Specifications for a fractional order filter with deep history

January 4, 2013

1 Background

Due to memory limitations in the physical controller, the existing algorithm for fractional order digital filters stores and uses a small number of data points into the recent past as input to the computation. However, this limited amount of data does not account for the true dependence of a fractional order derivative or integral upon the full history of the input signal. To account for this, we are developing a new algorithm that samples the long term history of the signal using the same small number of memory registers to store the data. In this note I will address the question of how to write the fractional order derivative, D^α, that appears in the fractional order differential equations governing the system.

1.1 The Grunwald integral

We choose to calculate the fractional order integral D^α using the Grunwald method. In its simplest form it can be written

$$D^\alpha = (\Delta x)^{-\alpha} \sum_{j=1}^{N-1} w_j f\left(x - \frac{j\Delta x}{N}\right) \tag{1}$$

where

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

and w_j can be calculated recursively from the relation

$$\frac{\Gamma(j-\alpha)}{\Gamma(j+1)} = \frac{j-1-\alpha}{j}\frac{\Gamma(j-1-\alpha)}{\Gamma(j)} \tag{3}$$

and $\Gamma(1) = 1$.

The Grunwald diffintegral can be expressed in a recursive form that is better adapted for handling the addition and multiplication of small numbers without the loss of precision. This recursive form can be written

$$D^{\alpha} = \frac{N^{\alpha}}{(\Delta x)^{\alpha}}[[[\cdots[[f_{N-1}r_{N-1} + f_{N-2}]r_{N-2} + f_{N-3}] \cdot]r_2 + f_1]r_1 + f_0] \quad (4)$$

where $f_i = f(x - \frac{i\Delta x}{N})$ and the recursive weights r_j are given by

$$r_j = \frac{N - \alpha - j}{j} \quad (5)$$

for $j = 1$ to $N-1$ and $r_0 = 1$. These weights can be computed recursively using the relation given in Equation 3.

In the current version of the code, I calculate the Grunwald integral or derivative (negative or positive α) using the full history of the input signal for comparison purposes. This is not the final product, since we are interested in less computationally intensive methods, but it has been helpful both in debugging and as a science benchmark. In my current code, the object FullGrundwald implements Equation 4 and Equation 1 is not implemented in its full history form anywhere in the code.

2 Adding deep history

Equations 1 and 4 are sufficient if the computing power is available to handle the full history of the input signal. Since that is in general not expected to be the case, below I explain the modifications we have made to these algorithms to handle a small amount of available memory in the context of a fractional order integral that requires a deep history. I'll explain the algorithms we've developed based on both Equation 1 and 4, though at this point we may have to chose a direction. Equation 1 has the advantage of simplicity. Equation 4 has the advantage that it is said to behave better numerically, though I have not yet seen this to be true in practice.

2.1 Average-shifting of data points

Integrating from present to past, it is possible to simply truncate either integral after some number of data points. However, this loses the contribution from the deep history. We label the full history (which will not be stored) extending from the present time into the past. As time is advanced to a new data point, another data point is added to the full history at the zero point, lengthening it by one up to a maximum of length of N and shifting the older data points to larger time registers.

To solve the problem of storing deep history in a small number of registers, we decided to split the full history into some number of bins using a partition labelled by a set of array indices p_k. For the kth bin, p_k gives the maximum full history register to be included in the average. The minimum full history register included in the kth bin is given by $p_{k-1}+1$ for $k > 0$ and zero for $k = 0$.

Each time step shifts one data point along, displacing one data point out of each bin. Along the way, already full bins will need to be updated to reflect new averages. At the end of the line, the final bin may already be full and a new bin may be begun or it may be in the process of filling. If the kth bin initially contains c_k elements, adding another element either leaves c_k unchanged ($c_k\prime = c_k$) or increments the number of elements in the bin such that $c_k\prime = c_k + 1$ if the bin is filling. Then the updated average value of the kth bin, $f_k\prime$, is given by

$$f_k\prime = \frac{c_k\prime - 1}{c_k\prime} f_k + \frac{1}{c_k\prime} f_{k-1} \tag{6}$$

where f_k is the initial average value of the input signal at the kth bin. If the bin becomes full. Whether a bin is full prior to shifting can be determined by comparing c_k to $p_k - p_{k-1} + 1$.

Equation 6 and the associated shifting and binning algorithm is currently implemented in the object AvgShiftRegister in my code.

2.2 The straightforward Grunwald sum

To extend the Grunwald sum of Equation 1, we simply sum the coefficients within each bin. We define

$$W_k = \sum_{j=p_k}^{p_{k+1}} w_j \tag{7}$$

$$D^\alpha = (\Delta x)^{-\alpha} \sum_k \bar{W}_k f_k \tag{8}$$

where f_k represents the average value from the kth bin and $\bar{W}_k$ is an average accounting for the degree to which the bin is currently filled of the summed weight associated with the kth bin. It is not very clear in what kind of average should be used for $\bar{W}_k$, since W_k is built of both sums and products. I have not had much luck obtaining an exact analytic expression for it so far. Between an arithmetic mean and a geometric mean, we choose an arithmetic mean, since a partial sum of weights should be less than the total and the weights will in general be less than one.

$$\bar{W}_k = \frac{c_k W_k}{p_k - p_{k-1} + 1} \tag{9}$$

In my code, Equation 8 with $\bar{W}_k$ replaced by W_k is implemented in a terribly out of date object called Dalpha. Averaging for filling is not implemented at all. Despite the fact this code is out of date and that this may be a less numerically precise approach, this approach is very straightforward, and I am not sure whether or not we wish to retain this option.

2.3 The recursive Grunwald diffintegral

In Equation 4, assume the ith through jth terms in the kth bin share the same average value, $f_i = f_{i+1} = \cdots = f_{j-1} = f_j$. Let G_k be the product of all previous terms prior to the kth bin starting from the $N-1$ end of the product. Then

$$G_{k+1} = [[[[G_k + f_j]r_j + f_{j-1}]r_{j-1} + \cdots]r_{i+1} + f_i]r_i \tag{10}$$

which can be expressed more simply as

$$G_{k+1} = \bar{\beta}_k G_k + \bar{\gamma}_k f_k \tag{11}$$

$$\beta_k = \prod_{j=p_k}^{p_{k+1}} r_j \tag{12}$$

$$\gamma_k = [[[[r_j + 1]r_{j-1} + 1] + \cdots]r_{i+1} + 1]r_i \tag{13}$$

where $\bar{\beta}_k$ and $\bar{\gamma}_k$ are averages of these weights over the kth bin to account for the proportion of the bin that has been filled. Since β is simply a product of weights, $\bar{\beta}_k$ is a geometric mean over the number of elements in the bin at the present time.

$$\bar{\beta} = \beta^{c_k/(p_k - p_{k-1}+1)} \tag{14}$$

It is less clear that the sum and product combination of γ should be averaged using a geometric mean; however, we lack a better tool at this time. Therefore,

$$\bar{\gamma} = \gamma^{c_k/(p_k - p_{k-1}+1)} \tag{15}$$

Equation 11 and the associated averaging algorithm are implemented in the object Grundwald, which is up to date.

3 Phase and amplitude plot

Phases and amplitudes are reconstructed by fitting the input signal to obtain its sine and cosine components using an algorithm modified by Professor Bohannan from Numerical Recipies. I simply imported Nrutil.cpp from his other code. However, through testing I found that the sine and cosine components were assigned in the opposite manner of how they were labelled in the comments. I certainly encourage independent testing of this matter!

I defined the phase to be with reference to the sine component of the oscillation. That is,

$$C\sin(\omega t + \phi) = A\sin(\omega t) + B\cos(\omega t) \tag{16}$$

From this, it is easy to see that

$$C = \sqrt{A^2 + B^2} \tag{17}$$

$$\phi = \arctan\left(\frac{B}{A}\right) \tag{18}$$

where the degeneracy in phase with arctangent can be lifted using the sign of the cosine or sine. In C++ this is a nonissue since the function atan2 can be used instead. In the current version of the code, I have implemented this in an equivalent form,

$$\phi = \arctan\left(\frac{A}{B}\right) + \frac{\pi}{2} \tag{19}$$

Phase and amplitude reconstruction are done in MAIN. In MAIN, genAmpPhaseGrundwald produces the frequency versus phase and amplitue plot data.

4 Partitions

The partitions we have tried so far include

- A "linear" partition with $p_j = j$. For $j_{max} = N - 1$, this partition should be equivalent to a full Grunwald integral. With $j_{max} = 10$, it shows a sharp peak in phase at middle frequencies unlike any of the other partitions listed; however, it also comes closest to the theoretically predicted phase.
- A "squared" partition with $p_j = j^2$. This has $2j - 1$ in the jth bin.
- A "power-law per bin" partition with p_j increasing by steps of 2^j with each j.
- A "power-law" partition with $p_j = 2^j$. This most closely approaches the high frequency limit in phase, where the function is not well behaved anyway. It is marginally worst at low frequencies. If we could improve the high frequency limit somehow, this one is promising.

Partitions are typically implemented within objects such as Grundwald, FullGrundwald, or Dalpha.