VERSITA

Fractional Calculus & Applied Analysis
An International Journal for Theory and Applications

VOLUME 16, NUMBER 3 (2013)
(Print) ISSN 1311-0454
(Electronic) ISSN 1314-2224

RESEARCH PAPER

FRACTIONAL INTEGRATION TOOLBOX

Toma M. Marinov [1], **Nelson Ramirez** [2], **Fidel Santamaria** [3,*]

Abstract

The problems formulated in the fractional calculus framework often require numerical fractional integration/differentiation of large data sets. Several existing fractional control toolboxes are capable of performing fractional calculus operations, however, none of them can efficiently perform numerical integration on multiple large data sequences. We developed a Fractional Integration Toolbox (FIT), which efficiently performs fractional numerical integration/differentiation of the Riemann-Liouville type on large data sequences. The toolbox allows parallelization and is designed to be deployed on both CPU and GPU platforms.

MSC 2010: Primary 26A33; Secondary 33F05, 60G22, 65D05, 65D25

Key Words and Phrases: fractional calculus, Riemann-Liouville fractional integral, Riemann-Liouville fractional derivative, numerical quadrature, fractional reaction-diffusion equation

1. Introduction

In the last several decades many processes in biology, pharmacology, chemistry, physics, engineering and finances have been formulated in the framework of fractional calculus [1], [2], [7], [8], [18]-[20].

Systems involving fractional processes exhibit residual memory and their fractional order is interpreted ([12], [13], [15]) as a measure of the memory strength, which depending on the fractional order α, positions them somewhere between complete memory and Markovian systems [13].

© 2013 Diogenes Co., Sofia
pp. 670–681 , DOI: 10.2478/s13540-013-0042-7

For practical applications such as fractional diffusion-reaction or other fractional dynamical systems simulations, it is sometimes necessary to calculate the fractional Riemann-Liouville (RL) integral of a function $f(\tau)$, given by:

$$\mathbf{I}_t^{\alpha} f(t) = \frac{1}{\Gamma(\alpha)} \int_0^t (t-\tau)^{\alpha-1} f(\tau) d\tau, \tag{1.1}$$

where $\Gamma(\alpha)$ is Euler's gamma function and α is positive real number.

For each spatial point x_i at each time step τ_j of the simulation, one needs to estimate numerically eq.(1.1) given the sequence $f(\tau_1)$, $f(\tau_2)$,..., $f(\tau_j)$ obtained from the previous steps of the simulation. Even for a relatively large spatial discretization step in a one dimensional fractional diffusion-reaction simulation, the numerical evaluation of eq.(1.1) becomes computationally prohibitive in terms of speed and memory requirements due to the increasing size of the input matrix. This problem becomes even more severe in the case of a simulation of multiple fractional diffusion-reaction equations describing the interactions in a biochemical reaction or in the case of multidimensional simulations. Furthermore, achieving higher accuracy of the numerical quadrature of eq.(1.1) involves multistep methods, which require even more computational resources.

While there are several fractional control toolboxes (Crone [11], Fomcon [22], Ninteger [23]) capable of performing fractional calculus operations, none of them is designed to fractionally integrate simultaneously multiple sequences of large length.

Here we report the development of a Fractional Integration Toolbox (available for download at `www.cbi.utsa.edu/projects/faculty`) which employs an efficient algorithm [5] and allows the user to choose computational platforms (CPU, GPU) and parallelization (openMP). Our Benchmark test shows that integration in High Performance platforms can achieve speed improvement of two orders of magnitude. This enables applications which require simultaneous fractional integration of multiple data sources in real time. For completeness we have included an efficient numerical routine [14] for fractional derivatives of the RL type.

The toolbox is generic. It can be used both in a standalone and in a simulation mode. The standalone mode fractionally integrates an arbitrary set of data. The simulation mode is designed for the case when repeated fractional integration is performed on sets of data, which increase one term at a time. In that case additional performance optimization is possible. This makes FIT a beneficial tool not only in the case of fractional reaction-diffusion studies, but also to other fields formulated in terms of fractional dynamics.

2. Methods

For the fractional integration we consider only fractional integrals of the RL type and employ the numerical algorithm provided in [5]. This algorithm is valid for any $\alpha > 0$. On the interval $[0, T]$ with equally spaced points $t_n = nh : n = 0, 1..., N$, where $h = T/N$, the integral's (eq.(1.1)) leading order approximation is given by

$$I^\alpha f_N(h) = \frac{h^\alpha}{\Gamma(2+\alpha)} \sum_{n=0}^{N} c_{n,N} f_n, \tag{2.1}$$

$$I^\alpha f(T) = I^\alpha f_N(h) + O(h^2),$$

where $f_n = f(t_n)$ and $c_{n,N}$ are the quadrature weight coefficients

$$c_{n,N} = \begin{cases} (1+\alpha)N^\alpha - N^{1+\alpha} + (N-1)^{1+\alpha} & n = 0; \\ (N-n+1)^{1+\alpha} - 2(N-n)^{1+\alpha} + (N-n-1)^{1+\alpha} & 1 < n < N; \\ 1 & n = N; \end{cases} \tag{2.2}$$

derived from a product trapezoidal rule.

Further refinement is possible by using the Richardson extrapolation:

$$I^\alpha f_v^u(T) = (I^\alpha f_{v-1}^{u-1}(T) - 2^{r_{u-1}} I^\alpha f_v^{u-1}(T))/(1 - 2^{r_{u-1}}), \tag{2.3}$$
$$I^\alpha f(T) = I^\alpha f_v^u(T) + O(h^{r_u}),$$

where

$$\begin{aligned} &I^\alpha f_0^0(T) = I^\alpha f_N(h) \\ &I^\alpha f_1^0(T) = I^\alpha f_N\left(\frac{h}{2}\right) \\ &. \\ &. \\ &. \end{aligned} \tag{2.4}$$

and the exponents $r_0, r_1, r_2, ...$ have to be ordered in a monotonic sequence, i.e.

$$\begin{aligned} &r_0 = 2,\ r_1 = 2+\alpha,\ r_2 = 3,\ r_3 = 3+\alpha, ... \ \text{ for } \ 0 < \alpha < 1, \\ &r_0 = 2,\ r_1 = 3,\ r_2 = 2+\alpha,\ r_3 = 4, ... \ \text{ for } \ 1 < \alpha < 2, \\ &. \\ &. \\ &. \end{aligned} \tag{2.5}$$

Thus for a leading order and a first correction approximation:

$$I^{\alpha} f_1^1(T) = \frac{I^{\alpha} f_0^0(T) - 2^2 I^{\alpha} f_1^0(T)}{1 - 2^2}, \tag{2.6}$$
$$I^{\alpha} f(T) = I^{\alpha} f_1^1(T) + O(h^{r_1}).$$

The discretization of the function $f(\tau)$ depends on the chosen time step h. However, the calculation of the first correction requires the evaluation of $f(\tau)$ at the midpoints of each interval, i.e. at a time step $h/2$. Since the leading order is the exact solution of eq.(1.1), provided that $f(\tau)$ is linearly interpolated in each of the intervals, there will be no gain in accuracy that could come from the calculation of the first correction based on resampled data from piecewise linear interpolation. In order to increase accuracy we use spline interpolation.

The FIT allows the user to choose between two types of spline interpolation, i.e. regular cubic splines and piecewise cubic Hermite interpolation. Cubic splines interpolate smooth functions very well due to the fact that they are constructed by piecewise smooth functions, matched at the nodes to preserve smoothness (matching both first and second derivatives). The piecewise cubic Hermite interpolation, on the other hand, is well suited to handle noisy data. It has minimal overshoots and undershoots and in general it follows better the behavior of piecewise continuous functions.

The data flow of the FIT is shown on Figure **1**. The input for the toolbox includes the fractional order α, the discretization time step h, the integration limit t_m, as well as a $m \times n$ data matrix. Each of the columns of the data matrix represents a sequence of discrete values of the integrand f (f_{1i}, f_{2i},...,f_{mi}), which is to be numerically integrated/differentiated. As a result, the toolbox returns n integral/derivative values-one for each sequence of data points.

The properties of the function to be integrated, such as size, stiffness or noise require the use of different integration algorithms. In order to provide a generic toolbox we have implemented multiple algorithms that can be chosen at the time of setup:

- Linear- leading order (LO)
- Cubic spline- leading order+correction
- Hermite spline- leading order+ correction

All of the integration algorithms are based on [5], however their implementations vary. The leading order approximation (eq.(2.1)) does not require interpolation and is implemented efficiently in `MATLAB` (Natick, MA) as a dot product of the vector containing the integration weights (eq.(2.2))

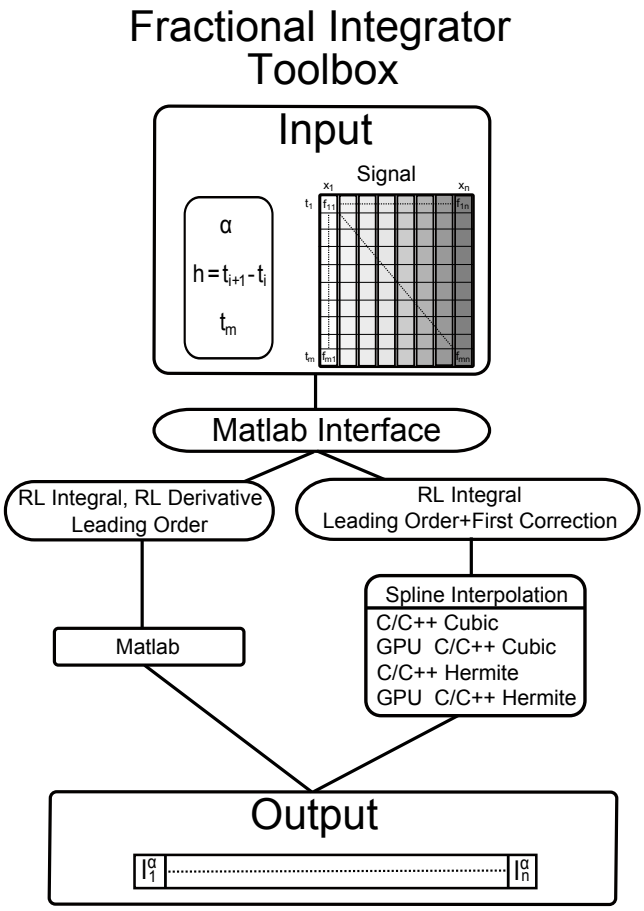

FIGURE 1. Block diagram of the Fractional Integral Toolbox

and the input matrix containing all the time signals. Since MATLAB supports implicit multithreading, the computation takes advantage of multicore computer architectures automatically. Additional acceleration is achieved through the use of highly efficient MATLAB vectorized notation.

The leading order+first correction approximation (eq.(2.6)) requires sampling at a time step $h/2$, i.e. interpolation is necessary. The FIT provides two separate interpolation methods- a regular cubic spline as well as Hermite piecewise cubic interpolation. For the setup of the cubic spline coefficients and the numerical solution of the resulting tridiagonal system we employ [3], [4], [21].

In the case of the Hermite piecewise cubic interpolation we obtain monotonic cubic polynomials by employing the algorithm described in [10]. Both the regular cubic and the Hermite piecewise cubic interpolation algorithms are implemented in C++ and are integrated with `MATLAB` via a MEX interface with OpenMP multithreading. For better performance the interpolated data needed for the calculation of the correction is not stored, but is generated and used dynamically. Additional simplification and improvement of efficiency is possible by taking advantage of the fixed time step in the spline midpoint evaluation. The FIT provides two more fractional integration options- a hybrid GPU+CPU Cubic and a hybrid GPU+CPU Hermite integration routines. In both cases, the interpolated data is generated on a CPU (C++ Cubic or C++ Hermite, OpenMP multithreading) and passed to the GPU, where both $I^\alpha f_0^0(T)$ and $I^\alpha f_1^0(T)$ in eq.(2.6) are computed as dot products, similarly to the leading order case in `MATLAB`.

For the fractional derivative we consider only the fractional RL type and employ the L1 numerical algorithm provided in [14]. The fractional RL derivative of a function $y = f(t)$ is given by:

$$\frac{d^\alpha f}{dt^\alpha} \approx \frac{h^{-\alpha}}{\Gamma(2-\alpha)} \left[\frac{(1-\alpha)f(0)}{N^\alpha} + \sum_{n=1}^{N} a_n (f(t_{n+1}) - f(t_n)) \right], \quad (2.7)$$

where $\Gamma(\alpha)$ is Euler's gamma function, α is a fractional order and the weights a_n for $n = 1, ..., N$ are given by

$$a_n = (N-n+1)^{1-\alpha} - (N-n)^{1-\alpha}. \quad (2.8)$$

Analogously to the leading order RL integration, the sum is implemented efficiently in `MATLAB` as a dot product.

Frequently, fractional integration is necessary in the context of a simulation. In that case additional performance improvements are possible. One such possible improvement is reusing the calculated weights (eq.(2.2) and eq.(2.8)). This is based on the observation that as N increases by one at each time step of the simulation, at the timestep $N+1$ we need to calculate only two new values, i.e. $c_{0,N+1}$ and $c_{1,N+1}$ and reuse $c_{n,N}$ for $n = 2, ..., N$.

Another approach in the case of RL fractional integration is the nested mesh technique in [9] and [6]. It is based on the scaling property of the fractional integral, i.e. for any $w > 0$ and $p \in \mathbb{N}$

$$\mathbf{I}_t^\alpha f(w^p t) = \frac{w^{p\alpha}}{\Gamma(\alpha)} \int_0^t (t-\tau)^{\alpha-1} f(w^p \tau) d\tau. \quad (2.9)$$

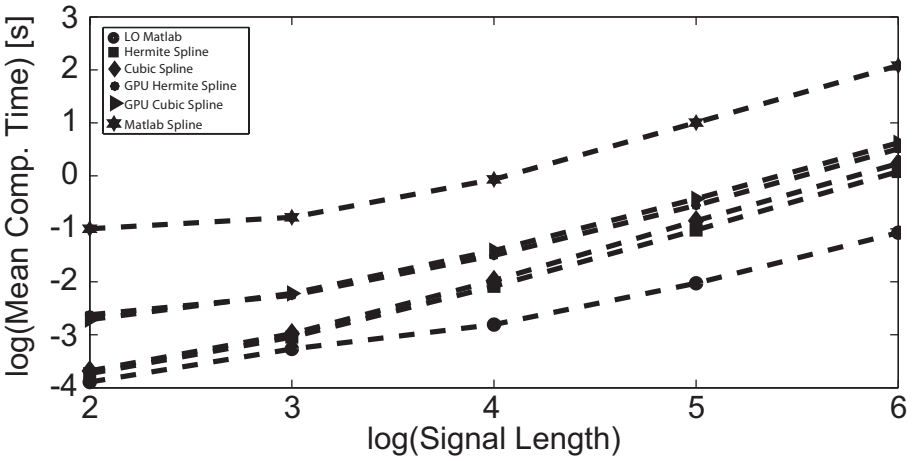

FIGURE 2. Fractional integration benchmark test: Mean computational time in seconds as a function of the signal length

For a fixed $T_0 > 0$ and a given $w > 0$, we can decompose the interval $[0, T]$ as

$$[0,T] = [0, T - w^m T_0] \cup [T - w^m T_0, T - w^{m-1} T_0] \cup ... \cup [T - wT_0, T - T_0] \cup [T - T_0, T]. \tag{2.10}$$

As a result, the RL fractional integral can be written as

$$\mathbf{I}^{\alpha}_{[0,T]} f(t) = \mathbf{I}^{\alpha}_{[T-T_0,T]} f(t) + \sum_{i=0}^{m-1} w^{i\alpha} \mathbf{I}^{\alpha}_{[T-w^{i+1}T_0, T-w^i T_0]} f(w^i t) + w^{m\alpha} \mathbf{I}^{\alpha}_{[0, T-w^m T_0]} f(w^m t). \tag{2.11}$$

The weights for calculating the integrals in eq.(2.11) are given by [6].

3. Results and Discussion

The results of the performance tests are given on Figure **2**, Figure **3** and Figure **4**. All the tests were performed on a GPU enabled node with 12 CPU cores (2 x Intel Xeon X5680, 3.33GHz processors), 195GB total RAM and a Tesla M2070 GPU with 4GB RAM. Figure **2** shows the mean computational time in seconds as a function of the sequence (signal) length for various implementations in the standalone mode. The tests were performed on one hundred signals with lengths ranging from 10, 10^2,..., 10^6, i.e. input matrices with size 100×10, 100×100,..., 100×10^6, correspondingly.

The difference between the best performing implementation- linear-leading order (`MATLAB`, implicit multithreading) and the cubic spline-leading order+correction(`MATLAB`, built in cubic spline routine) is three orders of magnitude for signal lengths $\sim 10^6$. The increase in computational efficiency comes from the custom implementation of the algorithm [5], the interpolation, as well as the OpenMP multithreading. The difference between the linear-leading order (`MATLAB`, implicit multithreading) and the more accurate higher order C++ Cubic spline implementation(MEX interface, openMP) is an order of magnitude for signal lengths $\sim 10^6$. This makes the employment of accurate higher order methods plausible in the case of fractional dynamical systems simulations.

For any type of numerical integration algorithm it is imperative to quantify the numerical error. In this case we use test functions for which eq.(1.1) results in an explicit expression ($f(t) = t$, $f(t) = t^2$, $f(t) = t^3$ and $f(t) = \sqrt{t}$). We evaluate the relative error given by

$$RE = \left| \frac{I^{\alpha}_{comp} f(t) - I^{\alpha}_{analytic} f(t)}{I^{\alpha}_{analytic} f(t)} \right|. \tag{3.1}$$

The results are shown in Table 1.

	t	t^2	t^3	$\sqrt{t}$
LO	$8.96 * 10^{-16}$	$3.12 * 10^{-9}$	$7.27 * 10^{-9}$	$1.32 * 10^{-7}$
Cubic	-	$8.55 * 10^{-12}$	$2.98 * 10^{-11}$	$7.13 * 10^{-8}$
Hermite	-	$1.29 * 10^{-11}$	$4.49 * 10^{-11}$	$7.93 * 10^{-8}$
GPUCubic	-	$8.55 * 10^{-12}$	$2.98 * 10^{-11}$	$7.13 * 10^{-8}$
GPUHermite	-	$1.29 * 10^{-11}$	$4.49 * 10^{-11}$	$7.93 * 10^{-8}$
RL Derivative	$2.74 * 10^{-15}$	$3.11 * 10^{-7}$	$7.76 * 10^{-7}$	$1.32 * 10^{-7}$

TABLE 1. Relative Error for 10^4 signal points, $\alpha = 0.5$

The higher order methods outperform the linear leading order implementation in terms of accuracy, justifying the use of spline interpolation for resampling of the interpolation data. Additionally, the integration using splines in High Performance mode is two orders of magnitude faster than the integration using the `MATLAB` built in spline routine. Thus the use of spline interpolation is feasible for large data sets, which makes sufficiently large real time simulations possible.

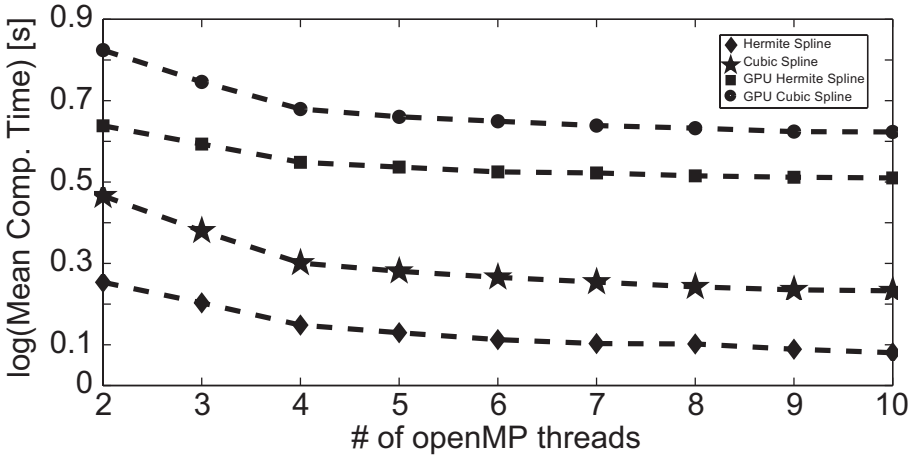

FIGURE 3. Parallelization benchmark test: Mean computational time in seconds as a function of number of threads

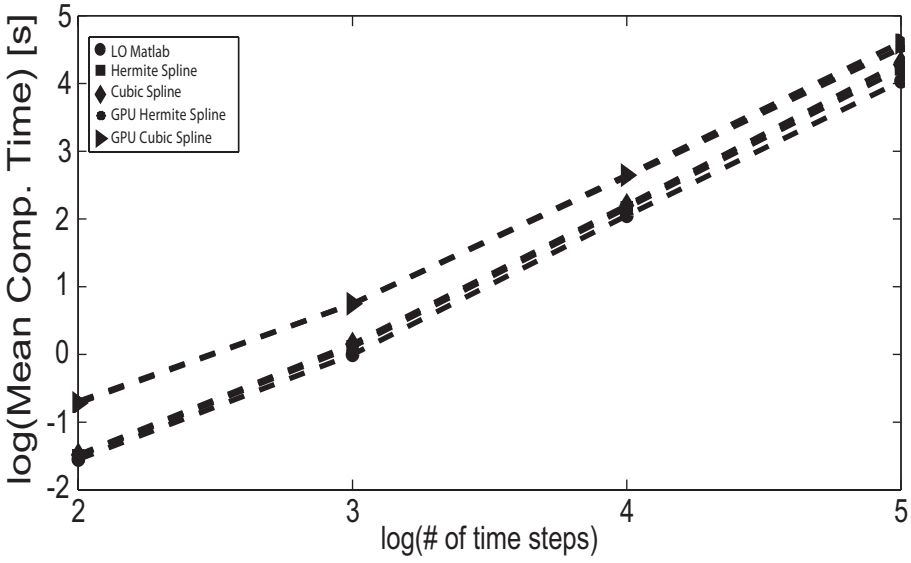

FIGURE 4. Diffusion simulation benchmark test: Mean computational time in seconds of the diffusion simulations in a standalone mode as a function of the number of time steps

Since the FIT supports OpenMP multithreading, it is important to test the scalability of the toolbox. Figure **3** shows the mean performance speed in a standalone mode for the different implementations as a function of the number of openMP threads for 100 input signals of length 10^6 for the four different implementations. The results (Figure **3**) demonstrate a two-fold performance improvement due to multithreading and thus justify the parallelization paradigm employed in the development of the FIT. Depending on available machine configuration (number of processor cores, hyperthreading capability), the user can adjust the number of openMP threads in order to optimize performance.

Another benchmark test we performed is a one dimensional fractional diffusion simulation. Fractional diffusion simulations provide a computational framework for studying anomalous diffusion. Anomalous diffusion plays a significant role in biochemical processes, including molecular signaling in neurons [16], [17]. Figure **4** shows the average computational time for all different implementations in FIT in a standalone mode for a system with 200 spatial points and 10^2, 10^3, 10^4 and 10^5 time steps, respectively. At each time step the FIT is called to fractionally integrate the time signal for each space point. All the implementations were numerically stable.

Additional comparison between the standalone (LO) mode and the simulation mode (nested mesh and weight optimization) in a one dimensional fractional diffusion simulation for a system with 200 spatial points and 10^5 time steps shows performance improvement by a factor of two (not shown).

The results show that for all the implementations in the FIT, one can perform fractional diffusion simulations in real time. Furthermore, due to the independence of time signals in the input matrix, the FIT can be used in any fractional dynamics simulation with no (or minimal) modifications.

Acknowledgements

This work received computational support from the Computational System Biology Core (`www.cbi.utsa.edu`), funded by Grant G12MD007591 from the National Institute on Minority Health and Health Disparities. This work was supported by Grants EF-1137897 and HDR-0932339 from the National Science Foundation.

References

[1] T.J. Anastasio, The fractional-order dynamics of brainstem vestibulo-oculomotor neurons. *Biol. Cybern.* **72** (1994), 69–79.

[2] K. Assaleh, W.M. Ahmad, Modeling of speech signals using fractional calculus. In: *9th International Symposium on Signal Processing and Its Applications, ISSPA 2007, 12-15 Feb.* (2007), 1–4.
[3] C. deBoor, *Practical Guide to Splines*. Springer, New York (2001).
[4] S.D. Conte, C. deBoor, *Elementary Numerical Analysis*. McGraw-Hill, New York (2002).
[5] K. Diethelm, N.J. Ford, A.D. Freed, Yu. Luchko, Algorithms for the fractional calculus: A selection of numerical methods. *Comput. Methods Appl. Mech. Engrg.* **194** (2005), 743–773.
[6] K. Diethelm, *The Analysis of Fractional Differential Equations*. Springer, Berlin (2010).
[7] A. Dokoumetzidis, R. Magin, P. Macheras, Fractional kinetics in multicompartmental systems. *J. Pharmacokinet. Pharmacodyn.* **37**, No 5 (2010), 507–524.
[8] J.F. Douglas, Some applications of fractional calculus to polymer science. *Advances in Chemical Physics* **102** (2007), 121–191.
[9] N.J. Ford, A.C. Simpson, The numerical solution of fractional differential equations: speed versus accuracy. *Numerical Algorithms* **26** (2001), 333–346.
[10] F.N. Fritsch, R.E. Carlson, Monotone piecewise cubic interpolation. *SIAM J. Numer. Anal* **17**, No 2 (1980).
[11] P. Melchior, B. Orsoni, O. Lavialle and A. Oustaloup, The CRONE toolbox for Matlab: Fractional Path Planning Design in Robotics; http://www.ims-bordeaux.fr/CRONE/toolbox.
[12] M. Monsrefi-Torbati, J.K. Hammond, Physical and geometrical interpretation of fractional operators. *J. Franklin Inst.* **335B**, No 6 (1998), 1077–1086.
[13] R.R. Nigmatullin, Fractional integral and its physical interpretation. *Teoret. Mat. Fiz.* **90**, No 3 (1992), 354–368.
[14] K.B. Oldham, J. Spanier, *The Fractional Calculus: Theory and Applications of Differentiation and Integration to Arbitrary Order*. Dover, Mineola, New York (2006).
[15] I. Podlubny, Geometric and physical interpretation of fractional integration and fractional differentiation. *Fract. Calc. Appl. Anal.* **5**, No 4 (2002), 367–386.
[16] F. Santamaria, S. Wils, E. de Schutter, G.J. Augustine, Anomalous diffusion in Purkinje cell dendrites caused by spines. *Neuron* **52** (2006), 635–648.
[17] F. Santamaria, S. Wils, E. de Schutter, G.J. Augustine, The diffusional properties of dendrites depend on the density of dendritic spines. *Eur. J. Neurosci.* **34**, No 4 (2011), 561–568.

[18] N. Sebaa, Z.E.A. Fellah, W. Lauriks, C. Depollier, Application of fractional calculus to ultrasonic wave propagation in human cancellous bone. *Signal Processing Archive* **86**, No 10 (2006), 2668–2677.
[19] E. Soczkiewicz, Application of fractional calculus in the theory of viscoelasticity. *Molecular and Quantum Acoustics* **23** (2002), 397–404.
[20] I.M. Sokolov, J. Klafter, A. Blumen, Fractional kinetics. *Physics Today* **55**, No 11 (2002), 48–54.
[21] J. Stoer, R. Bulirsch, *Introduction to Numerical Analysis*. Springer, New York (1980).
[22] A. Tepljakov, E. Petlenkov, J. Belikov, FOMCON: Fractional-Order Modeling and Control, Toolbox for MATLAB. In: *18th International Conf. "Mixed Design of Integrated Circuits and Systems"*, June 16-18 (2011), Gliwice, Poland.
[23] D. Valerio, J.S. da Costa, Ninteger: a non-integer control toolbox for Matlab; http://web.ist.utl.pt/duarte.valerio/FDA04T.pdf.

[1] *Department of Biology*
University of Texas at San Antonio
San Antonio, 78249 TX, USA

∗ *e-mail: toma.marinov@utsa.edu*

Received: October 26, 2012

Revised: April 22, 2013

[2] *CBI, University of Texas at San Antonio*
San Antonio, 78249 TX, USA

e-mail: nelson.ramirez@utsa.edu

[3] *Department of Biology*
University of Texas at San Antonio
San Antonio, 78249 TX, USA

∗ *e-mail: fidel.santamaria@utsa.edu*

Please cite to this paper as published in:

Fract. Calc. Appl. Anal., Vol. **16**, No 3 (2013), pp. 670–681;
DOI:10.2478/s13540-013-0042-7