Computational Complexity Comparison of Second-Order Volterra Filtering Algorithms

Sungbin Im*

※This work is supported by the Gyonae Research 96 of Soongsil University.

ABSTRACT

The objective of the paper is to compare the computational complexity of five algorithms for computing time-domain second-order Volterra filter outputs in terms of number of real multiplication and addition operations required for implementation. This study shows that if the filter memory length is greater than or equal to 16, the fast algorithm using the overlap-save method and the frequency-domain symmetry properties of the quadratic coefficients is the most efficient among the algorithms investigated in this paper. When the filter memory length is less than 16, the algorithm using the time-domain symmetry properties is better than any other algorithm.

I. Introduction

Recent research has demonstrated the importance of using Volterra filters in nonlinear signal processing and nonlinear system modeling. However, since the Volterra filter is a nonparametric model [1], in most cases, the Volterra filter is characterized by a large number of filter coefficients. In general, the large number of coefficients results in high computational complexity, which has tended to prevent the widespread application of Volterra filters to those practical problems requiring fast computation.

Two fast algorithms were put forward in [2] and [3] for reducing the computational complexity associated with the Volterra filters. It is notable that both algorithms utilize the FFT algorithm. However, the algorithm presented in [2] is based on the one-and two-dimensional overlap-save methods [4] while the algorithm in [3] utilizes the one-dimensional FFT-based convolutions when computing the quadratic output of a second-order Volterra filter. In addition, one should consider that the utilization of the symmetry properties of the quadratic Volterra filter coefficients may reduce the computational complexity. Unfortunately, there are no analyses available which indicate how much one can save with respect to the computational complexity when utilizing the symmetry properties. Motivated by these issues, in this paper we quantify the computational complexity of each of five algorithms in terms of number of real addition and multiplication operations. In two of the five cases we consider the reduction in complexity by considering second-order Volterra coefficient symmetries in either the time or frequency domains. Furthermore, we provide the complexity ratios of the algorithms with respect to the conventional algorithm.

The remainder of this paper is organized as follows. The following section describes the algorithms and quantifies the complexity of each algorithm in terms of the number of real multiplies and adds. In Section III, each algorithm is compared in terms of complexity ratio measured relative to the conventional algorithm. If the filter memory length is greater than or equal to 16, the fast algorithm using the overlap-save method and the frequency-domain symmetry properties of the quadratic coefficients is the most efficient among the algorithms investigated in this paper. When the filter memory length is less than 16, the algorithm using the time-domain symmetry properties is better than any other algorithm. Finally, the paper is concluded in Section IV.

II. Algorithms and Computational Complexity

If we assume that the nonlinear system to be represented by a second-order Volterra filter is stable and has finite memory, the Volterra filter [1] can approximate the output

*Department of Information and Telecommunication Engineering, Soongsil University.
Manuscript Received: August 1, 1997.

of the nonlinear system by its sampled data form; the output of which can be represented as

$$y(n)=\sum_{i=0}^{N-1} h_1(i)x(n-i)+\sum_{j=0}^{N-1}\sum_{k=0}^{N-1} h_2(j,k)x(n-j)x(n-k) \tag{1}$$

where $x(\cdot)$ denotes the input sequence to the filter, $y(\cdot)$ denotes the Volterra filter output, and $h_1(\cdot)$ and $h_2(\cdot,\cdot)$ represent the linear and quadratic Volterra filter coefficients, respectively. N is the filter memory length. Thus, the number of the linear coefficients is N while that of the quadratic ones is N^2.

In the following, we describe the algorithms considered in this paper and compute the number of real addition and multiplication required for implementing each algorithm. Throughout this analysis, we assume that the input sequence and the time-domain filter coefficients are real. The algorithms based on the time-domain operations take advantage of this assumption while those based on frequency-domain operations can not since they use complex arithmetic operations even for real time series data

A. Algorithm 1 : Standard Algorithm

This algorithm directly utilizes (1) to compute the time-domain filter output $y(n)$. As shown in (1), this algorithm requires implementation of one and two-dimensional convolutions in order to compute the linear and quadratic components of the filter output, respectively. Note that this algorithm does not take advantage of the symmetry properties of the quadratic filter coefficients, $h_2(j,k)=h_2(k,j)$, which will be discussed in the next section.

In this case, to produce one output point in time requires N real multiplies and $N-1$ real adds in the linear component; and $2N^2$ real multiplies and N^2-1 real adds in the quadratic component. 1 real add is required for summation of each component output which produces a final filtered output point. Thus, for one filtered output point, the total numbers of real multiplies and adds are $2N^2+N$ and N^2+N-1, respectively. This algorithm will be used as a base against which to compare the following four algorithms.

B. Algorithm 2

It is well-known that the quadratic Volterra coefficients can be assumed symmetric without any loss of generality [1]. That is,

$$h_2(j,k)=h_2(k,j) \tag{2}$$

When utilizing these symmetry properties in computing the time-domain filter output, we should consider the symmetry factor $I(j,k)$, which is defined by

$$I(j,k)=\begin{cases}1, & \text{if } j=k\\ 2, & \text{if } j\neq k\end{cases} \tag{3}$$

because $h_2(j,k)x(n-j)x(n-k)=h_2(k,j)x(n-k)x(n-j)$. With these symmetry properties, (1) can be rewritten as follows:

$$\begin{aligned}y(n)=&\sum_{i=0}^{N-1} h_1(i)x(n-i)\\&+\sum_{j=0}^{N-1}\sum_{k=j}^{N-1} h_2^s(j,k)x(n-j)x(n-k)\end{aligned} \tag{4}$$

where $h_2^s(j,k)=I(j,k)h_2(j,k)$.

Algorithm 2 utilizes (4) rather than (1) to compute the time-domain filter output $y(n)$. When counting the number of arithmetic operations required for Algorithm 2, the number of multiplies required for computing h_2^s is not taken into account because this cost may or may not be relevant depending upon whether the filter coefficients are initially given in the form of h_2 or h_2^s.

In (4), the number of linear coefficients is N, while the number of quadratic coefficients $h_2^s(j,k)$ is $N(N+1)/2$. Thus, to produce one output point in time requires N real multiplies and $N-1$ real adds in the linear component; and $N(N+1)$ real multiplies and $N(N+1)/2-1$ real adds in the quadratic component. In addition, 1 real add is required for summation of linear and quadratic component outputs. Thus, for one filtered output point, the total numbers of real multiplies and adds are N^2+2N and $0.5N^2+1.5N-1$, respectively.

Algorithms 1 and 2 utilize the time-domain real arithmetic operations based on the time-domain Volterra filters (1) and (4), respectively. However, Algorithms 3 to 5, which will be discussed in the following, employ the discrete Fourier transform (DFT) and inverse DFT (IDFT) algorithms, with the complex arithmetic operations in the frequency domain, to compute the filtered outputs. Actually, these algorithms utilize an M-point radix-2 FFT algorithm rather than the DFT algorithm. It is known that the M-point radix-2 FFT algorithm requires approximately $2M\log_2 M$ real multiplies and $3M\log_2 M$ real adds [5].

The relationship between complex and real arithmetic operations is given by

$$1 \text{ complex add} = 2 \text{ real adds} \tag{5}$$

$$1 \text{ complex multiply} = 4 \text{ real multiplies} + 2 \text{ real adds.} \tag{6}$$

C. Algorithm 3

Algorithm 3 presented in [2] is a fast algorithm for computing the time-domain output of a second-order Volterra filter using the corresponding frequency-domain Volterra filter and the overlap-save method. The discrete frequency-domain version [2] of the time-domain Volterra filter (1) is given by

$$Y(m) = H_1(m)X(m) + \sum_{p=0}^{N-1}\sum_{q=0}^{M-1} H_2(p,q)X(p)X(q)\delta_M(m-p-q) \tag{7}$$

where

$$\delta_M(m) = \begin{cases} 1, & \text{if } (m \text{ modulo } M) = 0 \\ 0, & \text{if } (m \text{ modulo } M) \neq 0 \end{cases} \tag{8}$$

In (7), $Y(m)$, $X(m)$, $H_1(m)$, and $H_2(p, q)$ are DFT's of $y(n)$, $x(n)$, $h_1(n)$, and $h_2(j, k)$, respectively. Note that $\delta_M(\cdot)$ is a modulo function.

In order to compute the time-domain filter output, Algorithm 3 utilizes the frequency-domain Volterra filter (7), the overlap-save method [4], [5], and the M-point radix-2 FFT algorithm [2]. First, an M-point segment consisting of $x(\cdot)$ which overlaps a previous segment by N points (M > N), is transformed by using the M-point FFT algorithm. An M-point output segment of $Y(m)$ is computed by applying the frequency-domain Volterra filter (7) to the FFTed input segment $\{X(0), ..., X(M-1)\}$. This output segment is inverse transformed into the time domain. The first $N-1$ points in the inverse FFTed output segment should be discarded because these points are incorrect due to the circular convolution effect. The remaining correct $M-N+1$ points are appended to those from the previous output segments, which correspond to the time-domain filter output points.

Note that the arithmetic operations involved in Algorithm 3 are complex operations. First, we compute the computational complexity of Algorithm 3 in terms of the number of complex operations, and then, use (5) and (6) to transform the number of complex operations to the number of real operations. Since Algorithm 3 uses the FFT algorithm and the sectioning technique to implement a linear convolution, the computational complexity of the algorithm depends also on the complexity of the FFT algorithm. Algorithm 3 requires only two one-dimensional FFT's to transform each input and output segment and one one-and two-dimensional FFT's to transform the linear and quadratic Volterra filter coefficients, respectively. However, we do not take the latter one- and two-dimensional FFT's to transform the linear and quadratic Volterra filter coefficients into consideration when counting the number of arithmetic operations, since this cost may or may not be relevant depending upon whether the Volterra filter coefficients are initially given in the frequency domain or time domain. To further reduce the complexity, we assume that the length of each input segment, M, (to be transformed) is a power of two.

For one output frequency bin, Algorithm 3 requires 1 complex multiply in the linear component; $2M$ complex multiplies and $M-1$ complex adds in the quadratic component; and 1 complex add for the summation of linear and quadratic outputs. Thus $1+2M$ complex multiplies and M complex adds are required for one output frequency bin. These numbers of complex arithmetic operations are equivalent to $4(1+2M)$ real multiplies and $2M+2(1+2M)$ real adds according to (5) and (6). This leads to a total of $4M(1+2M)+2(2M\log_2 M)$ real multiplies and $2M^2+2M(1+2M)+2(3M\log_2 M)$ real adds per segment, which include the complexity of the M-point radix-2 FFT algorithm mentioned previously. Note that an output segment consisting of M points contains $M-N+1$ correct time-domain output points. For one filtered output point, therefore, the total number of real multiplies and adds are $\{4M(1+2M)+2(2M\log_2 M)\}/(M-N+1)$ and $\{2M^2+2M(1+2M)+2(3M\log_2 M)\}/(M-N+1)$, respectively.

D. Algorithm 4

As Algorithm 2 utilizes the symmetry properties in the time domain, Algorithm 4 introduces the symmetry properties of the frequency-domain quadratic Volterra filter coefficients into Algorithm 3. The frequency-domain quadratic Volterra filter coefficients have the following symmetry properties [6];

$$H_2(p, q) = H_2(q, p) \tag{9}$$

Utilizing these symmetry properties in computing filtered output points, as in Algorithm 2, we should consider the frequency-domain symmetry factor $I(p, q)$, which is defined as (3). However, when counting the number of multiplies, we do not take the multiplies for the symmetry factors into account for the same reason stated in Algorithm 2. The number of the quadratic terms which contribute to

$Y(m)$ reduces to $\frac{M}{2}+1$ or $\frac{M}{2}$ depending on whether the output frequency bin number m is even or odd, respectively. The numbers of even and odd m's are $\frac{M}{4}+1$ and $\frac{M}{4}$, respectively. The reason for this is that the time-domain sequence is real.

For an even bin m, Algorithm 4 requires 1 complex multiply in the linear component; $2(\frac{M}{2}+1)$ complex multiplies and $\frac{M}{2}$ complex adds in the quadratic component; and 1 complex add for the summation of the linear and quadratic components. For an odd bin, 1 complex multiply in the linear component; $2(\frac{M}{2})$ complex multiplies and $\frac{M}{2}-1$ complex adds in the quadratic component; and 1 complex add for the summation are required. Thus, for an M-point frequency-domain output sequence, the total numbers of complex multiplies and adds are $\frac{1}{2}M^2+2M+3$ and $\frac{1}{4}M^2+\frac{3}{4}M+1$, respectively. Thus, for an M-point time-domain output segment (actually, only $M-N+1$ points are valid), $2M^2+8M+12+2(2M\log_2 M)$ real multiplies and $\frac{3}{2}M^2+\frac{11}{2}M+8+2(3M\log_2 M)$ real adds are needed. Finally, for one filtered output point in the time domain, the total number of real multiplies and adds are $\{2M^2+8M+12+2(2M\log_2 M)\}/(M-N+1)$ and $\{\frac{3}{2}M^2+\frac{11}{2}M+8+2(3M\log_2 M)\}/(M-N+1)$, respectively.

E. Algorithm 5

The basic idea of Algorithm 5 proposed in [3] is to reduce the computational complexity of the second-order Volterra filter by replacing the two-dimensional convolution of the quadratic Volterra filter with several one-dimensional operations based on the one-dimensional FFT and IFFT algorithms. Algorithm 5 decomposes the $L\times L$ quadratic Volterra filter coefficient matrix of (1) into L filter coefficient vectors of size $2L\times 1$ each. In [3], it is assumed that the filter memory size is equal to the length of the input sequence. For this reason, the notation for the system memory length is different from N of (1). Generally, it is known that a convolution of two sequences of length L generates an output sequence of $(2L-1)$ points. In [3], however, one zero is appended to the output sequence so that the length of the sequence becomes $2L$. The filter coefficient vectors are defined by

$$h_j=[\underbrace{0,\ldots,0}_{\leftarrow j\rightarrow},h(0,j),h(1,j+1),\ldots,h(L-j-1,L-1),\underbrace{0,\ldots,0}_{\leftarrow L\rightarrow}]^T \tag{10}$$

where $j=0,\ldots,L-1$. The quadratic input vectors are constructed as follows;

$$x_j=[x(0)x(j),x(1)x(j+1),\ldots,x(L-j-1)x(L-1),\underbrace{0,\ldots\ldots,0}_{\leftarrow L+j\rightarrow}]^T \tag{11}$$

Algorithm 5 applies the 2L-point FFT to the vectors $\overline{h}_j$ and $\overline{x}_j$ for $j=0,\ldots,L-1$. Then, the following relation is used for computing the frequency-domain quadratic output sequence;

$$Y_2(i)=H_0(i)X_0(i)+2\sum_{j=1}^{L-1}H_j(i)X_j(i) \tag{12}$$

where $i=0,1,\ldots,2L-1$. In (12), $H_j(i)$ and $X_j(i)$ represent the i-th component of the frequency-domain versions of $\overline{h}_j$ and $\overline{x}_j$, respectively. The time-domain quadratic output sequence is obtained by applying the IFFT to $Y_2(i)$ with $i=0,1,\ldots,2L-1$. In this algorithm, note that the length of the coefficient vectors h_j depends on the input-output sequence not on the system memory size N. For this reason, it is difficult to compare Algorithm 5 with the previous algorithms. Thus, in the following section, we consider two specific cases to facilitate comparison with the previous algorithms.

On the basis of a careful analysis of the algorithm provided in [3], we have counted the number of real multiplies and adds required for implementing Algorithm 5, because the complexity analysis of Algorithm 5 provided in [3] is not sufficient for this study. For each time-domain output point, the total number of the real multiplies is given by

$$\frac{4P+3(2P\log_2 P)+\frac{1}{2}L(L+1)+(2L+1)(2P\log_2 P)}{2L}, \tag{13}$$

while the number of the real adds is

$$\frac{2P+3(3P\log_2 P)+(2L+1)(3P\log_2 P)+4L^2+4L(L-1)+2L}{2L}. \tag{14}$$

In (13) and (14), as in [3], we assume that the time-domain Volterra filter memory size is equal to that of the input sequence L. P represents $2L$.

Ⅲ. Comparison

In Table I, we summarize the numbers of the real multiplies and adds required for implementing each algorithm. In Table I, N and M denote the filter memory size and the section length of the FFT, respectively, while for Algorithm 5, L represents the length of the filter memory and the input sequence and $P=2L$. Note that Algorithm 5 assumes that the filter memory length should be equal to the length of the input sequence to the second-order Volterra filter. Thus, it is difficult to directly compare the computational complexity of Algorithm 5 with those of the remaining algorithms. For this reason, we will compare the complexity of Algorithm 5 in a subsequent subsection.

In order to compare the computational complexity of the algorithms, the computational ratios of the real multiplies and adds of each algorithm are computed with respect to Algorithm 1 for various filter memory lengths. The ratios are defined as follows;

$$\mathrm{RM}_i=\frac{\text{Number of Multiplies for Algorithm i}}{\text{Number of Multiplies for Algorithm 1}} \tag{15}$$

and

$$\mathrm{RA}_i=\frac{\text{Number of Adds for Algorithm i}}{\text{Number of Adds for Algorithm 1}} \tag{16}$$

where $i=1$, 2, 3, and 4. RM_i and RA_i represent the complexity ratios of real multiplies and adds required for implementing Algorithm i, respectively. For Algorithms 3 and 4, we have two variables to consider: the filter mem-

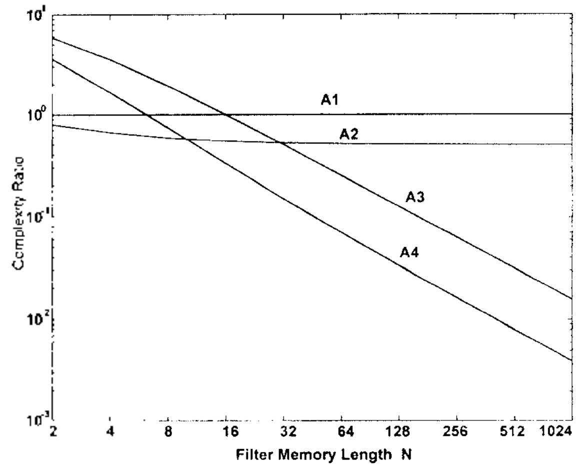

Fig. 1. Complexity ratios of real multiplies for Algorithms 1 to 4 relative to Algorithm 1 assuming $M=2N$. A1 to A4 represent Algorithms 1 to 4, respectively.

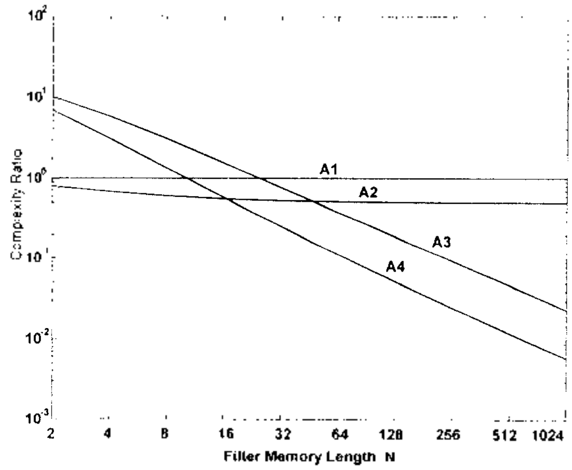

Fig. 2. Complexity rations of real adds for Algorithms 1 to 4 relative to Algorithm 1 assuming $M=2N$. A1 to A4 represent Algorithms 1 to 4, respectively.

Table 1. Computational complexity measured in terms of the number of real multiplies and adds required to compute one filtered output data point. N and M denote the filter memory size and the section length of the FFT, respectively. For Algorithm 5, L represents the length of the filter memory and the input sequence and $P=2L$.

Algorithms	No. of Real Multiplies	No. of Real Adds
1	$2N^2+N$	N^2+N-1
2	N^2+2N	$0.5N^2+1.5N-1$
3	$\frac{4M(1+2M)+2(2M\log_2 M)}{M-N+1}$	$\frac{2M^2+2M(1+2M)+2(3M\log_2 M)}{M-N+1}$
4	$\frac{2M^2+8M+12+2(2M\log_2 M)}{M-N+1}$	$\frac{1.5M^2+5.5M+8+2(3M\log_2 M)}{M-N+1}$
5	$\frac{4P+0.5L(L+1)+(2L+4)(2P\log_2 P)}{2L}$	$\frac{2P+(2L+4)(3P\log_2 P)+8L^2-2L}{2L}$

ory length N and the FFT segment length M. For simplicity of comparison, the segment length M is set to $2N$, where N is assumed to be a power of 2. Figs. 1 and 2 show the complexity ratios of each algorithm in terms of the number of real multiplies and adds, respectively. N varies from 2^1 to 2^{10}. In Figs. 1 and 2, A1 through A4 represent Algorithms 1 to 4, respectively.

Since the complexity ratios are computed with respect to Algorithm 1, the curves of Algorithm 1 denoted by A1 in Figs. 1 and 2, are equal to 1 for each filter memory length. We see that the complexity ratio curves of Algorithm 2 are always lower than the curves of Algorithm 1 and converge to 0.5 as the filter length N increases. This implies that the complexity of Algorithm 2 is approximately half of the complexity of Algorithm 1 for a sufficiently large N. This complexity reduction is obtained by utilizing the symmetry properties of the time-domain quadratic Volterra filter coefficients in computing the Volterra filter outputs.

The complexity ratio for Algorithms 3 decreases exponentially for the various filter memory lengths. Note that Figs. 1 and 2 are plotted in log scale. For the filter memory lengths $N > 16$, the complexity ratios of multiplies for Algorithm 3 are less than 1, while those of adds for Algorithm 3 become less than 1 when $N \geq 32$. This indicates that Algorithm 3 becomes superior to Algorithm 1 in terms of the number of real multiplies and adds for $N \geq 32$. This efficacy is obtained because of the use of the discrete frequency-domain Volterra filter and the 1-D FFT algorithm, as opposed to performing 1-D and 2-D convolutions in the discrete time domain.

The complexity of Algorithm 4 decreases more rapidly than that of Algorithm 3 as N increases. This is due to the use of the frequency-domain symmetry properties in addition to use of the discrete frequency-domain Volterra filter and the 1-D FFT algorithm in Algorithm 3. In terms of the number of real multiples, Algorithm 4 becomes superior to Algorithm 1 for the filter memory lengths $N \geq 8$, while for $N \geq 16$, Algorithm 4 is best in terms of the number of real adds. Furthermore, the complexity of Algorithm 4 is always lower than that of Algorithm 3 and becomes equal to about one quarter of that of Algorithm 3 for sufficiently large N.

Generally speaking, Algorithm 4 performs better than any of the other four algorithms for the filter memory lengths $N > 16$. In the remaining cases, that is, $N = 2$, 4, or 8, Algorithm 2 is recommended.

Algorithm 5

In this subsection, we compare the computational complexity of Algorithm 5 to those of Algorithms 1 and 4. As mentioned previously, Algorithm 5 is developed based on the assumption that the filter memory length is equal to the input sequence length, and, thus that the length of the output sequence is twice the input sequence length. Because of this assumption, we cannot employ the relatively simple comparison framework utilized previously. Thus, the complexity of Algorithm 5 is considered for two cases: when $N = L$ and when $N < L$. In both cases, we measure the complexity ratios of Algorithm 5 with respect to Algorithm 1 in terms of the numbers of real multiplies and adds.

The first case we consider is when the length of the input sequence L is equal to the filter memory length N, that is, when the assumption for Algorithm 5 is satisfied. In this case, the complexity ratios of multiplies and adds are given by

$$\mathrm{RM}_5 = \frac{4 + 0.25(N+1) + 2(2N+4)\log_2(2N)}{2N^2 + N} \tag{17}$$

and

$$\mathrm{RA}_5 = \frac{1 + 3(2N+4)\log_2(2N) + 4N}{N^2 + N - 1} \tag{18}$$

In Figs. 3 and 4, the complexity ratios, RM_5 and RA_5, denoted by A5 are plotted for $N = 2, 4, 8, \ldots, 1024$. For the purpose of comparison, RM_4 and RA_4 of Algorithm 4 are also plotted in Figs. 3 and4. We see that for each filter memory length, the curves for Algorithm 5 are higher

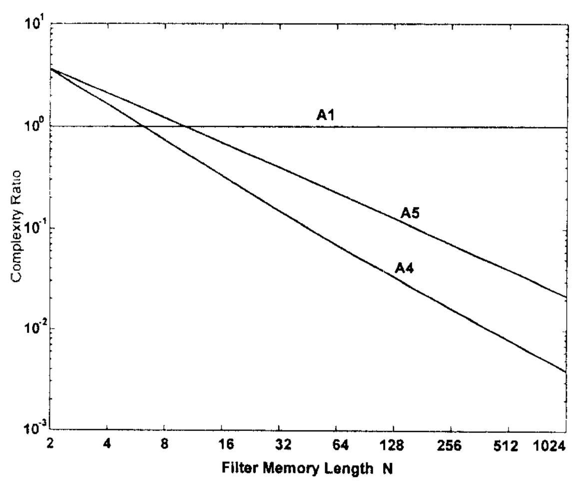

Fig. 3. Complexity ratios of real multiplies for Algorithms 1, 4, and 5 relative to Algorithm 1 assuming that the filter length is equal to the input sequence length and that $M = 2N$. A1, A4, and A5 represent Algorithms 1, 4, and 5, respectively.

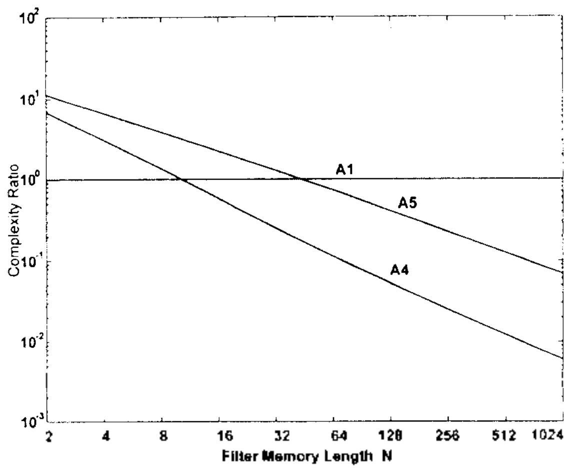

Fig. 4. Complexity ratios of real adds for Algorithms 1, 4, and 5 relative to Algorithm 1 assuming that the filter length is equal to the input sequence length and that $M = 2N$. A1, A4, and A5 represent Algorithms 1, 4 and 5, respectively.

than those of Algorithm 4. Thus, even for $N = L$, Algorithm 4 is more efficient than Algorithm 5. It is $N > 64$ when Algorithm 5 becomes superior to the standard algorithm, Algorithm 1, while Algorithm 4 does when $N > 16$.

Next, we consider the case that the filter memory length N is less than the input sequence length L. Thus, the actual output sequence length is $L + N - 1$. In order to apply Algorithm 5 to this situation, the memory length of the Volterra filter is supposed to be L by appending zeros to the non-zero actual Volterra filter coefficients. For this reason, the numbers of real multiplies and adds are same to those for the output sequence of $2L$ samples. As mentioned previously, however, the length of the actual output sequence is $L + N - 1$. Thus, the numbers of real multiplies and adds given in Table 1 is increased by a factor of $2L/(L + N - 1)$. For simplicity of comparison, we assume that there is a positive number β satisfying $L = \beta N$. Then, the numbers of real multiplies and adds required for computing one output point are given by

$$\beta\frac{8N + 0.5N(\beta N + 1) + 4N(2\beta N + 4)\log_2(2\beta N)}{(\beta + 1)N + 1} \qquad (19)$$

$$\beta\frac{2N + 6N(2\beta N + 4)\log_2(2\beta N) + 8\beta N^2}{(\beta + 1)N - 1} \qquad (20)$$

respectively. From (19) and (20), we see that the numbers of real multiplies and adds are dependent upon β. This implies that the complexity of Algorithm 5 varies according to the relative length between the filter memory length and input sequence length under the assumption $N < L$. This does not give a clear comparison. Thus, for $\beta = 10$, 1000 and 100000, we calculate the complexity ratios of Algorithm 5 relative to Algorithm 1 by increasing the filter memory length N from 2^1 to 2^{15}. The results are plotted in Figs. 5 and 6. Fig. 5 is for the number of real multiplies while Fig. 6 is for the number of real adds. The curves denoted by A4 in Figs. 5 and 6 are for the complexity ratios of Algorithm 4 under the same condition. In Figs. 5 and 6, we observe that the complexity ratios of Algorithm 5 increase as β increases while that of Algorithm 4 is independent of β, that is, the input sequence length. Furthermore, for any value of β, the corresponding curve of Algorithm 5 is higher than that of Algorithm 4, which indicates that Algorithm 4 is more efficient than Algorithm 5.

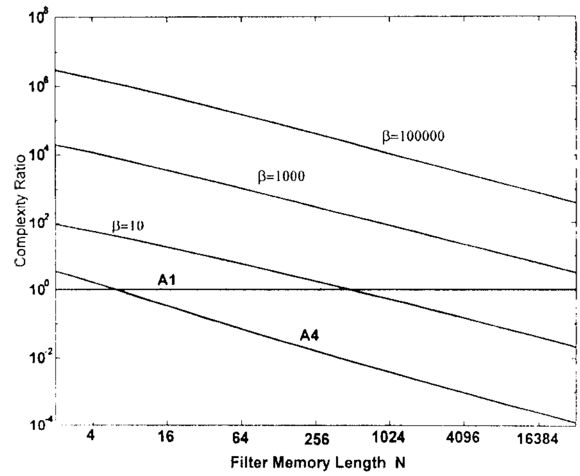

Fig. 5. Complexity ratios of real multiplies for Algorithms 1, 4, and 5 relative to Algorithm 1 assuming that $L = \beta N$, where $\beta = 10$, 1000, and 100000. A1 and A4 represent Algorithms 1 and 4, respectively.

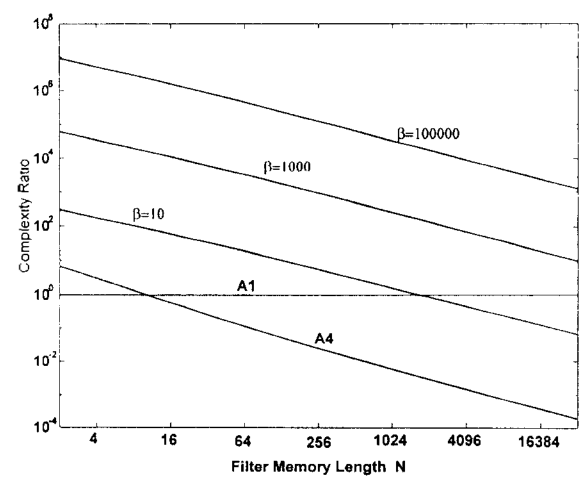

Fig. 6. Complexity ratios of real adds for Algorithms 1, 4, and 5 relative to Algorithm 1 assuming that $L = \beta N$, where $\beta = 10$, 1000, and 100000. A1 and A4 represent Algorithm 1 and 4, respectively.

Generally speaking, if the assumption, that the length of the filter memory is equal to that of the input sequence, required for Algorithm 5 is satisfied, Algorithm 5 is more efficient than the standard algorithm, Algorithm 1 for the filter memory lengths greater than 64. Otherwise, the complexity of Algorithm 5 is dependent also on the ratio of the input sequence length to the filter memory length. For an input sequence relatively longer than the filter memory length, for example, $\beta = 1000$ and $N = 256$, the complexity of Algorithm 5 is much higher than that of Algorithm 1. It is notable that the complexity of Algorithm 4 is lower than that of Algorithm 5 for any case.

Ⅳ. Conclusion

In this paper, we compare five second-order Volterra filtering algorithms in terms of computational complexity. The algorithms considered in the paper are as follows;

Algorithm 1: Time-domain one and two-dimensional convolution approach based on (1).

Algorithm 2: Time-domain one and two-dimensional convolution approach where the symmetry properties of the quadratic Volterra filter coefficients [1] are utilized when computing the quadratic component of the Volterra filter output.

Algorithm 3: Frequency-domain fast approach presented in [2].

Algorithm 4: Frequency-domain fast approach of [2] but in which the symmetry properties of frequency-domain quadratic Volterra filter coefficients [6] are utilized.

Algorithm 5: Fast algorithm proposed in [3].

The computation complexity of each algorithm is measured using the numbers of real multiplies and adds required for implementation. The numbers of real multiplies and adds for each algorithm are summarized in Table I. In order to demonstrate the relative efficacy of each algorithm, the complexity ratios defined by (15) and (16) are computed and plotted for the various filter memory lengths. For this, it is assumed that the segment length M is equal to $2N$ for Algorithms 3 and 4. For Algorithm 5, we consider the two cases; $L = N$ and $L = \beta N$ with $\beta = 10$, 1000, and 100000. Within this comparison framework, we can make several observations, as follows:

- Algorithm 2 is less complex than Algorithm 1 for any filter length. Furthermore, for large filter lengths, the complexity of Algorithm 2 is half of that of Algorithm 1.
- When the Volterra filter memory lengths are greater than or equal to 32, Algorithm 3 becomes superior to Algorithm 1.
- When the filter memory lengths are greater than or equal to 16, Algorithm 4 performs better than Algorithm 1. The complexity of Algorithm 4 is about one quarter of that of Algorithm 3 for sufficiently large filter lengths ($N \geq 128$).
- When the assumption for Algorithm 5 is satisfied, it becomes better than Algorithm 1 for the filter memory lengths greater than or equal to 64, while it is less efficient than Algorithm 4 for any filter memory length. When the filter memory length is shorter than an input sequence, the complexity of Algorithm 5 shows dependency on the relative lengths between the filter memory and the input sequence. Even in this case, Algorithm 4 exhibits less complexity than Algorithm 5.
- According to the results for Algorithms 2 and 4, the utilization of the symmetry properties of the quadratic Volterra filter coefficients in the time-domain algorithm and the frequency-domain algorithm reduces the computational complexity.

Clearly, Algorithm 4 is the most efficient one among the algorithms investigated in this paper in terms of the number of real multiplies and adds for $N \geq 16$.

References

1. M. Schetzen, *The Volterra and Wiener Theories of Nonlinear Systems*, John Wiley and Sons, New York, 1980.
2. S. Im and E.J. Powers, "A Fast Method of Discrete Third-Order Volterra Filtering," *IEEE Trans. on Signal Processing*, vol. 44, no. 9, pp. 2195-2208, September 1996.
3. M. Morháč, "A Fast Algorithm of Nonlinear Volterra Filtering," *IEEE Trans. on Signal Processing*, vol. 39, no. 10, pp. 2353-2356, October 1991.
4. A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice-Hall Inc., New Jersey, 1975.
5. H.D. Helms, "Fast Fourier Transform Method of Computing Difference Equations and Simulating Filters," in B. Gold and C.M. Rader (ed.), *IEEE Trans. Audio Electroacoust.* vol. AU-15, pp. 85-90, June 1967.
6. S.W. Nam and E.J. Powers, "Application of Higher-order Spectral Analysis to Cubically-Nonlinear System Identification," *IEEE Trans. on Signal Processing*, vol. 42, no. 7, pp. 1746-1765, July 1994.

▲Sungbin Im

Sungbin Im was born in Seoul, Korea, in 1963. He received the B. S. and M. S. degrees in electronics engineering from Seoul National University, Seoul, Korea, in 1986 and 1988, respectively, and the Ph.D. degree in electrical and computer engineering from The University of Texas at Austin, 1994.

In 1995, he joined the Department of Information and Telecommunication Engineering at Soongsil University, Seoul, Korea. His research interests lie in nonlinear signal processing and its application to communications, speech processing, and system modeling.