start = 3*size_int + 1;
         stop = start + size_int;
         break;
 case 4: start = 4*size_int + 1;
         stop = start + size_int + size_remain;
         break;  default: start = 0;
         stop = 0;
         break;
}

 /* Processing Data */
comp_count = Process(y_real,y_imag,start,stop,count);

/* get result from worker */
comp_count = get_result(y_real,y_imag,count,comp_count,1);

/* send to root */
send_result(y_real,y_imag,count,comp_count,0);

 KillMat();
}
```

Appendix 2 : Program Listing

C routines for calculating the DFT according to hybrid a.r.s. and generating random numbers.

```
/************************************** */
/*                                      */
/*          Hyrbid.c                    */
/*                                      */
/************************************** */

#include <math.h>
#include <conio.h>
#include <malloc.h>
#include <stdlib.h>
#include <complex.h>

long int Number_Add=0, Number_Mult=0;
static complex w;
static long int i,j,N;

/* a = pointer, point to input data                */
/* *(a) = starting address of input data           */
/* r = pointer, point to "Random Number" table     */
/* *(r) = starting address of table                */
/* No of points = 2 ^ m                            */
/*                                                 */
/* After transformation, output data is stored in  */
/* the same address.                               */


rft(complex huge *a, int m, float huge *r)
{
float randomj;
complex huge *b;
double sign;
double Cr,Ci, PI;

N = (long)pow(2.0,(double)m);
PI = 4.0*atan(1.0);

/* Get the memory for temporary storage */
b = (complex huge *) _fcalloc ((N+1),sizeof(complex));
if (!b) { printf ("\nNot enough memory for temporary buffer");
          exit(1);}

/* evaluating the DFT */
   for ( i = 0 ; i < N ; i++ )
    {     *(b+i) = complex(0,0); /* Initialization */
         for ( j = 1 ; j < N/4 ; j++ )
          { randomj = (float)j + *(r+j);
            if ( (i%2) == 0 ) sign = 1.0; else sign = -1.0;
            Cr = real((*a + sign * *(a+N/2)) +
                    (*(a+N/4) + *(a+(3*N/4))) * cos(2*PI*i/4));
            Ci = real((*(a+N/4) - *(a+(3*N/4))) * -sin(2*PI*i/4));
            *(b+i) += complex(real(*(a+j) + *(a+N-j) + sign * (*(a+N/2-j) +
                              *(a+N/2+j))) * cos(2*PI*i*randomj/N),
                          real(*(a+j) - *(a+N-j) + sign * (*(a+N/2+j) -
                            *(a+N/2-j))) * -sin(2*PI*i*randomj/N));
            Number_Add += (4+6);
            Number_Mult += (4+3);
            }
         *(b+i) += complex(Cr,Ci);
```

```
            Number_Add + = 1;
            }
 /* Transfer Data */
 for ( i = 0 ; i < N ; i+ + )
 *(a+i) = *(b+i);

 free (b);
 return 0;
 }

 /* Random number generator        */

 Gen_Random(float  huge *Random_Number, int length)
 { int i;
  randomize();

  for ( i = 1 ; i< length/4 ; i+ + )
   { *(Random_Number+i) = (float)(random(length) - (length > > 1))/length;
     *(Random_Number+length/2-i) = -*(Random_Number+i);
     *(Random_Number+length/2+i) =  *(Random_Number+i);
     *(Random_Number+length -i) = -*(Random_Number+i);
   }
  for ( i = 0 ; i< length ; i+ = length/4 )
    *(Random_Number + i) = 0;

 return 0;
 }
```

Remarks :

(i) The header < complex.h > contains a type definition :

```
typedef struct {
     float real, imag;
} complex;
```

(ii) In the second routine, Gen_Random(), randomize() and random() are functions of Borland C.

Appendix 3 : Program Listing

C routines for computing circular auto-correlation

```
/*********************/
/*                   */
/*      CCOR.C       */
/*                   */
/*********************/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

/* Inputs : timing file, data file, step size and % tolerance */

float match (int k, float step, double frac);
float *tim, *input, *dtime, *dinput, *output, *temp;
int len, N;

main (void)
{  int i,j,ch,divn;
   float st;
   double t,fr;
```

```
    char fname1[20], fname2[20];
    FILE *ti, *in, *out;
    tim = malloc(1024*sizeof(float));
    if (!tim) { printf ("\n Cannot allocate tim \n");
                exit (1);
              }
    input = malloc(1024*sizeof(float));
    if (!input) { printf ("\n Cannot allocate input \n");
                exit (1);
              }
    dtime = malloc(2048*sizeof(float));
    if (!dtime) { printf ("\n Cannot allocate dtime \n");
                exit (1);
              }
    dinput = malloc(2048*sizeof(float));
    if (!dinput) { printf ("\n Cannot allocate dinput \n");
                exit (1);
              }
    output = malloc(4096*sizeof(float));
    if (!output) { printf ("\n Cannot allocate output \n");
                exit (1);
              }
    temp = malloc(1024*sizeof(float));
    if (!temp) { printf ("\n Cannot allocate temp in sub.\n");
                exit (1);
              }

    printf("\n input timing file = ");
    gets(fname1);
     if ((ti= fopen(fname1, "r"))==NULL)
    {  printf ("\n Cannot open %s \n", fname1);
          exit(1);
    }
    for (i=0; ; i++)
    {  fscanf(ti, "%f", &tim[i]);
          if (feof(ti) != 0)  break;
    }
    fclose (ti);
    len = i;
    printf("\n  length = %d\n", len);

    printf ("\n input data file = ");
    gets(fname2);
    if ((in= fopen(fname2, "r"))==NULL)
    {  printf ("\n Cannot open %s \n", fname2);
          exit(1);
    }
    for (i=0; ; i++)
    {  fscanf(in, "%f", &input[i]);
          if (feof(ti) != 0)  break;
    }
    fclose (in);

/* repeat the input sequences */
    for (i=0; i<len; i++)      dtime[i] = tim[i];
    for (i=len; i<2*len; i++)  dtime[i] = 1.0+tim[i-len];
    for (i=0; i<len; i++)      dinput[i] = input[i];
    for (i=len; i<2*len; i++)  dinput[i] = input[i-len];
/*  for (i=0; i<2*len; i++)
          printf(" dinput[%d] = %f\n", i, dinput[i]);   */

/* correlate input and dinput with STEP size  */
    printf("\n Input division = ");
    scanf("%d",&divn);
    printf("\n ");
```

```
    printf(" Input tolerance as a percentage of step size = ");
    scanf("%lf",&t);
    printf("\n");

    N = divn*len;
    st = 1.0/(float)N;
    fr = (double)st*t/200.0;
    printf(" divn = %d, N = %d , st = %f, fr = +- %lf\n",divn,N,st,fr);

    output[0] = 0.0;
    for (i=0;i<len; i++)
          output[0] = output[0] + input[i]*input[i];
    for (i=1; i<N; i++)
          output[i] = match(i, st, fr);
    printf("\n");
    for (i=0; i< N; i+ = N-2)
          printf(" output[%d] = %f output[%d] = %f \n", \
            i,output[i],i+1,output[i+1]);

    if ((out = fopen ("output.prn", "w")) = =NULL)
          {  printf(" Cannot open output.prn for writing.\n");
            exit(1);
          }
    for (i=0; i<N; i++)
          fprintf (out, "%f\n", output[i]);
}


/*********************************/
float match (int k, float step, double tol)
{  float result, dt;
   double apart, diff ;
   int i,j,ch, back;

   printf(" %d\r",k);
   for (i=0; i<len; i++)
          temp[i] = tim[i] + (float)(k*step);   /* time shift by k steps */
   result = 0.0;
   apart = (double)step;
   j = 0;

   for (i=0; i<len; i++)
   {  dt = temp[i];
          while (j<2*len)
          {  diff = dtime[j] - dt;
            if (diff>apart)
              {   back = j - 2;
                    if (back > 0)
                          j = back; /* set pointer back */
                    else
                          j = 0;
                    break;
              }
                  /* dtime[j] passes target temp[i]; next i */

            if ( fabs(diff) < tol)     /* match */
              { result = result + input[i]*dinput[j];
                    break;
                  }       /* next i */
            j++;        /* next j */
          }
   }
   return result;
}
```