An efficient deep memory algorithm for computing fractional order operators

Steven Dorsher[a], Gary Bohannan[a]

[a]*Department of Chemistry and Physics, Saint Cloud State University, Saint Cloud, Minnesota*

Abstract

This paper outlines a method to achieve effective bandwidth of five or more decades in the approximation of a fractional order derivative. This constitutes an increase of two decades or more over current algorithms, such the Infinite Impulse Response (IIR) method based on continued fraction expansion, while demanding only slightly more computational steps and processor memory.

Keywords: fractional calculus, numerical methods, computational efficiency
2010 MSC: 65Y02, 65Z02, 65Y04, 65Z04

1. Introduction

Interest in the application of the fractional calculus has been growing at an ever increasing rate. Of long term and continued interest is in the use of fractional order (FO) operators, such as integrators and differentiators, in motion control applications. [1] While wide bandwidth analog controllers have been successfully demonstrated [2], fractional order analog circuit elements are not generally available and the prototypes that have been demonstrated do not have the ability to be retuned for a specific desired phase. [3]

Unfortunately, the existing techniques for digital approximation of FO operators are limited in bandwidth to on the order of three and a half decades of frequency response while nonlinear effects in motion control systems can span five decades or more. This places a severe constraint on the design and implementation of digital FO controllers, i.e. how to set the sampling frequency to meet the high speed requirements necessitated by the Nyquist sampling rate while at the same time providing enough deep memory to get

low offset error. Additionally, the infinite impulse response type of implementation cannot guarantee stability due to the limitations of finite precision arithmetic. See e.g. [4].

Given the current necessity to implement FO controls in digital form, it is desirable to obtain the most efficient algorithm to compute a fractional order operator while maximizing the numerical stability of the algorithm. Efficiency to be measured in both memory utilization and number of computations per time step. This paper outlines a computational method inspired by the Riemann-Liouville integral definition and the Grünwald algorithm. [5] The essential concept is the rescaling of time by successive accumulation of older data into increasing size bins for deeper memory.

Outline
. We will first briefly describe the current state of the art in fractional order operator approximation and then describe a novel approach based on successive binning of older data.

- Section 2 will contain algorithm definitions.
- Section 3 will contain an analysis of the computational resources required for the larger versus the smaller number of memory registers stored.
- Section 4 will contain a description of the numerical simulation written to verify the algorithm described in Section 2.
- Section 5 will contain the amplitude and phase response of these algorithms for a large and small number of bins in the partition.

2. Algorithm definitions

ADD REFERENCES

The technology most widely used today in digital fractional order circuitry is the continued fraction expansion (CFE) approximation to s^α. Its benefits are that it was a flat phase response over approximately two and a half decades in phase for a 9th order expansion (10 registers of input signal memory). It would be desirable to find an algorithm with an even broader flat phase frequency bandwidth and with a comparably small amount of memory required. We take the Grunwald algorithm as a starting point. As the signal

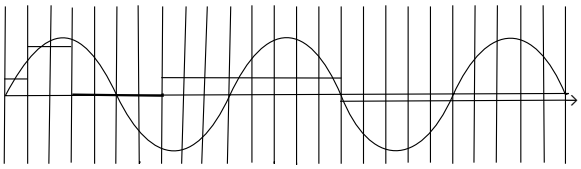

Figure 1: Let each division represent a single time step and each bar represent a bin, scaled such that the duration of a bin increases further back in time (toward the right). The value of the bin is the average of the input signal, the sine wave, at each point in time within that bin. There exists some oldest bin that responds sensitively to the input signal (in this figure, the second bin). For lower frequencies, that oldest bin moves to older times. This illustrates the sensitivity of short bins at recent times to high frequencies and long bins in the distant past to low frequencies.

history retained in the Grunwald sum grows longer, the bandwidth of flat phase response grows broader; however, the memory required also increases with input signal history length. To reduce this memory requirement and retain a long signal history, we propose partitioning the input signal history into bins that are longer further into the past. The value of the binned input signal is represented by the average value of the input signal within that bin. The presence of short bins at recent times maintains sensitivity to high frequencies, while the inclusion of long bins at past times adds sensitivity to low frequencies that would not usually be present in a Grunwald sum with the same number of terms. (See Figure 2).

2.1. Modified Grunwald

The Grunwald form of the fractional integral can be written

$$ {}_0D_t^\alpha f(t) = \lim_{N\to\infty} \left(\frac{t}{N_t}\right)^{-\alpha} \sum_{j=0}^{N_t-1} w_j x_j \tag{1} $$

where $f(t)$ is the input signal at time t, the jth value of the input signal history is $x_j = f\left(t - \frac{j\Delta t}{N_t}\right)$, and the jth Grunwald weight is

$$ w_j = \frac{\Gamma(j-\alpha)}{\Gamma(j+1)\Gamma(-\alpha)}. \tag{2} $$

To include more distant history at low computational cost, we modify the Grunwald sum of Equation 1 by partitioning its history into N_b bins. In each bin k, the input signal history x_j is represented by its average value over that bin, X_k.

Since we make the assumption that each value is well represented by its average within a bin, we can define a value for the "bin coefficient" by summing the Grunwald coefficients within that bin.

$$W_k = \sum_{j=p_{k-1}+1}^{p_k} w_j \tag{3}$$

where w_j is summed from the lowest index of the input data history within bin k to the highest index p_k within that bin. There is an additional factor that goes into $\bar{W}$ that will be discussed in Section 2.2.

With these definitions, the modified Grunwald differ-integral can be written

$${}_0D_t^\alpha f(t) = (\Delta t)^{-\alpha} \sum_{k=0}^{N_b} \bar{W}_k X_k \tag{4}$$

where Δt is the interval between time samples.

2.2. Updating the average history

When a new input data element is read, the history is updated. The new data element is shifted into the first bin through a weighted average. Since data elements represent time steps, they should be incompressible– when one element is shifted into a bin, another virtual element should be shifted out of that bin if the bin is full. It shifts into the next bin, and pushes a virtual element out of that one, until a bin which is partially full or empty is reached. To update the average data stored in the bins, we take the weighted average obtained by adding one virtual element from the $(k-1)$th bin to the b_k elements in the kth bin. This process is illustrated in Figure 2.2.

During start-up, it will be necessary to consider bins that have some set size b_k, but are not filled to that capacity. In that case, it is the current occupation number c_k of each bin that enters the calculation. If the kth bin initially contains c_k elements, updating the history either leaves c_k ($c_k\prime = c_k = b_k$) or increments the number of elements in the bin such that $c_k\prime = c_k+1$ if the bin is not yet at capacity. Either way, the updated average of the value of the kth bin, $X_k\prime$, is given by

$$X_k' = \frac{c_k' - 1}{c_k'} X_k + \frac{1}{c_k'} X_{k-1}. \tag{5}$$

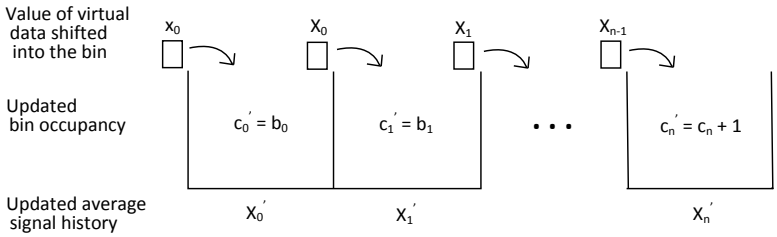

Figure 2: When the input data x_0 is read, virtual data elements with bin average values X_k are shifted from the kth to the $(k+1)$th bin. The first $n-1$ bins are at capacity, but the nth bin gains one data point. The bin averages are updated to value X'_k through a weighted average.

where X_{-1} is taken to be x_0, the input data that has just been read.

During start-up, these partially full bins may also factor into the Grunwald weights. To handle bins that are partially full, we weight the binned Grunwald weights by the ratio of the bin occupation number c_k to its capacity b_k,

$$\bar{W}_k = \frac{c_k W_k}{b_k}. \tag{6}$$

3. Computational efficiency

IS PER TIME STEP SCALING APPROPRIATE FOR A COMPUTER TARGET AUDIENCE? ARE WE BEING CLEAR WHEN WE SWITCH FROM FITTING OF A TIME SERIES TO PER TIME STEP QUANTITIES?

The computational efficiency of each algorithm concerns both the memory used by the algorithm and the number of computational steps required for the algorithm to execute, which is a measure of the speed. To characterize the operational efficiency of a digital fractor operating as a circuit element in real time, the quantities of interest are the memory or number of processor steps *per time step*.

There are three phases of the computation that factor into the computational cost of the binned Grunwald algorithm: initialization (including summing the binned Grunwald weights), updating the average values of the stored history in the bins as new input data is read, and summing the modified Grunwald differintegral.

During intialization, several things must happen. The array defining the binning structure, b_k, must be set. The history of the input signal must be zeroed and the first output must be set to zero as well. Most significantly, the binned weights must be calculated for the chosen binning structure.

The time it takes to calculate or set b_k is moderately dependent upon the details of the binning structure chosen. However, any binning structure will have a processing time that scales as N_b rather than N_t because of the number of bins that must be set. Here N_t is the maximum history depth in time-steps of the N_b memory storage bins. By definition,

$$N_t = \sum_{k=0}^{N_b} b_k. \tag{7}$$

On the other hand, since each of the N_t individual Grunwald weights must be summed, the time for initialization of the weights scales as N_t. Typical values of N_t are of order $1,000$ to 10^6. Typical values of N_b are 10 to 26. Since $N_t >> N_b$, the bulk of the initialization process occurs during the computation of the weights and the rest of the initialization process may be neglected. The time for intialization as a whole scales as N_t.

By the very design of the algorithm, there is no need to store the full Grunwald weights. The largest arrays that need to be stored are the binned weights and the bin capacity, each of which has size N_b. As a result, memory scales as N_b rather than N_t. Equation 3 gives the algorithm for the calculation of the bin weights. The following recursion formula for Grunwald weights can be used to iterate w_j to a new value with each time step. In the jth time step, w_j has the value

$$w_j = \frac{j - 1 - \alpha}{j} w_{j-1}. \tag{8}$$

In the implementation of an algorithm for initialization of the weights, individual Grunwald weights w_j need not be stored in an array after calculation. These individual weights can be summed immediately into the corresponding bin weights W_k according to Equation 3, at which time the value of w_j may be safely overwritten with w_{j+1} in order to minimize the amount of memory required such that it may be possible to later migrate the code to an MCU.

The time required to update the binned history values when new input data is read scales as N_b because there are N_b bins that must be updated.

Equation 5 may be used as the basis for an algorithm to accomplish this goal. Care must be taken to account for the filling of the bins until the approximation reaches steady-state.

Again, a memory saving opportunity is available. Since only one bin can be filling at a time and the rest must be either full ($c_k = b_k$) or empty ($c_k = 0$), the full array of N_b occupancy numbers c_k need not be stored. Instead, it is sufficient to store the index of the bin that is filling, $k_{filling}$ and its occupancy, $c_{filling}$. These unfilled bins with $k > k_{filling}$ and $c_k = 0$ are not updated when a new input signal value is read into the history. With this approach, the algorithm becomes

$$X'_k = \frac{c_{filling} - 1}{c_{filling}} X_k + \frac{1}{c_{filling}} X_{k-1} \tag{9}$$

for $k < k_{filling}$ and

$$X'_k = \frac{b_k - 1}{b_k} X_k + \frac{1}{b_k} X_{k-1} \tag{10}$$

for $k = k_{filling}$ with no shift occurring for $k > k_{filling}$ since there are no elements in those bins.

While it is possible to achieve a memory savings by storing only $c_{filling}$ and $k_{filling}$, in the C++ code reported in this paper, the full array of c_k is stored. This does not impact the memory scaling of the updating algorithm. Updating the stored bin history values scales as N_b in memory because the largest arrays that need to be stored are the binned average values, the bin capacity, and the bin occupancy, each of size N_b.

The two real-time operation stages, updating and differintegrating, together require less than 400 flops for 26 bins of history. For 26 bins, less than 100 memory registers are required. Initialization, on the other hand, is very resource-consuming. N_t may be larger than a million and still not fill 26 bins. Initialization therefore may require more than a million flops.

The binned Grunwald algorithm may be written in terms of either the number of elements per bin, b_k, or in terms of the maximum time step p_k included within each bin (see Equation 3). While we explain our algorithms in terms of b_k in this paper, the results derived from our C++ code were obtained using the p_k method.

INCLUDE ACTUAL TIMING MEASUREMENTS

4. Validation methods

While the algorithms described in Section 2 are ultimately intended for use in a digital fractor, in order to characterize their frequency response we have simulated them in C++ on a personal computer. The relative phase as a function of frequency is based upon the fractional derivative or integral of sinusoidal inputs, sweeping from low frequency to high frequency in logarithmic steps. Relative phase is reconsted using a Numerical Recipes in C algorithm to fit the last half of the output signal to the sine and cosine components of a sinusoidal signal with the same frequency as the input signal.

$$\phi' = \tan^{-1}\left(\frac{B}{C}\right) \tag{11}$$

gives the phase relative to an input cosine function over $-90°$ to $90°$, where B is the sine component and C is the cosine component of the fit. The following equation adjusts the relative phase to account for all four quadrants and to be relative to the input sine wave. **ACTUALLY USE ATAN2. FIX FIGURES OR FIX EQUATION**

$$\phi = \begin{cases} \phi' + \frac{3\pi}{2}, & \text{if } B > 0,\, C < 0 \\ \phi' + \frac{\pi}{2}, & \text{otherwise} \end{cases} \tag{12}$$

To verify phase reconstruction, we examine the phase shift of a single crossing in the time domain for an input frequency of 0.7924 Hz (Figures 3). Table 4 shows a comparison of reconstructed phases using the sinusoidal fit method and phases obtained from Figure 3 by measuring the phase difference in a single zero-crossing.

REPLACE PLOTS WITH BLACK AND WHITE NOT GRAYSCALE, EDIT LINES

Amplitude is reconstructed according to the following equation.

$$A = \sqrt{B^2 + C^2}. \tag{13}$$

DO WE NEED TO SAY SOMETHING ABOUT VERIFYING AMPLITUDE RECONSTRUCTION?

The simulation has successfully been run for up to $1,000,000$ time steps, limited primarily by memory considerations due to the storage necessary for phase and amplitude reconstruction. At this long simulation duration, amplitudes and phases over six decades in frequency are obtained.

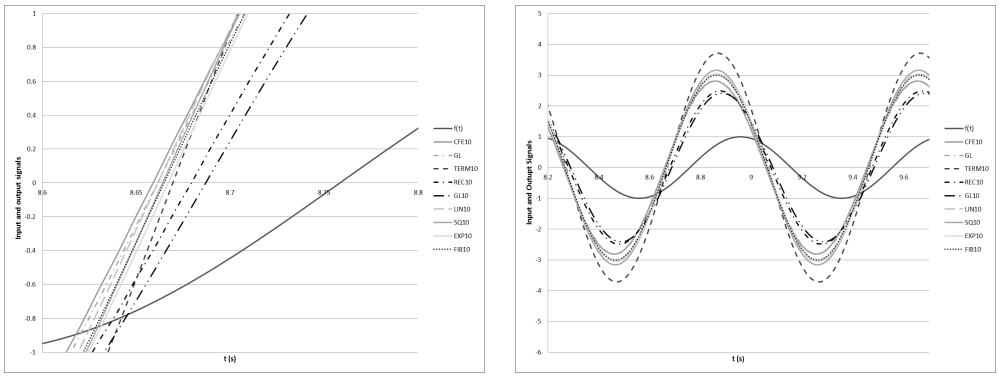

Figure 3: Zero crossings of a 0.5 order derivative of a sine wave input signal at long times relative to the period. From this the reconstructed phase can be verified using the knowledge that the frequency of this sine wave was 0.7924 Hz. Reconstructed and zero-crossing phases are shown in Table 4. In this plot, the duration was 10.0 s and there were 1000 time steps. There were 10 terms in the CFE and 10 bins in all binned Grunwald approximations. A variety of binning methods were used: truncated simple Grunwald (GL), linearly increasing bin sizes (LIN), bin sizes increasing as a square (SQ), exponentially increasing bin sizes (EXP), and bin sizes following a Fibbonacci rule (FIB).

Name	Reconstructed (degrees)	Zero-crossing (degrees)
CFE40	45.0	45 ± 2
GL	44.2	41 ± 2
GL26	23.0	32 ± 2
LIN26	46.1	41 ± 2
SQ26	44.4	41 ± 2
EXP26	42.1	41 ± 2
FIB26	43.0	41 ± 2

Table 1: Phases reconstructed through sinusoidal fits and phases approximated through single zero-crossings of a single frequency (0.7924 Hz) based upon Figure **??**.

5. Results

In a typical instance of the current state of the art algorithm, the continued fraction expansion (CFE), the expansion depends upon the last ten elements in the input signal history.**CITATION** To make a fair comparison between fractional derivative or integral algorithms, it we examine the average Grunwald algorithm with ten bins of input signal history, such that the history memory requirement is the same. Figure 5 contains a bode plot for a fractional derivative of order $\alpha = 0.5$ with ten history registers. Compared to the CFE10, the average Grunwald with an exponential binning structure (EXP10) has an approximately half a decade gain in constant-phase bandwidth. **OVERSAMPLING**

The performance of the average Grunwald algorithm improves dramatically with a small increase in the number of input signal history bins. The number of input signal history bins was limited by the computational abilities of the computer for both the CFE and the average Grunwald algorithms. For the CFE, the maximum input signal history depth was 40 steps into the past. For the average Grunwald, it was possible to run up to 26 bins ($> 10^7$ steps into the past for exponential binning). Figure 5 compares the 40 history memory registers of the CFE to the 26 history memory registers of the average Grunwald, for a variety of binning strategies. At low frequencies, we see that for both fractional derivatives and integrals there is a gain in the constant phase bandwidth of about a decade.

6. Conclusions

Acknowledgements

Thanks to Chad Bohannan for his contributions to the C++ code base and for his thoughts on computational efficiency. The authors also thank Saint Cloud State University for the use of computational resources.

References

[1] Luo Y, Chen YQ. Fractional Order Motion Controls. Chichester, West Sussex: John Wiley & Sons; 2013.

[2] Bohannan G. Analog fractional order controller in temperature and motor control applications. Journal of Vibration and Control 2008;14(9-10):1487–98.

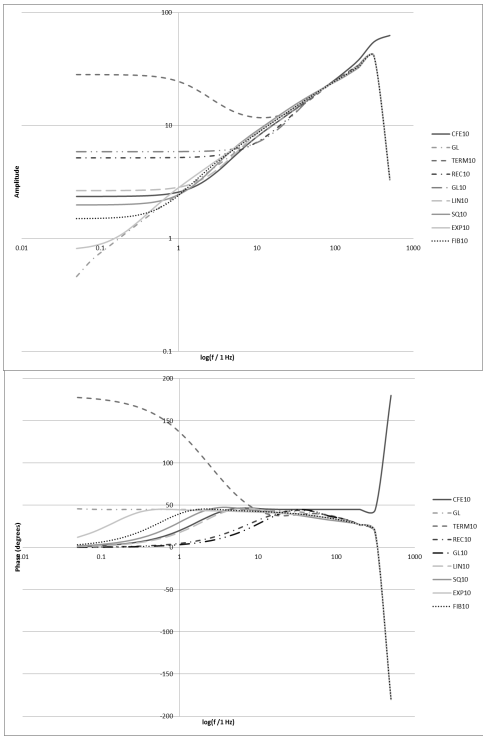

Figure 4: Amplitude and phase as a function of frequency for 10 registers of binned (SQ10, EXP10, FIB10) or unbinned (GL10, CFE10) input signal history. GL is the full Grunwald calculation, for the entire input signal history prior to that time.

[3] Monje CA, Chen Y, Vinagre BM, Xue D, Feliu V. Fractional-order Systems and Controls: Fundamentals and Applications, a monograph in the Advances in Industrial Control Series. Berlin: Springer; 2010.

[4] Chen YQ, Vinagre BM, Podlubny I. Continued fraction expansion approaches to discretizing fractional order derivatives – an expository review. Nonlinear dynamics 2004;38:155–70.

[5] Oldham K, Spanier J. The Fractional Calculus. New York: Academic Press; 1974.

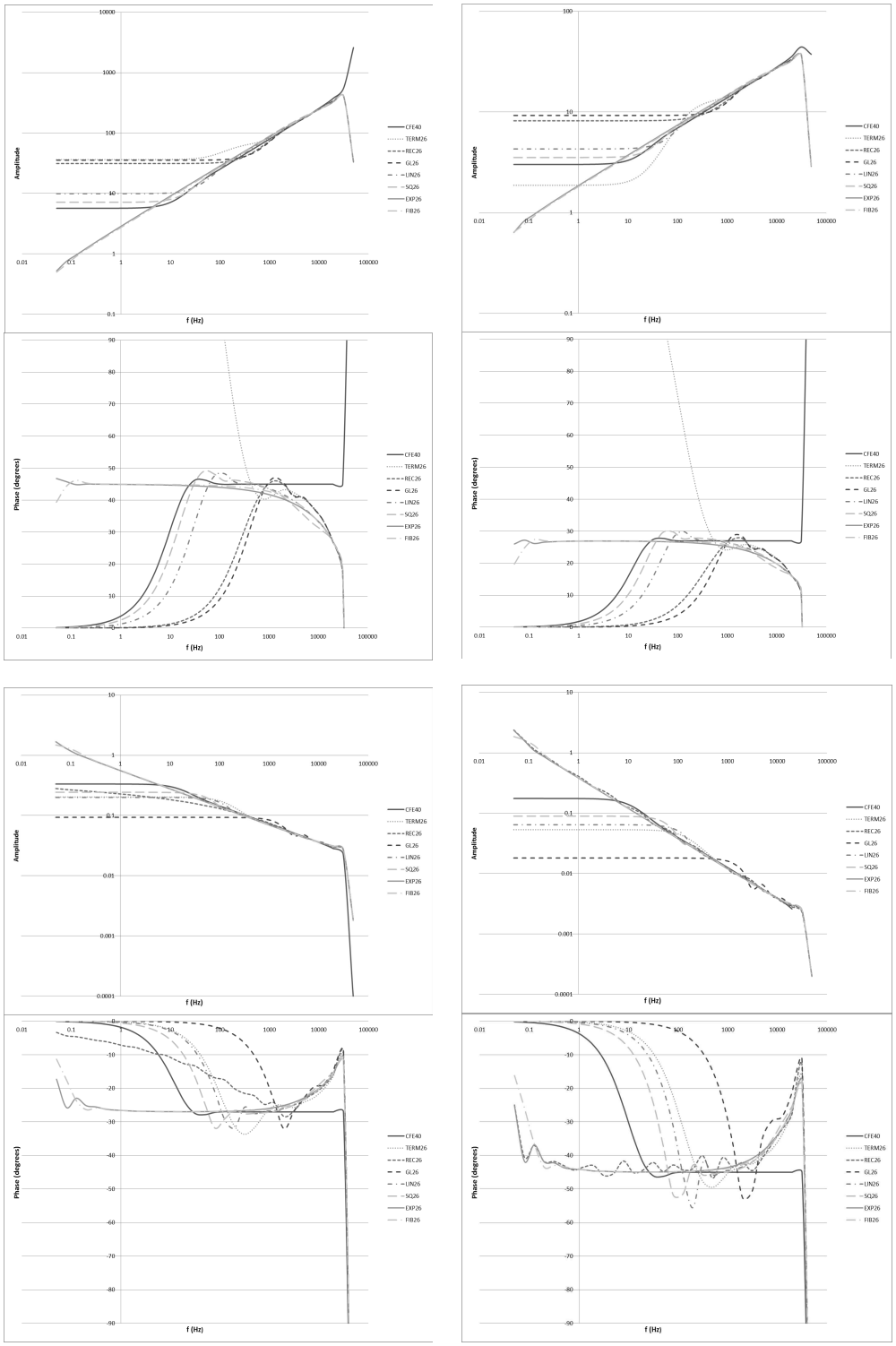

Figure 5: Bode plots for CFE with 40 input signal history registers and average Grunwald with 26 input signal history bins for $\alpha = 0.5, 0.3, -0.3,$, and -0.5.