AN EFFICIENT DEEP MEMORY ALGORITHM FOR COMPUTING FRACTIONAL ORDER OPERATORS

Steven Dorsher [1], Gary Bohannan [2], Chad Bohannan [3]

June 12, 2013

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.

Key Words and Phrases: digital fractors, memory kernel, fractional calculus, numerical methods, computational efficiency

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. [3] While wide bandwidth analog controllers have been successfully demonstrated [1], 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. [4]

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. [2].

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 the amplitude and phase response of these algorithms for a large and small number of bins in the partition.
- Section 4 will contain an analysis of the computational resources required for the larger versus the smaller number of memory registers stored.

2 Algorithm definitions

2.1 Partitioning the Grunwald history into averaged bins

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 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. Since the Grunwald weights decrease rapidly moving further back in time, the older and longer bins will sum together many small contributions, never exceeding the total from the more recent and shorter bins. 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.

2.1.1 Modified Grunwald

The Grunwald form of the fractional integral can be written

$$D_t^\alpha f(t) = \lim_{N\to\infty} \left(\frac{t}{N_t}\right)^{-\alpha} \sum_{j=0}^{N_t-1} w_j x_j \tag{2.1}$$

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

$$w_j = \frac{\Gamma(j-\alpha)}{\Gamma(j+1)\Gamma(-\alpha)}. \tag{2.2}$$

To include more distant history at low computational cost, we modify the Grunwald sum of Equation 2.1 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.

$$W_k = \sum_{j=p_{k-1}+1}^{p_k} w_j \tag{2.3}$$

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 2.1.2.

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

$$D_t^\alpha f(t) = (\Delta t)^{-\alpha} \sum_{k=0}^{N_b} \bar{W}_k X_k \tag{2.4}$$

where Δt is the interval between time samples.

2.1.2 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.

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 ($c_{k'} = c_k = b_k$) or increments the number of elements in the bin such that $c_{k'} = c_k + 1$ if the bin is not yet at capacity. Either way, the updated average of the value of the kth bin, $X_{k'}$, is given by

$$X'_k = \frac{c'_k - 1}{c'_k} X_k + \frac{1}{c'_k} X_{k-1}. \tag{2.5}$$

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,

$$\bar{W}_k = \frac{c_k W_k}{b_k}. \tag{2.6}$$

3 Amplitude and phase characteristics

4 Analysis of computational resources

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 /em per time step.

Type	Cost
Integer assign	Free
Floating point assign	Free
Negation	Free
Integer $+,-$	Free
Floating point $+,-$	1
Integer $*,$	1
Floating point $*,$	1
Floating point exponentiation	1
If (int clause)	Free
If (float clause)	1
While (int clause)	1
While (float clause)	2

Table 1: Number of processor steps required for various operations.

4.1 The average Grunwald algorithm

To count the number of processor steps required, the algorithms described in Section 2.1 are converted to pseudocode then the conversions listed in Table **??** are applied.

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,

$$N_t = \sum_{k=0}^{N_b} b_k. \tag{4.1}$$

Since zeroing (assignment) is computationally free, the bulk of the initialization process occurs during the computation of the weights. Since each individual Grunwald weight must be summed, this process scales as N_t. Equation **??** 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

$$w_j = \frac{j-1-\alpha}{j} w_{j-1}. \tag{4.2}$$

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 2.3, at which time the value of w_j may be safely overwritten with w_{j+1}.

Using these formulas, pseudocode for the weight-initialization algorithm is given in Algorithm 1.

Algorithm 1 Initialization of the binned weights. α is the order of the differintegral and b_k is the bin capacity of the kth bin.

```
for $0 \le k < N_b$ do
  $W_k \leftarrow 0.0$
end for
$j \leftarrow 1$
$k \leftarrow 0$
$w \leftarrow 1.0$
$p \leftarrow b_0$
while $k < Nb$ do
  $W_k \leftarrow W_k + w$
  $j \leftarrow j + 1$
  if $j > p$ then
    $k \leftarrow k + 1$
    $p \leftarrow p + b_k$
  end if
  $w \leftarrow \frac{j-1.0-\alpha}{j} w$
end while
```

Using Table 4.1, the number of processor steps required to implement the weight initialization algorithm is $5N_t + 1$.

Equation 2.5 may be used as the basis for an algorithm to update the binned average input signal history values when a new input signal value is read. 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.

Pseudocode for the bin-average updating algorithm is given in Algorithm 2.

Algorithm 2 Algorithm for updating the bin average values of the input signal history. X_k is the kth average binned value of the history. x_0 is the input signal value that has just been read into the digital fractor.

```
$k \leftarrow N_b$
while $k > 0$ do
  if $k = k_{filling}$ then
    $c_{filling} \leftarrow c_{filling} + 1$
    if $c_{filling} > b_k$ then
      $k_{filling} \leftarrow k_{filling} + 1$
    end if
    $k \leftarrow k + 1$
    $X_k \leftarrow \left(1.0 - \frac{1.0}{c_{filling}}\right) X_k - \frac{1.0}{c_{filling}} X_{k-1}$
  else if $k < k_{filling}$ then
    $X_k \leftarrow \left(1.0 - \frac{1.0}{b_k}\right) X_k - \frac{1.0}{b_k} X_{k-1}$
  end if
  $k \leftarrow k - 1$
end while
if $k_{filling} = 0$ then
  $c_{filling} \leftarrow c_{filling} + 1$
  if $c_{filling} > b_0$ then
    $k_{filling} \leftarrow 1$
    $c_{filling} \leftarrow 1$
  end if
  $X_k \leftarrow \left(1.0 - \frac{1.0}{c_{filling}}\right) X_0 - \frac{1.0}{c_{filling}} x_0$
else
  $X_k \leftarrow \left(1.0 - \frac{1.0}{b_0}\right) X_0 - \frac{1.0}{b_0} x_0$
end if
```

Table 4.1 yields $12N_b$ processor steps required to update the bin average values of the input signal history when a new data point is read.

Our algorithm for differintegration is based upon Equations **??** and 2.6. The algorithm is shown in Algorithm 3.

Based upon Table 4.1, the number of processor steps required to calculate the differintegral using the modified Grunwald approach presented in this

Algorithm 3 The algorithm for differintegration using the modified Grunwald method described in Section **??**

```
$k \leftarrow N_b$
while $k \geq 0$ do
    if $k = k_{filling}$ then
        $W_k \leftarrow \frac{c_{filling}}{b_k} W_k$
    end if
    $D_t^\alpha \leftarrow D_t^\alpha + W_k X_k$
end while
$D_t^\alpha \leftarrow (\Delta t)^{-\alpha} D_t^\alpha$
```

Stage	Processor steps required
Initialization of weights	$5N_t + 1$
Updating the average bin history values	$12N_b$
Summing the differintegral	$3N_b + 2$

Table 2: Processor steps required for the different stages of the average Grunwald algorithm.

paper is $3N_b + 2$.

The number of processor steps required for the three stages are summarized in Table 2. Memory requirements for the algorithms described in this section are shown in Table 4.1.

The two real-time operation stages, updating and differintegrating, together require $15N_b + 2$ processor steps. With 26 bins ($N_b = 26$), 392 processor steps are required for each time-step of a fractional order differentiation or integration. $3N_b + 6 = 84$ memory registers are required for 26 bins of history.

Initialization, on the other hand, is a very resource-consuming process. For an exponentially scaled binning structure with a scale factor of two, $N_t = 2^{26} - 1 = 67,108,863$ for 26 bins. Initialization therefore requires 3.36×10^8 processor steps, a calculation perhaps best achieved by a computer in a one-time operation.

Variable	Definition	Size	Stages
b_k	Bin capacity	N_b	I, U, D
W_k	Binned weights	N_b	I, U, D
X_k	Binned history	N_b	I, U, D
x_0	Current input data	1	U
D_t^α	Output differintegral	1	I, D
α	Order of differintegral	1	I, D
N_b	Number of bins	1	I,U,D
N_t	Number of time steps stored	1	I
Δt	Time step	1	D
w_j	Grunwald weight, temp variable	1	I
$k_{filling}$	Currently filling bin	1	U, D
$c_{filling}$	Occupancy of filling bin	1	U, D
	Total w/o initialization	$3N_b + 6$	
	Total	$3N_b + 9$	

Table 3: Accounting for memory in the Average-Shift algorithm. I = Initialization, U=Updating, D = Differintegration

5 Conclusions

Acknowledgements

The author thanks his institution for the support, under Grant No ...

References

[1] G. Bohannan, Analog Fractional Order Controller in Temperature and Motor Control Applications. *Journal of Vibration and Control* **14(9-10)** (2008), 1487-1498.

[2] Y. Chen, B. Vinagre, I. Podlubny, Continued fraction expansion approaches to discretizing fractional order derivatives – an expository review. *Nonlinear dynamics.* **38** (2004), 155-170.

[3] Y. Luo, Y. Chen, *Fractional Order Motion Controls.* John Wiley & Sons, Chichester, West Sussex (2013).

[4] C. Monje, Y. Chen, B. Vinagre, D. Xue, V. Feliu, *Fractional-order Systems and Controls: Fundamentals and Applications, a monograph in the Advances in Industrial Control Series.* Springer, Berlin (2010).

[5] K. Oldham, J. Spanier, *The Fractional Calculus.* Academic Press, New York (1974).

[1] *Department of Physics*
St Cloud State University
mailing address: 3152 Blackheath Drive
Saint Cloud, MN 56301, USA

email: sdorsher@gmail.com

Recieved: ???

[2] *Department of Physics*
Saint Cloud State University
720 4th Avenue South
Saint Cloud, MN 56301, USA

email: gwbohannan@stcloudstate.edu
[3] *Chad Bohannan*
email: chad.bohannan@gmail.com