Journal of Computational Physics 297 (2015) 221–236

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Efficient computation of the Grünwald–Letnikov fractional diffusion derivative using adaptive time step memory

Christopher L. MacDonald [a,1], Nirupama Bhattacharya [a,1], Brian P. Sprouse [a], Gabriel A. Silva [a,b,*]

[a] Department of Bioengineering, University of California, San Diego, United States
[b] Department of Ophthalmology, University of California, San Diego, United States

ARTICLE INFO

Article history:
Received 1 June 2014
Received in revised form 15 April 2015
Accepted 29 April 2015
Available online 5 May 2015

Keywords:
Fractional calculus
Subdiffusion
Neuroscience
Neural signaling
Calcium

ABSTRACT

Computing numerical solutions to fractional differential equations can be computationally intensive due to the effect of non-local derivatives in which all previous time points contribute to the current iteration. In general, numerical approaches that depend on truncating part of the system history while efficient, can suffer from high degrees of error and inaccuracy. Here we present an adaptive time step memory method for smooth functions applied to the Grünwald–Letnikov fractional diffusion derivative. This method is computationally efficient and results in smaller errors during numerical simulations. Sampled points along the system's history at progressively longer intervals are assumed to reflect the values of neighboring time points. By including progressively fewer points backward in time, a temporally 'weighted' history is computed that includes contributions from the entire past of the system, maintaining accuracy, but with fewer points actually calculated, greatly improving computational efficiency.

1. Introduction

Fractional differential equations are an extension of integrals and derivatives of integer order. Over the last few decades fractional differential equations have played an increasing role in physics [4,6,25,35], chemistry [8,33], and engineering [5,1,22]. In particular, fractional calculus methods offer unique approaches for modeling complex and dynamic processes at molecular and cellular scales in biological and physiological systems, and there has been a recent interest in the use of fractional calculus methods in bioengineering and biophysical modeling [18,19]. These methods have been applied to modeling ultrasonic wave propagation in cancellous bone [32], bio-electrode behavior at the cardiac tissue interface [20], and describing spiny dendrite neurons using a fractional cable equation [9]. Other work has shown that endogenous lipid granules within yeast exhibit anomalous subdiffusion during mitotic cell division [34].

Here we investigate the numerical implementation and computational performance of a fractional reaction–diffusion equation. The continuous diffusion equation has been the focal point of transport modeling in physical and biological systems for over a hundred years, since first proposed by Adolf Fick in 1855. The practical motivation for the present work arose from considering cell signaling data between Muller neural glial cells in the neural sensory retina. The neural retina

* Corresponding author at: UCSD Jacobs Retina Center, 9415 Campus Point Drive, La Jolla, CA 92037-0946, United States.
E-mail addresses: chris.l.macdonald@gmail.com (C.L. MacDonald), nbhattac@ucsd.edu (N. Bhattacharya), b.p.sprouse@gmail.com (B.P. Sprouse), gsilva@ucsd.edu (G.A. Silva).
[1] CLM and NB contributed equally to this work.

http://dx.doi.org/10.1016/j.jcp.2015.04.048
0021-9991/

is a direct extension and part of the brain itself. Muller cells in the retina can communicate in a paracrine fashion by secreting adenosine triphosphate (ATP) that diffuses into the extracellular space that then triggers intracellular calcium waves. In some cases these signaling events can produce long range regenerative intercellular signaling in networks of cells; see [27,24,37]. In particular, from experimental data published in [27], we qualitatively observed a peaked diffusion profile for ATP where the central cusp-like shape persisted over a longer period of time than would be expected for Gaussian diffusion; this is a typical characteristic of anomalous subdiffusion [26]. Therefore we suspected that the diffusion of ATP in this physiological system was subdiffusive in nature. Neurobiologically, this could ultimately have an important effect on underlying physiology and cellular signaling. Motivated by these observations, we decided to explore ways of simulating and exploring such subdiffusive dynamics using numerical methods amenable to experimental methods and data. Modeling subdiffusive neurophysiological transport processes necessitates the use of fractional differential equations or other related objects. However, the numerical implementation of such equations involves non-local fractional derivative operators that require taking into account the entire past history of the system in order to compute the state of the system at the current time step. This produces a significant computational burden that limits the practicality of the models. In the present paper we introduce an 'adaptive time step memory' approach for computing the contribution of the memory effect associated with the history of a system in a way that makes it both numerically and resource (i.e. computer memory) efficient and robust while maintaining accuracy. While our algorithms can be applied to any diffusion application modeled as a fractional process, they offer particular value for modeling complex processes with long histories, such as lengthy molecular or cellular simulations.

Approaches for numerically approximating solutions to fractional diffusion equations have been extensively studied [2,10,12,21,23,29,38], but in general there is always a trade off between computational efficiency, complexity, and the accuracy of the resultant approximations. There has also been a considerable amount of work done on developing fast convolution quadrature algorithms, which is relevant to fractional differential equations because the non-local nature of fractional calculus operators results in either a continuous or discrete convolution of some form. In particular, Lubich and others [15,14,17,31,13] have built a generalized and broadly applicable convolution quadrature framework to numerically approximate a continuous convolution integral with a wide array of possible convolution kernel functions (including oscillatory kernels, singular kernels, kernels with multiple time scales, and unknown kernels with known Laplace transforms), while achieving good algorithmic performance with respect to complexity and memory requirements. However, their algorithms are necessarily very complicated in order to handle a wide range of functions and expressions in the convolution while limiting storage requirements and scaling of arithmetic operations. They involve approximating a continuous convolution based on numerical inversions of the Laplace transform of the convolution kernel function using contour integrals discretized along appropriately chosen Talbot contours or hyperbolas. The scaling of the number of required arithmetic operations involved in the convolution, and overall memory requirements, are both reduced by splitting up the continuous convolution integral or discrete convolution quadrature summation into a carefully chosen series of smaller integrals or summations, and solving a set of ordinary differential equations one time step at a time without storing the entire history of the solutions. For methods explicitly involving a discrete convolution summation, the quadrature weights are calculated using the Laplace transform of the original kernel and linear multistep methods for solving differential equations. FFT techniques can be applied to calculate these weights simultaneously and further increase the efficiency of the algorithm [16]. These authors have demonstrated the success of their framework by simulating various differential equations involving convolutions, including a one-dimensional fractional diffusion equation [31]. In [11], Li takes an alternative approach from this framework and focuses on a fast time-stepping algorithm specifically for fractional integrals by constructing an efficient Q point quadrature, but does not focus on how this fits into numerical algorithms representing larger mathematical models.

The methods we introduce here are also efficient, significantly reducing simulation times, computational overhead, and required memory resources, without significantly affecting the computed accuracy of the final solution. However, in contrast to broad generalized frameworks, our algorithms are focused on solving fractional differential equations involving non-local fractional derivative operators. We approximate the fractional derivative based on the Grünwald–Letnikov definition instead of pursuing a general quadrature formula approximation to the Riemann–Liouville integral definition of the fractional derivative. The Grünwald–Letnikov derivative is used in many numerical schemes for discretizing fractional diffusion equations from a continuous Riemann–Liouville approach [3,38,28] and involves a discrete convolution between a 'weight' or coefficient function and the function for which we are interested in taking the derivative. The mathematics and theory of this form of a weighting function are well established in the literature [28,15]. Building on this foundation avoids the need for domain transformations, contour integration or involved theory. Our algorithms were specifically developed for real world applied diffusion and transport physics and engineering problems, often where there are practical measurement limitations to the types and quality (e.g. granularity) of the data that can be collected or observed. For situations such as these, our methods for handling discrete convolutions associated with fractional derivatives are more intuitive and accessible than other generalized mathematical methods, where it is often not obvious or clear how to implement and apply a generalized approach to a specific physical problem under a defined set of constraints. The approaches we introduce here can be incorporated with various combinations of finite difference spatial discretizations and time marching schemes of a larger mathematical model in a straightforward way.

The increased efficiency and memory savings in the approaches we describe here lie in the way the discrete summation is calculated. The conceptual basis that results in a saving of computational time without a huge tradeoff in accuracy is in the interpretation of the 'weight' function as a measure of the importance of the history of a system. The more recent history

of the system is more important in determining the future state of the system, and therefore we make use of an 'adaptive time step memory' method by changing the interval of the backwards time summation in the Grünwald–Letnikov fractional derivative. Instead of incorporating every previous time point into the most current calculation, only certain points are recalled based upon their proximity to the current point in time, weighted according to the sparsity of the time points. This substantially reduces the computational overhead needed to recalculate the prior history at every time step, yet maintains a high level of accuracy compared to other approximation methods that make use of the Grünwald–Letnikov definition. We consider two adaptive step approaches – one based on an arithmetic sequence and another based on a power law, that when combined with a linked-list computational data structure approach yield $\mathcal{O}(log_2(N))$ active memory requirements. This is a significant improvement over keeping all N steps of the system's history. We compare our adaptive time step approach with the 'short memory' method described by Volterra [36] and Podlubny et al. [28], and examine differences in simulation times and errors under similar conditions for both. In the last section we sketch out a 'smart' adaptive memory method based on a continuous form of the Grünwald–Letnikov that will be able to accommodate more dynamic past histories, i.e., histories with sharp and abrupt changes relative to the time step being considered, that produce larger error in the discrete methods we discuss. Finally, we note that the scope of the present work focuses on a detailed development of the methods and algorithms. A full theoretical analysis of stability and algorithmic complexity is the topic of a companion paper that will follow this one.

2. Fractional diffusion and the Grünwald–Letnikov derivative

2.1. Derivation of the fractional diffusion equation

We begin by considering the standard diffusion equation with no reaction term

$$\frac{\partial u}{\partial t} = \alpha_s \nabla^2 u, \tag{2.1}$$

where α_s is the standard diffusion coefficient given in units distance^2/time. The time-fractional version of Eq. (2.1) is given by

$$\frac{\partial^\gamma u}{\partial t^\gamma} = \alpha \nabla^2 u, \tag{2.2}$$

where here the diffusion coefficient α has fractional order units of distance^2/time$^\gamma$. This is the simplest form of the fractional diffusion equation. The real values taken by γ in Eq. (2.2) dictate the diffusion regimes obtained by the equation. Values of $0 < \gamma < 1$ result in subdiffusion, characterized by a slow diffusion profile relative to regular diffusion. When $\gamma > 1$, an oscillatory component emerges in the diffusion profile resulting in superdiffusion. As required, when $\gamma = 1$ the fractional diffusion equation reduces to standard diffusion that results in a Gaussian profile. When $\gamma = 2$ it reduces to the standard wave equation. As a point of discussion, we note that there no obvious theoretical (mathematical) limits to the value γ can take. However, there necessarily exist practical limits on γ when considering stability and maintaining order of accuracy for the numerical implementation and algorithms of the fractional differential equation. Here we only concern ourselves with a physically plausible interpretation behind the equation. We know what subdiffusion and superdiffusion correspond to physically, but, so far as we are aware, there is no straight forward meaning of Eq. (2.2) when $\gamma > 2$ that we are aware of. This may be a shortcoming of our current physical understanding in the sense that the physics underlying such cases are not yet well understood or have not yet been discovered to be connected to particular real world dynamical processes. So although theoretically there is no apparent limit, there is a practical numerical implementation limit that will depend on a desired order of accuracy and computational efficiency. As we eluded to in the Introduction, we leave a full theoretical analysis of the stability and complexity of our methods to follow up companion paper.

Using the relation

$$\frac{\partial u}{\partial t} = \frac{\partial^{1-\gamma}}{\partial t^{1-\gamma}} \frac{\partial^\gamma u}{\partial t^\gamma} \tag{2.3}$$

and substituting into Eq. (2.2) yields

$$\frac{\partial u}{\partial t} = \frac{\partial^{1-\gamma}}{\partial t^{1-\gamma}} \alpha \nabla^2 u \tag{2.4a}$$

$$\frac{\partial u}{\partial t} = \alpha D^{1-\gamma} \nabla^2 u \tag{2.4b}$$

where $D^{1-\gamma}$ denotes the fractional derivative operator which we define and discuss at length in the next section.

If we take into account consumption or generation processes, then we can rewrite the diffusion equation as

$$\frac{\partial u(\vec{x}, t)}{\partial t} = \alpha D^{1-\gamma} \nabla^2 u(\vec{x}, t) + f(u), \quad \vec{x} = \{x_1, x_2 \ldots x_N\}, \quad u(\vec{x}, 0) = u_0(\vec{x}) \tag{2.5}$$

where $\vec{x}$ is an N dimensional vector and $f(u)$ represents a consumption or generation term. For the specific examples we describe in this paper we implement the simplest Dirichlet boundary conditions where all boundaries are set to zero. However, our method can handle various other boundary conditions, including Dirichlet conditions set to any values, Neumann conditions, and others, in a straight forward manner. In the following section we consider the fractional derivative operator $D^{1-\gamma}$.

2.2. Definition of the Grünwald–Letnikov derivative

The ath order fractional derivative of a function, denoted by $D^a f$, extends the order of the differential operator from the set of integers to the set of real numbers. The operator can be mathematically defined in several ways, including standard descriptions like the Riemann–Liouville definition, the Grünwald–Letnikov definition, and others. For numerical simulations, we find the Grünwald-Letnikov derivative to be convenient, since it is based on the standard differential operator but made applicable to arbitrary order a with a discrete summation and binomial coefficient term:

$$D^a f(x) = \lim_{h\to 0} h^{-a} \sum_{m=0}^{x/h} (-1)^m \binom{a}{m} f(x-mh). \tag{2.6}$$

Expanding the binomial coefficient yields

$$\binom{a}{m} = \frac{a!}{m!(n-m)!} \tag{2.7}$$

and expressing the factorial components of the binomial coefficient by the gamma function gives

$$\binom{a}{m} = \frac{\Gamma(a+1)}{m!\Gamma(a+1-m)}. \tag{2.8}$$

Combining Eqs. (2.6) and (2.8) yields the Grünwald–Letnikov derivative for real numbers:

$$D^a f(x) = \lim_{h\to 0} h^{-a} \sum_{m=0}^{x/h} \frac{(-1)^m \Gamma(a+1)}{m!\Gamma(a+1-m)} f(x-mh) \quad a \in \mathbb{R},\ a \neq -\mathbb{N}_1. \tag{2.9}$$

It should be noted that computation of the Grünwald–Letnikov derivative requires knowledge of the entire past history of the system due to the summation operator. This means that this fractional derivative, unlike standard derivatives, is no longer local but global. Computing the value of a real derivative requires knowledge of the system's past history. The derivative must take into account all past points from $m = 0$, the current point, all the way back to the beginning point $m = x/h$. This requirement places significant computational demands on the numerical implementation of the Grünwald–Letnikov derivative, and is the principle technical motivation for the results presented here.

In the rest of the paper we explore the implementation of Eq. (2.9) to a diffusion regime constrained to $0 < \gamma \equiv a \leq 2$ (as explained in Section 2.1):

$$D^\gamma f(x) = \lim_{h\to 0} h^{-\gamma} \sum_{m=0}^{x/h} \frac{(-1)^m \Gamma(\gamma+1)}{m!\Gamma(\gamma+1-m)} f(x-mh) \text{ for } 0 < \gamma \leq 2. \tag{2.10}$$

Applying Eq. (2.10) to the fractional diffusion equation (Eq. (2.4b)) yields:

$$\frac{\partial u}{\partial t} = \lim_{\tau\to 0} \tau^{\gamma-1} \alpha \sum_{m=0}^{t/\tau} \frac{(-1)^m \Gamma(2-\gamma)}{m!\Gamma(2-\gamma-m)} \nabla^2 u(t-m\tau) \tag{2.11}$$

where t represents instantaneous time, and τ is the time step.

Next, define a function $\psi(\gamma, m)$ such that

$$\psi(\gamma, m) = \frac{(-1)^m \Gamma(2-\gamma)}{m!\Gamma(2-\gamma-m)}. \tag{2.12}$$

Given that the standard definition of the gamma function implies $\Gamma(a) = \frac{1}{a}\Gamma(a+1)$, Eq. (2.12) can be reduced to a recursive relationship. Consider the first four terms in the chain from $m = 0$ to $m = 3$:

$$\psi(\gamma, 0) = \frac{(-1)^0 \cdot \Gamma(2-\gamma)}{(0!) \cdot \Gamma(2-\gamma-0)} = 1$$

$$\psi(\gamma, 1) = \frac{(-1)^1 \cdot \Gamma(2-\gamma)}{(1!) \cdot \Gamma(2-\gamma-1)} = \frac{(-1) \cdot (-1)^0 \cdot \Gamma(2-\gamma)}{(1) \cdot (0!) \cdot \frac{1}{(2-\gamma-1)} \cdot \Gamma(2-\gamma-0)}$$

$$\psi(\gamma,2)=\frac{(-1)^2\cdot\Gamma(2-\gamma)}{(2!)\cdot\Gamma(2-\gamma-2)}=\frac{(-1)\cdot(-1)^1\cdot\Gamma(2-\gamma)}{(2)\cdot(1!)\cdot\frac{1}{(2-\gamma-2)}\cdot\Gamma(2-\gamma-1)}$$

$$\psi(\gamma,3)=\frac{(-1)^3\cdot\Gamma(2-\gamma)}{(3!)\cdot\Gamma(2-\gamma-3)}=\frac{(-1)\cdot(-1)^2\cdot\Gamma(2-\gamma)}{(3)\cdot(2!)\cdot\frac{1}{(2-\gamma-3)}\cdot\Gamma(2-\gamma-2)}$$

Examining this system of equations suggests that for any $\psi(\gamma,m)$:

$$\psi(\gamma,m)=\frac{(-1)^m\cdot\Gamma(2-\gamma)}{(m!)\cdot\Gamma(2-\gamma-m)}=\frac{(-1)\cdot(-1)^{(m-1)}\cdot\Gamma(2-\gamma)}{(m)\cdot(m-1)!\cdot\frac{1}{(2-\gamma-m)}\cdot\Gamma(2-\gamma-(m-1))}. \tag{2.13}$$

Because the function $\psi(\gamma,m)$ is dependent on $\psi(\gamma,m-1)$, an iterative relationship forms that scales by $\frac{-(2-\gamma-m)}{m}$:

$$\psi(\gamma,m)=-\psi(\gamma,m-1)\frac{2-\gamma-m}{m}. \tag{2.14}$$

This recursive function is valid for all γ including subdiffusion, standard diffusion and superdiffusion, so this equation is general over all regimes. Because the entire history of the system must be taken into account when computing the current time step, $\psi(\gamma,m)$ is used for many m values, many times over the course of the simulation, and can be precomputed for values of $m=0$ to $m=N$, where N is the total number of time points, resulting in a significant savings in computational performance. A similar simplification has been previously discussed and used in [28]. Taking $\psi(\gamma,m)$ into account yields the final form of the fractional diffusion equation, given by

$$\frac{\partial u}{\partial t}=\lim_{\tau\to 0}\tau^{\gamma-1}\alpha\sum_{m=0}^{t/\tau}\psi(\gamma,m)\nabla^2u(t-m\tau). \tag{2.15}$$

We can also arrive at Eq. (2.15) by an alternative approach. Begin by considering again the Grünwald-Letnikov derivative in terms of binomial notation:

$$D^{1-\gamma}f(t)=\lim_{\tau\to 0}\tau^{\gamma-1}\sum_{m=0}^{t/\tau}(-1)^m\binom{1-\gamma}{m}f(t-m\tau). \tag{2.16}$$

Applying Eq. (2.16) to the fractional diffusion equation (Eq. (2.4b)) yields

$$\frac{\partial u}{\partial t}=\lim_{\tau\to 0}\tau^{\gamma-1}\alpha\sum_{m=0}^{t/\tau}(-1)^m\binom{1-\gamma}{m}\nabla^2u(\vec{x},t-m\tau) \tag{2.17}$$

Knowing the gamma function definition of the factorial operator, $a!=\Gamma(a+1)$, the binomial coefficient can be expanded to accommodate real numbers (also cf. Eq. (2.8)):

$$\binom{1-\gamma}{m}=\frac{(1-\gamma)!}{m!(1-\gamma-m)!}=\frac{\Gamma(2-\gamma)}{m!\Gamma(2-\gamma-m)}. \tag{2.18}$$

Note how combining Eqs. (2.17) and (2.18) yields Eq. (2.11) as required.

Next, defining the function $\psi(\gamma,m)$ (Eq. (2.12)) using binomial notation gives

$$\psi(\gamma,m)=(-1)^m\binom{1-\gamma}{m} \tag{2.19}$$

and substituting the relation $\Gamma(a)=(a-1)\Gamma(a-1)$ into Eq. (2.18) results in

$$\psi(\gamma,m)=\frac{(-1)^{m-1}\Gamma(2-\gamma)}{(m-1)!\Gamma(2-\gamma-(m-1))}\frac{-1}{m\frac{1}{2-\gamma-m}} \tag{2.20}$$

yielding the iterative relationship in Eq. (2.14) above:

$$\psi(\gamma,m)=-\psi(\gamma,m-1)\frac{2-\gamma-m}{m}.$$

With the substitution of $\psi(\gamma,m)$ into Eq. (2.17) we once again recover the time-fractional diffusion equation in the form

$$\frac{\partial u}{\partial t}=\lim_{\tau\to 0}\tau^{\gamma-1}\alpha\sum_{m=0}^{t/\tau}\psi(\gamma,m)\nabla^2u(\vec{x},t-m\tau). \tag{2.21}$$

2.3. Discretization of the fractional diffusion equation

Using

$$\nabla^2 u(\vec{x}, t - m\tau) = \frac{\partial^2 u(\vec{x}, t - m\tau)}{\partial x_1^2} + \frac{\partial^2 u(\vec{x}, t - m\tau)}{\partial x_2^2} \tag{2.22}$$

as the formulation in two spatial dimensions, one can discretize the function into a finite difference based FTCS scheme (forward time centered space) on a grid $u_{j,l}^k$ (where $k = t/\Delta_t$, $j = x_1/\Delta_x$, $l = x_2/\Delta_x$, Δ_x is the grid spacing in both directions assuming an equally spaced grid, and Δ_t is the time step), using the relations [30]

$$\begin{aligned}
\frac{\partial^2 u(\vec{x}, t - m\tau)}{\partial x_1^2} &= \frac{u_{j+1,l}^{k-m} - 2u_{j,l}^{k-m} + u_{j-1,l}^{k-m}}{\Delta_x^2} \\
\frac{\partial^2 u(\vec{x}, t - m\tau)}{\partial x_2^2} &= \frac{u_{j,l+1}^{k-m} - 2u_{j,l}^{k-m} + u_{j,l-1}^{k-m}}{\Delta_x^2} \\
\frac{\partial u(\vec{x}, t)}{\partial t} &= \frac{u_{j,l}^{k+1} - u_{j,l}^{k}}{\Delta_t}.
\end{aligned} \tag{2.23}$$

In the discrete limit where $\tau \to \Delta_t$, we approximate the fractional diffusion equation (Eqs. (2.15) and (2.21)) with the following finite difference expression, which has a first-order approximation $O(\tau)$ (Chap. 7 in [28], [15]):

$$\frac{u_{j,l}^{k+1} - u_{j,l}^{k}}{\Delta_t} = \alpha \frac{\Delta_t^{\gamma-1}}{\Delta_x^2} \sum_{m=0}^{k} \psi(\gamma, m) \delta_{j,l}^{k-m} \tag{2.24}$$

where $\delta_{j,l}^{k-m}$ is the finite difference kernel given by

$$\delta_{j,l}^{k-m} = \left(u_{j+1,l}^{k-m} + u_{j-1,l}^{k-m} - 4u_{j,l}^{k-m} + u_{j,l+1}^{k-m} + u_{j,l-1}^{k-m} \right). \tag{2.25}$$

Adding a consumption/generation term is straightforward in this implementation. For example, take an exponential decay term given by

$$\frac{\partial u}{\partial t} = -\beta u \tag{2.26}$$

with the complementary finite difference relation

$$\frac{u_{j,l}^{k+1} - u_{j,l}^{k}}{\Delta_t} = -\beta u_{j,l}^{k}. \tag{2.27}$$

Incorporating Eq. (2.26) into the form of Eq. (2.5) results in

$$\frac{\partial u}{\partial t} = \alpha D_t^{1-\gamma} \nabla^2 u - \beta u, \tag{2.28}$$

which gives the full finite difference implementation in two dimensions

$$\frac{u_{j,l}^{k+1} - u_{j,l}^{k}}{\Delta_t} = \alpha \frac{\Delta_t^{\gamma-1}}{\Delta_x^2} \sum_{m=0}^{k} \psi(\gamma, m) \delta_{j,l}^{k-m} - \beta u_{j,l}^{k} \tag{2.29}$$

Fig. 1 shows the results of four simulations for different values of γ. Simulations were run on a 100×100 grid with initial conditions $U_{50,50}^0 = 0.1$, $U_{51,50}^0 = U_{50,51}^0 = U_{49,50}^0 = U_{50,49}^0 = 0.05$ and zero elsewhere. Dirichlet boundary conditions were implemented and the simulation edge set to zero. Simulations were run for $t = 100$ seconds with $\alpha = 1$, $\beta = 0$, and $\Delta_x = 5$.

3. Adaptive time step memory as an arithmetic sequence

3.1. Derivation

In Eq. (2.29) each iteration requires the re-calculation and summation of every previous time point convolved with $\psi(\gamma, m)$. This becomes increasingly cumbersome for large times, which require significant numbers of computations and memory storage requirements. To address this, Podlubny et al. [28] introduced the 'short memory' principle which assumes that for a large t the role of the previous steps or 'history' of the equation become less and less important as the convolved

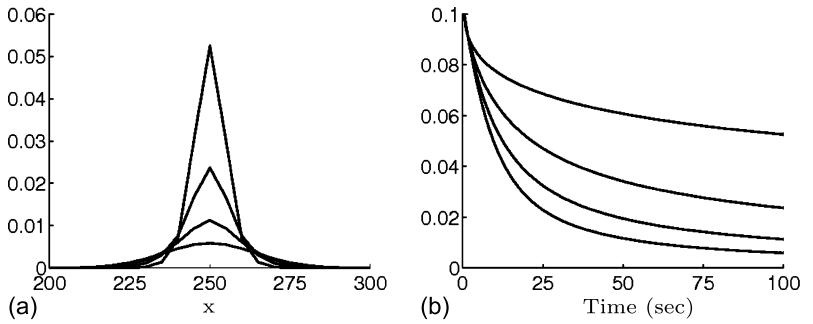

Fig. 1. Simulation results for $\gamma = 0.5,\ 0.75,\ 0.9,\ 1.0$ (for traces from top to bottom) in one dimensional space (panel (a)) and time (panel (b)). As the subdiffusion regime is approached the profile becomes more and more hypergaussian.

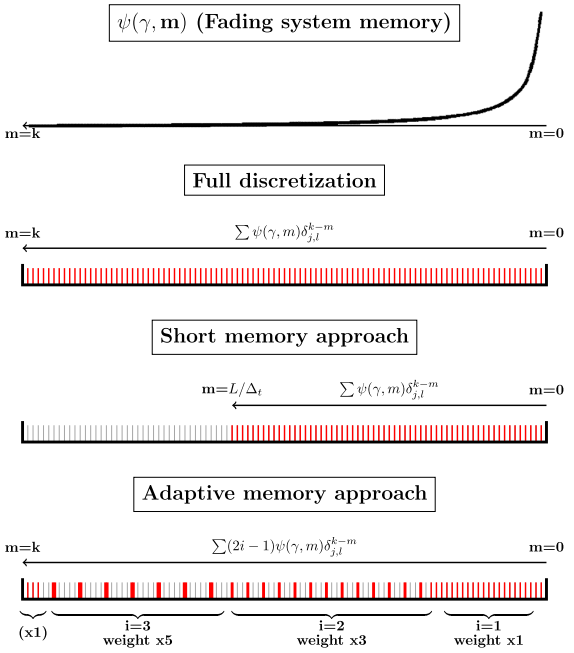

Fig. 2. Short memory and adaptive memory methods for estimating the Grünwald–Letnikov discretization. Both approximations rely on the sharply decreasing function $|\psi(\gamma, m)|$ as m increases to omit points from the backwards summation. While short memory defines a sharp cut off of points, adaptive memory provides a weighted sampling of points for the entire history of the system. Points included in the computation by each method are highlighted in red. The shape of ψ is different for $\gamma < 1$ and $\gamma > 1$, but the shape of $|\psi|$ remains a monotonically decreasing function (as m increases) for both cases, and remains consistent with the principle that more recent time points contribute (whether positively or negatively) more to the solution at the next time step, than time points further back in the history of the system. See text for details. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$\psi(\gamma, m)$ shrinks towards zero. This would then result in approximating Eq. (2.29) by truncating the backwards summation, only taking into account times on the interval $[t - L, t]$ instead of $[0, t]$, where L is defined as the 'memory length' (Eq. (7.4) in [28]; Fig. 2). While computationally efficient, this approach leads to errors in the final solution since not all points are counted in the summation. Despite the resultant errors, this numerical method represents a powerful and simple approach for providing a reasonable trade off between computational overhead and numerical accuracy. In the context of the implementation derived here, it would result in a discretization scheme given by

$$\frac{u_{j,l}^{k+1} - u_{j,l}^{k}}{\Delta_t} = \alpha \frac{\Delta_t^{\gamma-1}}{\Delta_x^2} \sum_{m=0}^{min(L/\Delta_t, k)} \psi(\gamma, m)\delta_{j,l}^{k-m} - \beta u_{j,l}^{k} \tag{3.1}$$

As an alternative to the method of Podlubny, Ford and Simpson proposed an alternative 'nested mesh' variant that gives a good approximation to the true solution at a reasonable computational cost [7]. However, this method is exclusive to the Caputo fractional derivative. Here we introduce an approach that is applicable to the more general Grünwald–Letnikov definition. Like these other methods, it also shortens computational times but at the same time results in much greater

accuracy than the use of 'short memory'. We achieve this by introducing the concept of an 'adaptive memory' into the Grünwald–Letnikov discretization.

The underlying principle of the adaptive time approach is that relative to the current time point previous time points contribute different amounts to the summation. Values relatively closer to the current time point will have a greater contribution to the current numerical calculation than values many time points back due to the multiplier $\psi(\gamma, m)$. For smooth functions, as m increases and $\left|\psi(\gamma, m)\right|$ decreases, neighboring points in the summation exhibit only small differences. Consequently, one can take advantage of this and utilize an 'adaptive memory' approach in which neighboring values at prior time points are grouped together in defined increments and the median taken as a representative contribution for the increment weighted according to the length of the increment to account for the skipped points. This results in fewer time points that need to be computed in a summation. Algorithmically, for an arbitrary number (define this as parameter a) of time steps back from the current time point k for which the history of the system is being computed, consider an initial interval $[0, a]$ for which all time points within this interval are used in the summation and therefore contribute to the Grünwald–Letnikov discretization. Let subsequent time intervals become longer the further away from k they are and within them only every other d time points are used in the summation term, i.e., only the median values of a temporally shifting calculation increment of length d along the current interval are considered. As subsequent intervals become longer, so does d, thereby requiring less points to compute (Fig. 2).

Definition 3.1. Let k be the current iterative time step in a reaction diffusion process for which a Grünwald–Letnikov discretization is being computed. Consider an arbitrary time point a in the history of the system backwards from k. For $i \in \mathbb{N}_1, i \neq 1$, define an interval of this past history by

$$I = [a^{i-1} + i, a^i] \tag{3.2}$$

where $\mathbb{N}_1$ represents the set of natural numbers beginning from one. Given how the indices i are defined, the very first interval backwards from k is independent of Eq. (3.2) and is given by $[0, a]$. This is considered the base interval. Subsequent intervals are defined as a function of this base, i.e., as a function of a and determined by Eq. (3.2). Let i_{max} be the value of i such that $k \in I_{max} = [a^{i_{max}-1} + i_{max}, a^{i_{max}}]$. The complete set of intervals then is defined as $\zeta = \{I = [a^{i-1} + i, a^i] : i \in \mathbb{N}_1, i \neq 1, i \leq i_{max}\}$.

Definition 3.2. For the set of intervals ζ as defined in Definition 3.1, $D = \{d = 2i - 1 : i \in \mathbb{N}_1, i \neq 1\}$ is the set of distances d by which the corresponding intervals in ζ are sampled at.

Theorem 3.3. *In general, for two-dimensional diffusion without consumption or generation terms for any interval as defined in Definition 3.1, the Grünwald-Letnikov discretization with adaptive time step memory is given by*

$$\frac{u_{j,l}^{k+1} - u_{j,l}^{k}}{\Delta_t} = \alpha \frac{\Delta_t^{\gamma-1}}{\Delta_x^2} \left[\sum_{n=0}^{a} \psi(\gamma, n) \delta_{j,l}^{k-n} + \cdots \right.$$
$$\left. \sum_{i=2}^{i_{max}} \sum_{m_i=a^{i-1}+i}^{a^i} (2i-1) \psi(\gamma, m_i) \delta_{j,l}^{k-m_i} + \sum_{p=m_{max}+i_{max}}^{k} \psi(\gamma, p) \delta_{j,l}^{k-p} \right] \tag{3.3}$$

where $p \in \mathbb{N}_1$, and for each i (i.e. for each interval) $M = \{m_i = a^{i-1} + (2i-1)\eta - i + 1 : \eta \in \mathbb{N}_1$ and $m_i \leq m_{max}\}$ is the set of time points over which $\psi(\gamma, m)\delta_{j,l}^{k-m}$ is evaluated. Since the time point k may be less than the full length of the last interval I_{max}, $|m_{max}| \leq |k - i_{max}|$ represents the maximum value in I_{max} that is evaluated, i.e. the last element in the set M for I_{max}.

Proof. The first summation represents the basis interval and the limits of the summation operator imply the contribution of every time point, i.e., $n \in \mathbb{N}_1$. For intervals beyond a: any arithmetic sequence defined by a recursive process $\nu_\eta = \nu_{\eta-1} + d$, $\eta \in \mathbb{N}_1$ for some distance d, the ηth value in the sequence can be explicitly calculated as $\nu_\eta = \nu_1 + (\eta - 1)d$ given knowledge of the sequence's starting value ν_1. For the set ζ this implies that $\nu_1 = a^{i-1} + i$ and $d = 2i - 1$ for a given i. This then yields $\nu_\eta = a^{i-1} + i + (\eta - 1)(2i - 1) = a^{i-1} + (2i - 1)\eta - i + 1 := m_i$ as required. The outer summation collects the summations of all intervals that belong to the set ζ. The last summation on the interval $[m_{max} + i_{max}, k]$ ensures that the final point(s) of the backwards summation are still incorporated into the computation even if the arbitrarily chosen value of a does not generate a final increment length that ends on k. □

Note that D is not explicitly needed for computing Eq. (3.3) because the distances d are implicitly taken into account by ζ. Using the median value of each increment avoids the need for interpolation between time points. The implementation of the adaptive memory method described here is necessarily limited to smooth functions due to the assumption that the neighbors of the median values used in the summation do not vary much over the increment being considered. This method essentially allows for a contribution by all previous time points to the current calculation, yet reduces computational times by minimizing the total number of calculations.

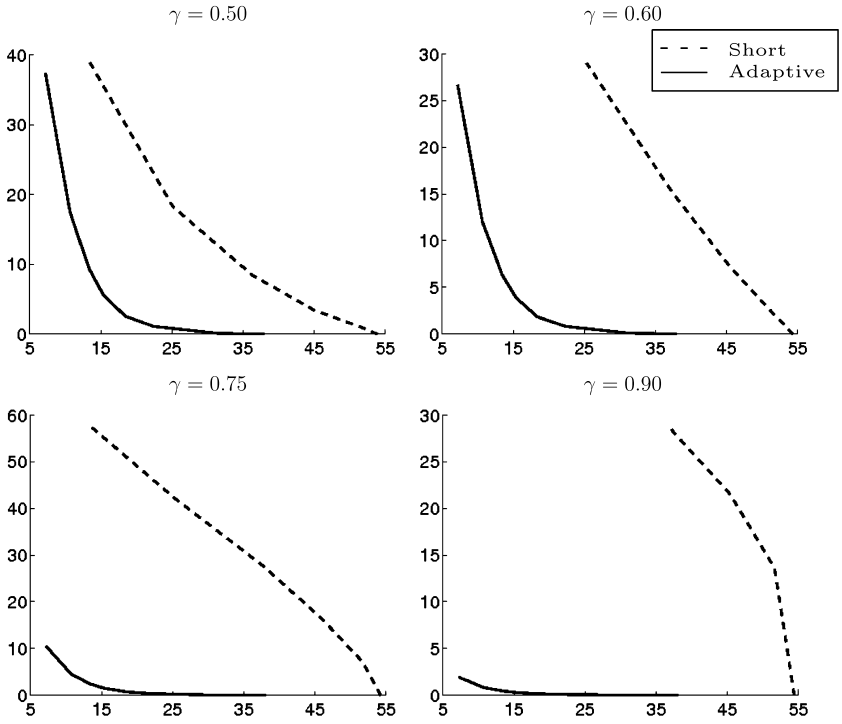

Fig. 3. Comparison of the error between adaptive memory and the short memory as a function of the calculation time (x-axis: computation run time in seconds) expressed as a percentage error relative to the computed value for the full non-shortened approach (y-axis). Four different values of γ are shown.

3.2. Comparison to short memory methods

The results of using various L (for short memory) and interval steps a (for adaptive memory) are shown in Fig. 3. Increasing the values of L and a resulted in a decrease in the error of the estimated results but at the cost of increased computation times. Conversely, decreasing the values of L and a resulted in a decrease in computation times, but at the expense of accuracy. In all cases however, the adaptive memory method had a significantly lower error for the same simulation time, and also reached a clear asymptotic minimum error much faster than the minimum error achieved by the short memory method. In these simulations, $\alpha = 1$, $\beta = 0$, $\Delta_t = 1$, $\Delta_x = 10$, using a 20×20 grid, and ran for $t = 1500$ where $U^0_{10,10} = 10$. The error for the 'short memory' method increased comparatively quickly and worsened as $\gamma \to 1$. This was due to the fact that the evolution of the solution was initially quite fast at times near $t = 0$, which were the first time points cut by the 'short memory' algorithm.

4. Adaptive time step as a power law

While computationally efficient and intuitively straightforward, the major drawback to the adaptive memory method computed as an arithmetic sequence, is the necessary amount of allocated memory. The backwards time summation grid changes with every step such that every point in the history needs to be retained and stored during the course of a simulation. For high dimensional diffusion this limits the practical applicability of the method. For example, if one were to solve the three dimensional diffusion equation on just a $100 \times 100 \times 100$ set of grid points, that would require the storage of 1,000,000 data points per time step, which over the course of a long simulation would overwhelm the memory capacity of most computers in addition to greatly slowing simulation times. The same memory issues arise when considering another popular approach used for solving the fractional diffusion equation, discussed in [29], that uses an implicit matrix solution to simultaneously solve all points in time using a triangular strip matrix formulation. This later approach is very powerful but does not take advantage of any short memory algorithms.

In this section we improve on our results and develop a framework that uses a power law-based rule for eliminating past history points without sacrificing numerical accuracy. While the adaptive memory algorithm can be used with various distributions other than an arithmetic sequence, our motivation for choosing a power law-based distribution is that it matches the natural power law dynamics of the underlying mathematics associated with the time-fractional derivative, and it is also easy to model and implement. We therefore considered it a good starting point for improvements on the original arithmetic sequence version of the adaptive memory algorithm. We note however that other distributions can be combined with an adaptive step approach that would result in varying degrees of efficiency and accuracy. This will depend on how

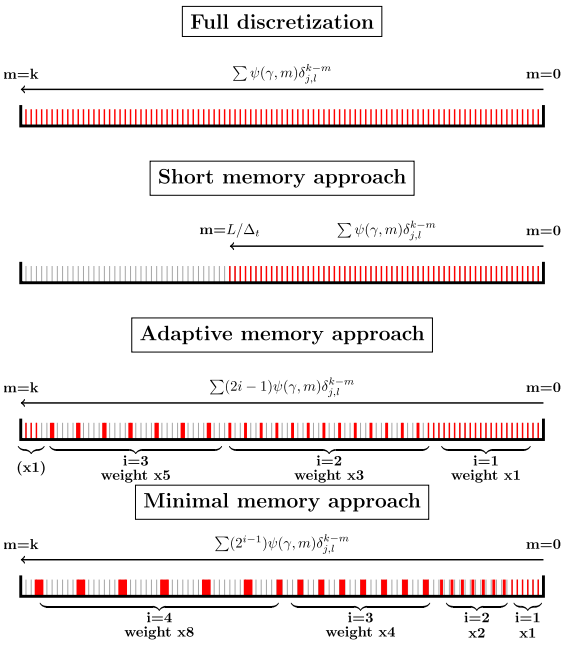

Fig. 4. Minimal memory implementation.

well they match the dynamics associated with time-fractional diffusion. In this section though we only consider a power law-based rule.

An adaptive memory approach applied to a power law distribution, in combination with a linked-list based method, minimizes the storage of the past history of the function. This results in decreasing the amount of total memory allocated to run a simulation of N time steps from $\mathcal{O}(N)$ to $\mathcal{O}(log_2(N))$, resulting in a tremendous memory savings. Given a simulation of N time steps and number of grid points X/Δ_x, where X is the grid width, storing the entire past history would require $\frac{X}{\Delta_x}N$ points to be stored, which grows linearly with N. In contrast, an adaptive mesh with a power law growth in spacing (in this case $1, 2, 4\ldots$), results in $\frac{X}{\Delta_x}log_2(N)$ points being stored in memory, which grows with the *log* of the number of points N. The advantage of using such a power law scaling is that one can *a priori* calculate memory points which will not be used at every time step of the simulation and de-allocate the memory assigned to storing those time points. With the adaptive memory method, past points needed for each subsequent time step change with each iteration, necessitating that all time points be stored for future calculations. The use of a self similar power law allows the elimination of points that will not be needed at any future time in the calculation. A comparison of the implementation of this method with the full implementation using every time point, is shown in Fig. 4.

4.1. Set theoretic implementation

In one spatial dimension, an FTCS discretization of the fractional diffusion equation on a grid u_j^i where $i = t/\Delta_t$, $j = x_1/\Delta_x$, can be written as

$$\frac{u_j^{i+1} - u_j^i}{\Delta_t} = \alpha \frac{\Delta_t^{\gamma-1}}{\Delta_x^2} \sum_{m=0}^{i} \psi(\gamma, m)\delta_j^{i-m} \tag{4.1}$$

(cf. Eqs. (2.23) to (2.25) above and [30]), where

$$\delta_j^i = \left(u_{j+1}^i - 2u_j^i + u_{j-1}^i\right) \tag{4.2}$$

Definition 4.1. Define a well-ordered set U in time consisting of elements u_j^i ordered by the point in time $i = t/\Delta_t$ at which that grid point occurred. The least elements in this set are then the points where u_j^0, and the greatest are the points u_j^k, where the current time point is $k = t_{current}/\Delta_t$. For all integers A and B, if $A > B$, then $u_j^A > u_j^B$.

Given the numerical scheme in (4.1), the set U can be expressed as the recursive algorithm

$$u_j^{k+1} = u_j^k + \alpha \frac{\Delta_t^\gamma}{\Delta_x^2} \sum_{\{u_j^i \in U\}} \psi(\gamma, k-i)\delta_j^i, \tag{4.3}$$

where u_j^0 is the set of initial conditions at $t = 0$.

Taking advantage of this result we can state the following lemma for the short memory approach:

Lemma 4.2. *Assume a short memory scheme. The elements $u^i < u^{k-\frac{L}{\Delta t}}$ in U for a time step i are not used in future computations and can be removed from U.*

The set U can then become a list of sets U^i with only the necessary elements to complete the recursive relation in each step i.

Proof. The short memory approach shown in Fig. 4 can be written as

$$u_j^{k+1} = u_j^k + \alpha \frac{\Delta_t^\gamma}{\Delta_x^2} \sum_{\{u_j^i \in U : u_j^i > u^{k-\frac{L}{\Delta t}}\}} \psi(\gamma, k-i)\delta_j^i. \tag{4.4}$$

This recursive relation can be written as

$$\begin{aligned} u_j^{k+1} &= u_j^k + \alpha \frac{\Delta_t^\gamma}{\Delta_x^2} \sum_{\{u_j^i \in U^k\}} \psi(\gamma, k-i)\delta_j^i. \\ U^{k+1} &:= \{u^i \in U^k : u^i > u^{k-\frac{L}{\Delta t}}\} + \{u^{k+1}\} \end{aligned} \tag{4.5}$$

with $U^0 := \{u^0\}$. As the function evolves, elements in U given by $u^i < u^{k-\frac{L}{\Delta t}}$ will never be used again and can be removed. □

Numerically only the set U^k needs to be stored in memory for the current time point k and all points after. As discussed above, by its construction adaptive memory time step as an arithmetic sequence necessitates a shifting of the calculated window of points, and as such the entire history of the system needs to be stored in memory for the calculation at all time points. However, we can construct an adaptive memory as a power law, such that once we know what past history points need to be calculated, all other points in between will never need to be calculated. This then requires only enough memory to store the known and computed past intervals of the systems history, allowing the deallocation and recovery of much of the stored memory. This results in less memory requirements which translates into much faster computations.

Definition 4.3. Define a parameter η which determines the 'reset interval'. This represents the number of points in the past history to store in the current time point set U^k at each weight.

Definition 4.4. Define weighting sets W^i with elements w^i ordered in the same manner as the sets U^i.

Algorithm 4.1. Assume an adaptive memory time step power law scheme. Assume $w^0 = 1$. When there are more than η points in the set W^i of any given weight, define a subset of W^i as the elements in W^i of the given weight. Then from this subset, the weight of the least element is doubled, and the second lowest element is removed from W^i altogether. The time marching scheme is given as follows:

$$\begin{aligned} u_j^{k+1} &= u_j^k + \alpha \frac{\Delta_t^\gamma}{\Delta_x^2} \sum_{\{u_j^i \in U^k\}} \psi(\gamma, k-i) w^i \delta_j^i. \\ w^{k+1} &= 1 \\ W^{k+1} &:= \{w^i \in W^k\} + \{w^{k+1}\} \\ U^{k+1} &:= \{u^i \in U^k\} + \{u^{k+1}\} \end{aligned} \tag{4.6}$$

so that when the number N elements of a given weight is $N > \eta$, the algorithm is condensed.

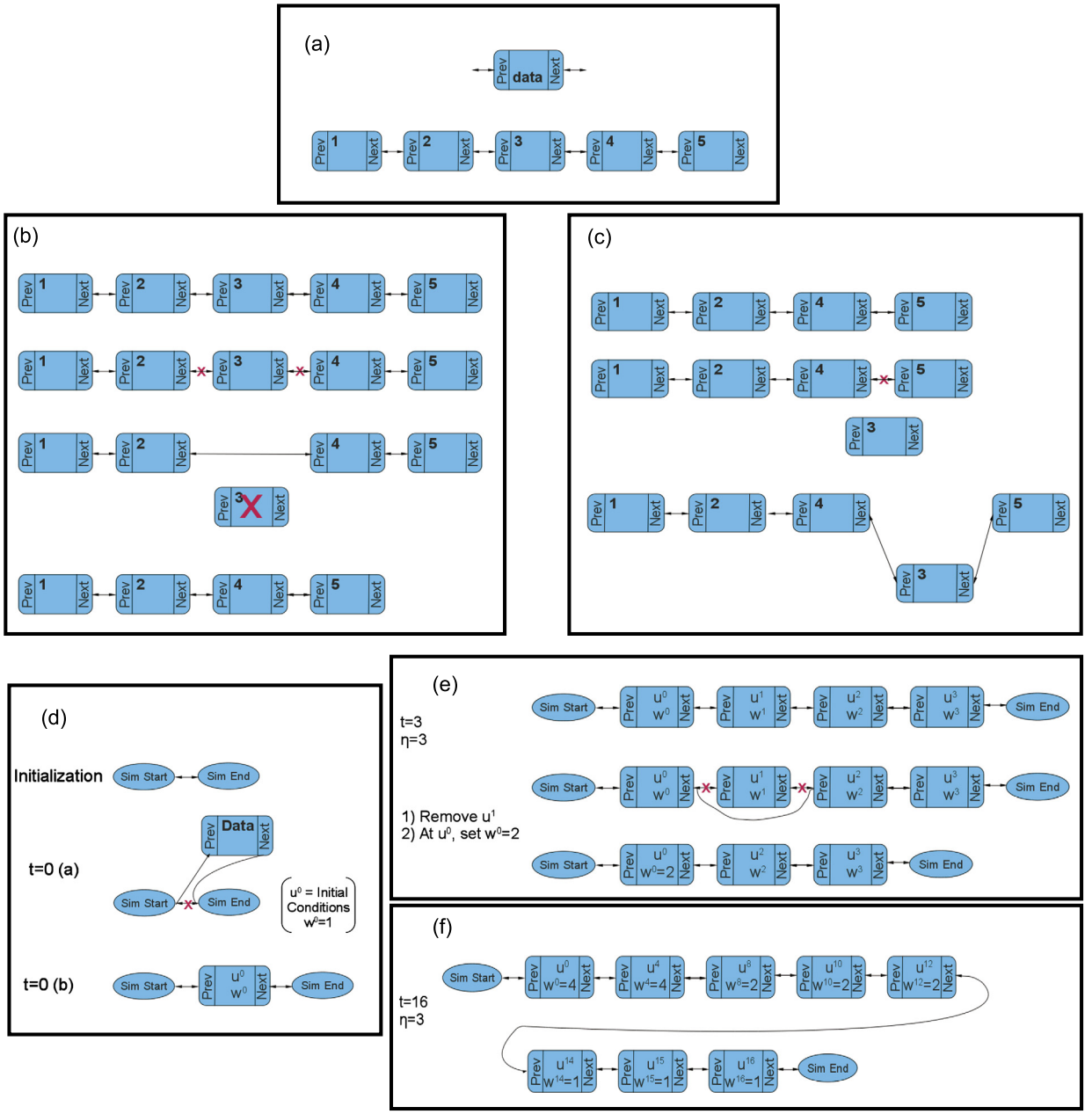

Fig. 5. Overview of the use of linked list data structures for the algorithmic numerical implementation of adaptive memory time step as a power law scheme. See the text for details.

4.2. Numerical implementation

From an applied perspective, to keep a well ordered list of points in U we make use of linked lists as the data structure. A linked list is one of the fundamental data structures used in computer programming. It consists of nodes that can contain data and their own data structures, and also pointers from each node to the next and/or previous nodes (Fig. 5(a)). This means one node can be removed from the set and deallocated from memory while the order of the set is maintained. In our case each node is an item u of the set U describing a time point necessary for the current time step, with the entire list representing all points u that make up the current iteration U^k. Once a time point u is no longer necessary for the simulation, it can be removed (Fig. 5(b)). New computed time points, u^{k+1} are added into the list (Fig. 5(c)). The data structure is initialized as illustrated in Fig. 5(d). At each time step, a new structure containing u^{k+1} and w^{k+1} is added onto the end of the list. When the elements of a specific weight ω grow larger than the limit η, the first two values of the weight ω are condensed into one value, with the weight doubling and one value being removed from the memory. We show as an

example the fourth step in a simulation when $\eta = 3$ (Fig. 5(e)). As this process iterates in time, one has a condensed list of only the time points that will be needed for the rest of the simulation in the list. For example, Fig. 5(f) shows the transition to the 17th step of a simulation.

5. A smart adaptive time step memory algorithm

Regular diffusion relies on the assumption of a Markovian process that is not necessarily present in natural systems. One approach to modeling these non-Markovian processes is using the fractional diffusion equation introduced above. Mathematically such methods have been around for decades but it is relatively recently that they have been applied to the natural sciences. Computationally efficient methods for numerically evaluating these equations are a necessity if fractional calculus models are to be applied to modeling real world physical and biological processes. It should be noted that while in this work the simulations were done in the subdiffusive regime for a simple point source, the methods we derive here are directly applicable to more complex sources or superdiffusion ($\gamma > 1$). However, complex fast-evolving past histories in the form of a forcing function ($f(u)$) or oscillations generated in a superdiffusion regime will result in much larger errors for both the short and adaptive memory methods. In the case of the adaptive memory method introduced in this paper this is due to its 'open-loop'-like algorithm that blindly increases the spacing between points in the summation as the calculation goes further back in time and which does not take into account the speed of evolution of the equation. Adaptive time approaches for regular differential equations often make the integration step size a function of the derivative, i.e., more closely spaced time steps are used when the function is oscillating quickly and cannot be averaged without loss of accuracy, and widely spaced time steps are used where the function is smooth. In the current implementation we have assumed that the past history function $\psi(\gamma, m)\delta_{i,j}^{k-m}$ is smooth. In this last section we extend the original implementation to develop a 'smart' 'closed-loop'-like algorithm where the step size of the backwards summation is dependent on the derivative of the past history function, i.e., a form of feedback. This optimizes the computational speed of the simulation while reducing the error due to the averaging of time points in the backwards summation, ultimately resulting in low errors for both high frequency forcing functions in the subdiffusion regime and for oscillations intrinsic to the superdiffusion regime.

5.1. Approximating the discrete Grünwald–Letnikov series as a continuous time integral

Our analytical approach for smart adaptive memory makes use of the continuous Grünwald–Letnikov integral. In this form, we can then define a minimum threshold error function based on the derivative that ensures that no period in the history of the system is misrepresented up to the defined error.

Recalling Eq. (2.29), the discretized form of the fractional diffusion equation, the summation term can be defined as a Riemann sum, which in the limit as $\Delta_t \to 0$ approaches a continuous time integral. The benefits of an integral formulation is that the backwards integration would be separate from a defined time grid for the series, and higher order methods for integrating, such as Simpson's rule, could be used. This would be impossible in discrete time since it is necessary to project and interpolate between points on the grid.

Defining a Riemann sum S over an interval x_1 to x_n,

$$S = \sum_{i=0}^{n} g(y_i)(x_i - x_{i-1}) \tag{5.1}$$

there is a correspondence between the discrete summation and the integral

$$\frac{u_{j,l}^{k+1} - u^k}{\Delta_t} = \Delta_t^{\gamma-1} \sum_{m=0}^{t/\tau} \psi(\gamma, m) f(u_{k-m})$$

such that,

$$\begin{aligned} g(y_i) &= \psi(\gamma, m) f(u_{k-m}) \\ i &= m \\ n &= t/\tau \\ x_i &= m = 0 \ldots n \end{aligned}$$

where the width of each segment is 1. As $\Delta_t \to 0$, this sum gets closer and closer to, and can be approximated by, the continuous time integral

$$\int_{\tau=0}^{t} \psi(\gamma, \tau/\Delta_t) f(u(t-\tau)) d\tau \tag{5.2}$$

which allows the discretized version to be rewritten in continuous form as

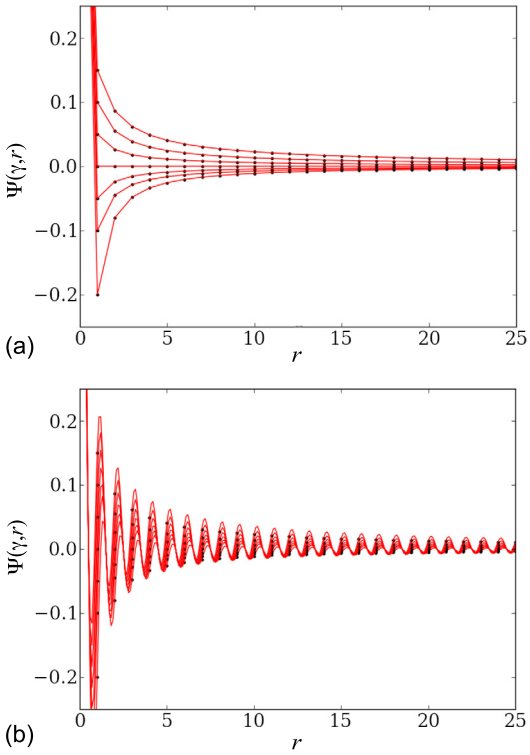

Fig. 6. Computing a continuous version of $\psi(\gamma, m)$ for various values of γ. In all subplots, $\gamma = (.85, .90\ldots, 1.10, 1.15)$ from the bottom trace to the top. Exact function ψ is denoted by discrete points. (a) $\Psi(\gamma, r)$ as a linear interpolation of the exact $\psi(\gamma, m)$. (b) Extending the definitions of $\psi(\gamma, m)$ to all r in the positive real domain.

$$\frac{u_{j,l}^{k+1} - u^k}{\Delta_t} = \Delta_t^{\gamma-1} \int_{\tau=0}^{t} \psi(\gamma, \tau/\Delta_t) f(u(t-\tau)) d\tau \tag{5.3}$$

The function $f(u(t-\tau))$ can be interpolated from the previously calculated values of u. The original definition of $\psi(\gamma, m)$ is only defined for $m \in \mathbb{N}_0$, and so needs to extend into the continuous domain. With an analytical continuous function representing ψ, one is then free to rediscretize the continuous integral in the most optimal way to solve the problem efficiently.

5.2. Extension of $\psi(\gamma, m)$ to the positive real domain

We are interested in a function $\Psi(\gamma, r)$ that is an extension of $\psi(\gamma, m)$ into the positive real domain and is defined such that for all $r \in \mathbb{N}_0$, $\psi(\gamma, r) = \psi(\gamma, m)$. We begin with a basic linear interpolation of $\psi(\gamma, m)$ over non-integer space, and the result is shown in Fig. 6(a) for various values of γ. While a linear interpolation provides a reasonable approximation of ψ, it is not a smooth function and it is a first order approximation that obviously does not work well for areas of ψ that have a high second derivative (e.g., $r \ll 1$, for $\gamma < 1$). Since we don't have the ability to increase the number of points we are basing our interpolation on, we consider other options to obtain a more accurate and smoother approximation.

Next, we consider simply extending the original definition of $\psi(\gamma, m)$ and expanding it to all points r by using the gamma function to re-express the factorial. With r extending to non-integer space in the positive real domain, and with the $(-1)^r$ term, we get an oscillating complex function. A plot of the real part of the function,

$$\Psi(\gamma, r) = Real\left\{\frac{(-1)^r \Gamma(2-\gamma)}{\Gamma(r+1)\Gamma(2-\gamma-r)}\right\}, \tag{5.4}$$

shows that this method results in a poor approximation of ψ due to the oscillatory behavior (Fig. 6(b)).

Another possibility for deriving a continuous function Ψ is to consider a rational polynomial function. One can rewrite the recursive series $\psi(\gamma, m)$ as a truncated approximation using the relation from Eq. (2.14):

$$\psi(\gamma, m) = -\psi(\gamma, m-1)\frac{2-\gamma-m}{m}$$
$$\psi(\gamma, 0) = 1$$

This can be written in terms of a finite product as

$$\psi(\gamma,m)=\frac{(-1)^m\Gamma(2-\gamma)}{m!\Gamma(2-\gamma-m)}=\cdots$$
$$-\psi(\gamma,m-1)\frac{2-\gamma-m}{m}=\prod_{n=1}^{m}\left[\frac{\gamma+n-2}{n}\right]=\prod_{n=1}^{m}\left[1+\frac{\gamma-2}{n}\right] \tag{5.5}$$

Then, using a transform to show that an infinite rational function will intersect all these points, define a rational function so that

$$\Psi(\gamma,r)=R_{\alpha,\beta}=\frac{P_\alpha}{Q_\beta} \tag{5.6}$$

where

$$R_{\alpha,\beta}=\frac{p_0+p_1r+p_2r^2\ldots p_\alpha r^\alpha}{q_0+q_1r+q_2r^2\ldots q_\beta r^\beta}. \tag{5.7}$$

As $(\alpha,\beta)\to\infty$, $\Psi(\gamma,r)$ will approach $\psi(\gamma,m)$ for all $r\in\mathbb{N}_0$.

An infinite rational expression, however, would be too costly to compute. One can truncate the expression however, and get a closed-form expression with a very close fit that approaches the analytical recursive $\psi(\gamma,m)$. As $m\to\infty$, $\psi\to 0$, which implies that $\beta>\alpha$ for this truncation.

Given the number of coefficients $p_0,\ldots,p_\alpha,q_0,\ldots,q_\beta$ and flexibility in choosing α and β, there are multiple possible solutions to Eq. (5.7). But for all cases we consider the basic set of constraints given by the system of equations:

$$\begin{aligned}\psi(\gamma,0) &= \Psi(\gamma,0)\\ \psi(\gamma,1) &= \Psi(\gamma,1)\\ \psi(\gamma,2) &= \Psi(\gamma,2)\\ &\cdots\\ \psi(\gamma,M) &= \Psi(\gamma,M)\end{aligned} \tag{5.8}$$

where M is an integer.

The values of α and β are also important considerations, since higher order polynomials will yield a more accurate approximation. Although the resulting function Ψ will increase in complexity along with accuracy, a powerful advantage of a $\psi\to\Psi$ transform that uses a finite rational polynomial function is that we will obtain a continuous version of the recursive function ψ, that is a closed-form expression.

No matter the exact method or transform used to obtain the new function $\Psi(\gamma,r)$, once derived, we can then directly insert it into the numerical implementation. We can drop all points in the past history function $(\Psi(\gamma,m)f(u_{k-m}))$ in the regions where the second derivative is below a certain threshold (i.e., where the function is slowly changing), and integrate the function on the resultant mesh using Eq. (5.3), with the substitution of the continuous $\Psi(\gamma,r)$, with $r=\tau/\Delta_t$:

$$\frac{u_{j,l}^{k+1}-u^k}{\Delta_t}=\Delta_t^{\gamma-1}\int_{\tau=0}^{t}\Psi(\gamma,\tau/\Delta_t)f(u(t-\tau))d\tau \tag{5.9}$$

As discussed before, this smart adaptive step extension to the original algorithm will allow us to minimize errors by using smaller integration step sizes when the past history function is quickly changing due to a fast-evolving external forcing function to the system, or oscillating behavior integral to the dynamics of the system itself. In addition, this approach will be able to take advantage of the large body of existing literature and numerical methods solutions packages for solving continuous equations.

Acknowledgements

The authors wish to thank Dr. Richard Magin and Dr. Igor Podlubny for helpful discussions during the course of this work. This work was supported by NIH NS054736 and ARO 63795EGII.

References

[1] B. Baeumer, D.A. Benson, M.M. Meerschaert, S.W. Wheatcraft, Subordinated advection–dispersion equation for contaminant transport, Water Resour. Res. 37 (6) (2001) 1543–1550.

[2] C. Chen, F. Liu, K. Burrage, Finite difference methods and a Fourier analysis for the fractional reaction–subdiffusion equation, Appl. Math. Comput. 198 (2) (2008) 754–769.

[3] C.M. Chen, F. Liu, I. Turner, V. Anh, Numerical schemes and multivariate extrapolation of a two-dimensional anomalous sub-diffusion equation, Numer. Algorithms (2009) 1–21.
[4] Albert Compte, Continuous time random walks on moving fluids, Phys. Rev. E 55 (6) (1997) 6821–6831.
[5] J.H. Cushman, T.R. Ginn, Fractional advection–dispersion equation: a classical mass balance with convolution-Fickian flux, Water Resour. Res. 36 (12) (2000) 3763–3766.
[6] Z.E.A. Fellah, C. Depollier, M. Fellah, Application of fractional calculus to the sound waves propagation in rigid porous materials: validation via ultrasonic measurements, Acta Acust. Acust. 88 (2002) 34–39.
[7] Neville J. Ford, A. Charles Simpson, The numerical solution of fractional differential equations: speed versus accuracy, Numer. Algorithms 26 (2001) 333–346.
[8] R. Gorenflo, F. Mainardi, D. Moretti, P. Paradisi, Time fractional diffusion: a discrete random walk approach, Nonlinear Dyn. 29 (1) (2002) 129–143.
[9] B.I. Henry, T.A.M. Langlands, S.L. Wearne, Fractional cable models for spiny neuronal dendrites, Phys. Rev. Lett. 100 (12) (2008) 128103.
[10] T.A.M. Langlands, B.I. Henry, The accuracy and stability of an implicit solution method for the fractional diffusion equation, J. Comput. Phys. 205 (2) (2005) 719–736.
[11] Jing-Rebecca Li, A fast time stepping method for evaluating fractional integrals, SIAM J. Sci. Comput. 31 (6) (2010) 4696–4714.
[12] F. Liu, P. Zhuang, V. Anh, I. Turner, K. Burrage, Stability and convergence of the difference methods for the space–time fractional advection–diffusion equation, Appl. Math. Comput. 191 (1) (2007) 12–20.
[13] María López-Fernández, Christian Lubich, Achim Schädle, Adaptive, fast, and oblivious convolution in evolution equations with memory, SIAM J. Sci. Comput. 30 (2) (2008) 1015–1037.
[14] C. Lubich, Convolution quadrature and discretized operational calculus I, Numer. Math. 52 (2) (1988) 129–145.
[15] Ch. Lubich, Discretized fractional calculus, SIAM J. Math. Anal. 17 (3) (1986) 704–719.
[16] Christian Lubich, Convolution quadrature and discretized operational calculus II, Numer. Math. 52 (1988) 413–425.
[17] Christian Lubich, Achim Schädle, Fast convolution for nonreflecting boundary conditions, SIAM J. Sci. Comput. 24 (1) (2002) 161–182.
[18] R. Magin, Fractional Calculus in Bioengineering, Critical Reviews in Biomedical Engineering, 2004.
[19] Richard L. Magin, Fractional Calculus in Bioengineering, Begell House Publishers, Jan 2006.
[20] R.L. Magin, M. Ovadia, Modeling the cardiac tissue electrode interface using fractional calculus, J. Vib. Control 14 (9–10) (2008) 1431.
[21] F. Mainardi, A. Mura, G. Pagnini, R. Gorenflo, Sub-diffusion equations of fractional order and their fundamental solutions, Math. Methods Eng. (2006) 20–48.
[22] B. Mathieu, P. Melchior, A. Oustaloup, C. Ceyral, Fractional differentiation for edge detection, Signal Process. 83 (11) (2003) 2421–2432.
[23] W. McLean, K. Mustapha, Convergence analysis of a discontinuous Galerkin method for a sub-diffusion equation, Numer. Algorithms 52 (1) (2009) 69–88.
[24] Monica R. Metea, Eric A. Newman, Calcium signaling in specialized glial cells, GLIA 54 (7) (2006) 650–655.
[25] R. Metzler, E. Barkai, J. Klafter, Anomalous diffusion and relaxation close to thermal equilibrium: a fractional Fokker–Planck equation approach, Phys. Rev. Lett. 82 (18) (1999) 3563–3567.
[26] R. Metzler, J. Klafter, The random walk's guide to anomalous diffusion: a fractional dynamics approach, Phys. Rep. 339 (1) (2000) 1–77.
[27] E.A. Newman, Propagation of intercellular calcium waves in retinal astrocytes and Müller cells, J. Neurosci. 21 (7) (2001) 2215–2223.
[28] I. Podlubny, Fractional Differential Equations, Academic Press, New York, 1999.
[29] I. Podlubny, A. Chechkin, T. Skovranek, Y.Q. Chen, B.M. Vinagre Jara, Matrix approach to discrete fractional calculus II: partial fractional differential equations, J. Comput. Phys. 228 (8) (2009) 3137–3153.
[30] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C, Cambridge Univ. Press, Cambridge, MA, USA, 1992.
[31] Achim Schädle, María López-Fernández, Christian Lubich, Fast and oblivious convolution quadrature, SIAM J. Sci. Comput. 28 (2) (2006) 421–438.
[32] N. Sebaa, Z.E.A. Fellah, W. Lauriks, C. Depollier, Application of fractional calculus to ultrasonic wave propagation in human cancellous bone, Signal Process. 86 (10) (2006) 2668–2677.
[33] K. Seki, M. Wojcik, M. Tachiya, Fractional reaction–diffusion equation, J. Chem. Phys. 119 (2003) 2165.
[34] C. Selhuber-Unkel, P. Yde, K. Berg-Sørensen, L.B. Oddershede, Variety in intracellular diffusion during the cell cycle, Phys. Biol. 6 (2009) 025015.
[35] E. Soczkiewicz, B. Krzywoustego, P. Gliwice, Application of fractional calculus in the theory of viscoelasticity, Mol. Quantum Acoustics 23 (2002) 397.
[36] V. Volterra, Leçons sur la théorie mathématique de la lutte pour la vie, Paris, France, 1931.
[37] D. Yu, M. Buibas, S.K. Chow, I.Y. Lee, Z. Singer, G.A. Silva, Characterization of calcium-mediated intracellular and intercellular signaling in the rMC-1 glial cell line, Cell. Mol. Bioeng. 2 (1) (2009) 144–155.
[38] S.B. Yuste, L. Acedo, On an explicit finite difference method for fractional diffusion equations, arXiv:cs/0311011, 2003.