Stability and Complexity Analyses of Finite Difference Algorithms for the Time-fractional Diffusion Equation

Nirupama Bhattacharya[1] and Gabriel A. Silva *[1,2,3]

[1]Department of Bioengineering, University of California, San Diego
[2]Department of Neurosciences, University of California, San Diego
[3]Center for Engineered Natural Intelligence, University of California, San Diego

Abstract. Fractional differential equations (FDEs) are an extension of the theory of fractional calculus. However, due to the difficulty in finding analytical solutions, there have not been extensive applications of FDEs until recent decades. With the advent of increasing computational capacity along with advances in numerical methods, there has been increased interest in using FDEs to represent complex physical processes, where dynamics may not be as accurately captured with classical differential equations. The time-fractional diffusion equation is an FDE that represents the underlying physical mechanism of anomalous diffusion. But finding tractable analytical solutions to FDEs is often much more involved than solving for the solutions of integer order differential equations, and in many cases it is not possible to frame solutions in a closed form expression that can be easily simulated or visually represented. Therefore the development of numerical methods is vital. In previous work we implemented the full 2D time-fractional diffusion equation as a Forward Time Central Space finite difference equation by using the Grünwald-Letnikov definition of the fractional derivative. In addition, we derived an adaptive time step version that improves on calculation speed, with some tradeoff in accuracy. Here, we explore and characterize stability of these algorithms, in order to define bounds on computational parameters that are crucial for performing accurate simulations. We also analyze the time complexity of the algorithms, and describe an alternate adaptive time step approach that utilizes a linked list implementation, which yields better algorithmic efficiency.

1 Introduction

Fractional calculus as a mathematical field has existed almost as long as classical integer-order calculus, much of the theory having been developed by Liouville, Laplace, Fourier, Euler, Lagrange, Riemann, and others [12, 8]. Analogous to ordinary and partial differential equations being used to model physical phenomena using classical calculus, fractional differential equations (FDEs) are an extension of the theory of fractional calculus. However, due to the difficulty in finding analytical solutions to FDEs, there have not been extensive applications of these equations until recent decades. With the advent of increasing computational capacity along with advances in numerical methods, there has been increased interest in using FDEs to represent complex physical processes, where dynamics may not be as accurately captured with classical differential equations.

Fractional differential equations have been useful for modeling such varied applications as advection-dispersion transport phenomena [1], wave propagation in bone and other rigid porous materials [14, 3], viscoelasticity [15], edge detection in image processing [10], and neurodynamics and modeling of signal processing in neuronal dendrites [5]. Richard Magin discusses a wide array of fractional calculus modeling applications in the field of bioengineering alone, including signaling across membranes, feedback control in neural systems, neurodynamics, capacitor and

*Corresponding author - G.S., Email: gsilva@ucsd.edu

dielectric models, the behavior of viscoelastic materials, circuit models of electrode interfaces, cell biomechanics, and electrochemistry, among many other applications [8, 9].

In particular, the time-fractional diffusion equation is an FDE that represents the underlying physical mechanism of anomalous diffusion. In the same way that the classical diffusion equation can be derived from statistical analysis of particle interactions, one can also show how the micromolecular behavior of particles can, under a different set of fundamental statistical assumptions, be represented on a macro scale in two dimensions by the fractional diffusion equation of the form [2]:

$$\frac{\partial^\gamma u(\vec{x},t)}{\partial t^\gamma} = \alpha \frac{\partial^2 u(\vec{x},t)}{\partial x^2} + \beta \frac{\partial^2 u(\vec{x},t)}{\partial y^2} \tag{1.1}$$

where $\gamma < 1$ denotes the subdiffusion regime, $\gamma = 1$ denotes classical Gaussian diffusion, $\gamma > 1$ denotes the superdiffusion regime, and α and β represent the diffusion coefficients in the x and y directions, respectively (in $\frac{spatial\ unit^2}{time\ unit^\gamma}$).

Finding tractable analytical solutions to FDEs like Eq. 1.1 is often much more involved than solving for the solutions of integer order differential equations, and in many cases it is not possible to frame solutions in a closed form expression that can be easily simulated or visually represented. Therefore the development of numerical methods is vital. Over the last decade there have been numerous algorithms developed for FDEs like the time-fractional diffusion equation, and there is often a tradeoff between calculation speed and efficiency, complexity, stability, and accuracy. In [7] we walk through the discretization of Eq. 1.1 into the full two-dimensional fractional FTCS (Forward Time Central Space) finite difference equation, making use of the Grünwald-Letnikov definition of the fractional derivative, which allows us to use a discrete summation in our numerical algorithm. We also derive an adaptive time step algorithm which builds on the foundation of the full 2D fractional FTCS equation but improves calculation speed and efficiency, while maintaining accuracy [7]. However, both the FTCS equation and adaptive algorithms are explicit methods with a limited stability regime. Here we fully explore and characterize the stability of these two finite difference schemes in order to define bounds on important computational parameters like time step and spatial discretization and grid size; selecting these parameters appropriately is crucial for performing accurate simulations.

There are numerous approaches to analyzing the stability of finite difference schemes, including modified wavenumber analysis, matrix eigenvalue analysis, and other more mathematically complex methods derived from stability definitions involving matrix norms. Several methods, including matrix analysis, are widely applicable but sometimes impossible to approach analytically or without the aid of iterative procedures. However, since our numerical algorithms are both fully discretized in space and time, and linear with constant coefficients (in their homogeneous versions), we show in the next section that an appropriate choice is a von Neumann stability analysis. We assume the general solution at an arbitrary location and time step is a linear combination of special solutions, represented by a function that is separable in its temporal and spatial variables. We find that our stability analysis and expressions of parameter bounds agree very well with our results, where we simulate several examples with various parameter combinations.

In addition to stability analysis, a complexity analysis is another important metric used to characterize the efficiency of numerical algorithms, as measured by execution time of the algorithm as a function of some input variable. In the case of the algorithms described in this paper, we are interested in how the execution time varies with N_x, N_y(grid size of the simulation in the x and y directions, respectfully), or N, the number of timesteps. The relationship between execution time and input variable is usually given in Big-O notation using a worst case scenario, or average case scenario which is often a better reflection of the average behavior of the algorithm run time. In Section 3, we explore the time complexity of the same two algorithms for which we complete the stability analysis, and in addition, analyze an alternate version of the adaptive timestep algorithm that involves a linked list implementation (introduced in [7]) that yields better algorithmic efficiency. For mathematical simplicity we analyze our algorithms according to worst case scenarios and interpret the results as a proof of bounding behavior - the actual run time of the algorithms may be more efficient, but never less efficient. We also present simulated data that verifies our theoretical complexity analyses.

2 Stability Analysis of the Two-dimensional Fractional FTCS Discretization

2.1 Full Implementation

We begin by considering the circumstances under which the 2D fractional FTCS finite difference equation is stable, and consider the advantages and disadvantages of several different analytical approaches. We restate the FTCS discretized

equation in two dimensions (see [7, 2] for more details):

$$\begin{aligned} u_{j,l}^{n+1} - u_{j,l}^{n} &= \Delta_t^{\gamma} \sum_{m=0}^{n} \psi(\gamma, m) \left(\frac{\alpha}{\Delta x^2} \delta x_{j,l}^{n-m} + \frac{\beta}{\Delta y^2} \delta y_{j,l}^{n-m} \right) \\ \delta x_{j,l}^{n} &= u_{j+1,l}^{n} - 2u_{j,l}^{n} + u_{j-1,l}^{n} \\ \delta y_{j,l}^{n} &= u_{j,l+1}^{n} - 2u_{j,l}^{n} + u_{j,l-1}^{n} \end{aligned} \tag{2.1}$$

where Δ_t is the time step, and Δx and Δy are spatial grid discretizations in the x and y directions. Our spatial range is

$$u_{j,l}^{n} \mid j = 0, 1, 2, ..., N_x - 1;\ l = 0, 1, 2, ..., N_y - 1$$

and

$$u_{j,l}^{n} \mid j = 0, N_x - 1;\ l = 0, N_y - 1$$

denotes the boundary points. $\psi(\gamma, m)$, which we refer to as a 'memory function', represents a binomial term $(-1)^m \binom{1-\gamma}{m}$, and results from the use of the Grünwald-Letnikov definition of the fractional derivative, as discussed in [7, 13, 2].

2.1.1 Matrix Stability Analysis

Matrix stability analysis is a method of analysis that can be applied to any problem without particular assumptions or constraints on coefficients, and includes the effects of boundary conditions [11]. We would like to reformulate Eq. 2.1 into a matrix equation in the form $W_{n+1} = L_n W_n + f_n$ where the W matrices hold u values at times $n+1$ and n, the L_n matrix represents the difference operator at time n, and f_n is a constant vector that takes into account boundary conditions at time step n (here we assume boundary conditions are known for all timesteps).

For example, for $n = 0$, Eq. 2.1 becomes:

$$u_{j,l}^{1} = u_{j,l}^{0} + \Delta_t^{\gamma} \psi(\gamma, 0) \left(\frac{\alpha}{\Delta x^2} \delta x_{j,l}^{0} + \frac{\beta}{\Delta y^2} \delta y_{j,l}^{0} \right)$$

Then the matrix equation is formulated as follows. We order grid values u into a vector (excluding the boundary points) such that W at some time step n is

$$W_n = \begin{bmatrix} u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ \vdots \\ u_{1,N_y-2} \\ \vdots \\ u_{N_x-2,1} \\ u_{N_x-2,2} \\ \vdots \\ u_{N_x-2,N_y-2} \end{bmatrix}^n \tag{2.2}$$

For $n = 0$ our matrix equation now becomes $W_1 = L_0 W_0 + f_0$ where the matrix L_0 accounts for interior non-boundary

grid points and consists of block matrices T_0 (a tridiagonal matrix) and R_0 (a diagonal matrix):

$$L_0 = \begin{bmatrix} T_0 & R_0 & 0 & \cdots & 0 \\ R_0 & T_0 & \ddots & \ddots & \vdots \\ 0 & R_0 & \ddots & R_0 & 0 \\ \vdots & \ddots & \ddots & T_0 & R_0 \\ 0 & \cdots & 0 & R_0 & T_0 \end{bmatrix}$$

$$T_0 = \begin{bmatrix} 1-2r_x\psi(\gamma,0)-2r_y\psi(\gamma,0) & r_y\psi(\gamma,0) & 0 & \cdots & 0 \\ r_y\psi(\gamma,0) & & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & & 0 \\ \vdots & \ddots & & & r_y\psi(\gamma,0) \\ 0 & \cdots & 0 & r_y\psi(\gamma,0) & 1-2r_x\psi(\gamma,0)-2r_y\psi(\gamma,0) \end{bmatrix}$$

$$R_0 = \begin{bmatrix} r_x\psi(\gamma,0) & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & r_x\psi(\gamma,0) \end{bmatrix}$$

where $r_x = \frac{\alpha\Delta_t^\gamma}{\Delta x^2}$ and $r_y = \frac{\beta\Delta_t^\gamma}{\Delta y^2}$. L_0 is size $(N_x-2)(N_y-2)$ by $(N_x-2)(N_y-2)$. T_0 and R_0 are each of size (N_y-2) by (N_y-2). f_0 takes into account the boundary points and is a column vector of constants.
For $n=1$ we now must take into account multiple past time points, and our matrix equation becomes

$$W_2 = L_1W_0 + L_0W_1 + f_{01}$$

$$L_1 = \begin{bmatrix} T_1 & R_1 & 0 & \cdots & 0 \\ R_1 & T_1 & \ddots & \ddots & \vdots \\ 0 & R_1 & \ddots & R_1 & 0 \\ \vdots & \ddots & \ddots & T_1 & R_1 \\ 0 & \cdots & 0 & R_1 & T_1 \end{bmatrix}$$

$$T_1 = \begin{bmatrix} -2r_x\psi(\gamma,1)-2r_y\psi(\gamma,1) & r_y\psi(\gamma,1) & 0 & \cdots & 0 \\ r_y\psi(\gamma,1) & & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & & 0 \\ \vdots & \ddots & & & r_y\psi(\gamma,1) \\ 0 & \cdots & 0 & r_y\psi(\gamma,1) & -2r_x\psi(\gamma,1)-2r_y\psi(\gamma,1) \end{bmatrix}$$

$$R_1 = \begin{bmatrix} r_x\psi(\gamma,1) & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & r_x\psi(\gamma,1) \end{bmatrix}$$

and where f_{01} is a column vector that takes into account the boundary conditions from timesteps 0 and 1. We can repurpose this multistep matrix equation into a single step equation:

$$W_{combined,2} = L_{combined,1}W_{combined,1} + f_{01} \text{ with}$$
$$W_{combined,2} = \begin{bmatrix} W_2 \\ W_1 \end{bmatrix}, W_{combined,1} = \begin{bmatrix} W_1 \\ W_0 \end{bmatrix}, L_{combined,1} = \begin{bmatrix} L_0 & L_1 \\ I & 0 \end{bmatrix}$$

and I is simply the identity matrix. For $n=2$ the complexity of the matrices involved continues to grow with

$$W_{combined,3} = L_{combined,2}W_{combined,2} + f_{012}$$
$$\begin{bmatrix} W_3 \\ W_2 \\ W_1 \end{bmatrix} = \begin{bmatrix} L_0 & L_1 & L_2 \\ I & 0 & 0 \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} W_2 \\ W_1 \\ W_0 \end{bmatrix} + f_{012}$$

where $L_0, L_1, L_2 ...$ are block matrices which consist of tridiagonal and diagonal sub-matrices, as demonstrated in previous steps. In general, we have

$$W_{combined,n+1} = L_{combined,n} W_{combined,n} + f_{0...n} \tag{2.3}$$

$$W_{combined,n+1} = \begin{bmatrix} W_{n+1} \\ \vdots \\ \vdots \\ \vdots \\ W_1 \end{bmatrix}$$

$$L_{combined,n} = \begin{bmatrix} L_0 & L_1 & \cdots & L_{n-1} & L_n \\ I & 0 & & \cdots & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & I & 0 \end{bmatrix}$$

$$L_n = \begin{bmatrix} T_n & R_n & 0 & \cdots & 0 \\ R_n & T_n & \ddots & \ddots & \vdots \\ 0 & R_n & \ddots & R_n & 0 \\ \vdots & \ddots & \ddots & T_n & R_n \\ 0 & \cdots & 0 & R_n & T_n \end{bmatrix}$$

$$T_n = \begin{bmatrix} -2r_x\psi(\gamma,n) - 2r_y\psi(\gamma,n) & r_y\psi(\gamma,n) & 0 & \cdots & 0 \\ r_y\psi(\gamma,n) & & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & & 0 \\ \vdots & \ddots & & & r_y\psi(\gamma,n) \\ 0 & \cdots & 0 & r_y\psi(\gamma,n) & -2r_x\psi(\gamma,n) - 2r_y\psi(\gamma,n) \end{bmatrix}, \text{for } n \geq 1$$

$$R_n = \begin{bmatrix} r_x\psi(\gamma,n) & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & r_x\psi(\gamma,n) \end{bmatrix}$$

The goal with matrix stability analysis is to show that for a matrix problem Eq. 2.3, the spectral radius $\rho(L_{combined,n})$ of the difference matrix $L_{combined,n}$, (i.e., the modulus of its largest eigenvalue) is bounded, usually by 1 [4]. This involves finding the maximum eigenvalue for $L_{combined,n}$, for all timesteps n, and proving that its modulus is ≤ 1. The matrix $L_{combined,n}$ is quite complex and is not in the form of a special matrix where there are known analytical algorithms to find the eigenvalues. This makes the process of finding the eigenvalues very complicated and most likely beyond the realm of an analytic solution, so we consider this to be outside the scope of this paper, and conclude that despite the difficulty of finding the spectral radius, a major advantage of this method would be its generality - it could be applied to any set of boundary conditions, or schemes with non-constant coefficients.

2.1.2 Von Neumann Analysis

As we saw in the previous section, matrix stability analysis for complex algorithms with no simplifying assumptions, requires methods for finding eigenvalues, possibly including iterative numerical schemes. However, our finite difference scheme in Eq. 2.1 has the advantage of being linear with constant coefficients (i.e. α, β don't depend on t or $\vec{x}$) on a uniformly spaced grid. With an additional simplifying assumption of periodic boundary conditions, we can make use of von Neumann stability analysis and Fourier series expansion. While von Neumann analysis doesn't take into

account boundary conditions in the same way matrix stability analysis does, we make the reasonable assumption that stability issues are mostly affected by discretization of differential equations inside the domain, and minimally affected by boundary conditions [11]. In the majority of modeling and simulation problems in engineering, we consider this to be a valid and applicable simplifying assumption.

Given our linear scheme with constant coefficient and periodic boundary conditions, we assume the solution of scheme 2.1 at a particular location and time point, can be expressed as a discrete finite complex Fourier series expansion:

$$u_{j,l}^n = \sum_{d=0}^{N_y-1} \sum_{c=0}^{N_x-1} A_{cd}^n e^{ik_c x_j} e^{ik_d y_l} \tag{2.4}$$

where A_{cd}^n are the Fourier mode amplitudes, $x_j = j\Delta x,\ j = 0,1,...,N_x-1, y_l = l\Delta y,\ l = 0,1,...,N_y-1$, and k_c and k_d are the spatial wavenumbers in the x and y directions, respectively. The wavenumbers represent spatial frequency and are related to wavelength by $\lambda_x = \frac{2\pi}{k_c},\ \lambda_y = \frac{2\pi}{k_d}$. In the context of Fourier modes and stability analysis, we are interested in suppressing (or as we shall see later in this section, limiting the amplification factor of) the high frequency modes (represented by wavelengths on the order of grid sizes Δx and Δy, and wavenumber values away from 0) because these modes have a low probability of corresponding to any real features of the true solution, and usually represent noise arising from finite precision arithmetic for numerical calculations. The time evolution of each Fourier mode is determined by the same time marching scheme as the full solution $u_{j,l}^n$. And since no Fourier mode can be allowed to increase unbounded in time, we can single out any arbitrary harmonic $A_{cd}^n e^{ik_c x_j} e^{ik_d y_l}$ and substitute this form into Eq. 2.1 [6]. We utilize the relationships $x_{j+1} = x_j + \Delta x,\ x_{j-1} = x_j - \Delta x,\ y_{l+1} = y_l + \Delta y$ and $y_{l-1} = y_l - \Delta y$, as well as Euler's formula and half angle trigonometric identities, to obtain

$$A_{cd}^{n+1} - A_{cd}^n + \Delta_t^\gamma \left(\frac{4\alpha}{\Delta x^2} sin^2\left(\frac{k_c \Delta x}{2}\right) + \frac{4\beta}{\Delta y^2} sin^2\left(\frac{k_d \Delta y}{2}\right)\right) \sum_{m=0}^{n} \psi(\gamma, m) A_{cd}^{n-m} = 0 \tag{2.5}$$

If we make the assumption that A_{cd} is independent of time step, we can write

$$A_{cd}^{n+1} = \sigma_{cd} A_{cd}^n \tag{2.6}$$

where σ_{cd} is known as the amplification factor that gives the growth or decay for Fourier mode (c,d). We require $|\sigma_{cd}| \leq 1$ for all Fourier modes, or else some modes will be amplified at every time step and dominate the solution. We divide Eq. 2.6 by A_{cd}^n and rearrange to get

$$\sigma_{cd} = 1 - \Delta_t^\gamma \left(\frac{4\alpha}{\Delta x^2} sin^2\left(\frac{k_c \Delta x}{2}\right) + \frac{4\beta}{\Delta y^2} sin^2\left(\frac{k_d \Delta y}{2}\right)\right) \sum_{m=0}^{n} \psi(\gamma, m) \sigma_{cd}^{-m}$$

We require $|\sigma_{cd}| \leq 1$ for all values of σ_{cd}, which are the roots of this polynomial. We can also see that the difficulty of solving for the roots increases as time progresses and n increases. Past the first few timesteps, solving for the roots of the polynomial requires iterative matrix procedures and does not result in a tractable analytical solution from which we can find the bounds on the relationship between $\alpha, \beta, \Delta t, \Delta x$, and Δy. Instead of solving for roots, we will approach the problem by considering 'worst-case' scenarios to simplify our expression, and proceed in a similar manner as Yuste and Acedo's treatment of the analogous one-dimensional case [16]. Consider the bound on the parameters resulting from the case $\sigma_{cd} = 1$:

$$0 \geq \Delta_t^\gamma \left(\frac{4\alpha}{\Delta x^2} sin^2\left(\frac{k_c \Delta x}{2}\right) + \frac{4\beta}{\Delta y^2} sin^2\left(\frac{k_d \Delta y}{2}\right)\right) \sum_{m=0}^{n} \psi(\gamma, m) \tag{2.7}$$

The summation term and parameters are always greater than 0, and therefore Eq. 2.7 can only hold true if both wavenumbers are 0 (a constant function). Therefore, we consider the other 'worst-case' scenario, where $\sigma_{cd} = -1$:

$$2 = \Delta_t^\gamma \left(\frac{4\alpha}{\Delta x^2} sin^2\left(\frac{k_c \Delta x}{2}\right) + \frac{4\beta}{\Delta y^2} sin^2\left(\frac{k_d \Delta y}{2}\right)\right) \sum_{m=0}^{n} \psi(\gamma, m) (-1)^m \tag{2.8}$$

From Eq. 2.8 we can infer the inequality bounding the parameters as

$$\Delta_t^\gamma \left(\frac{4\alpha}{\Delta x^2} sin^2\left(\frac{k_c \Delta x}{2}\right) + \frac{4\beta}{\Delta y^2} sin^2\left(\frac{k_d \Delta y}{2}\right)\right) \leq \frac{2}{\sum_{m=0}^{n} \psi(\gamma, m) (-1)^m} = B_1(\gamma, n) \tag{2.9}$$

We can see that the bounded value is dependent on the time step n but the dependence is very weak. Yuste and Acedo demonstrate that in the limit $n \to \infty$, the summation term in Eq. 2.9 involving the memory function, is an alternating converging series; as n increases, B_1 oscillates but clearly converges to an equilibrium value [16]. Therefore we can approximate $\sum_{m=0}^{n} \psi(\gamma,m)\,(-1)^m$ by taking the limit as $n \to \infty$. In [7] we originally derived the 2D FTCS full finite difference equation and adaptive memory algorithms by using the Grünwald-Letnikov definition to make a first-order approximation of the Riemann-Liouville fractional derivative operator [13]. And in the first-order approximation, the memory function $\psi(\gamma,m)$ is defined as $(-1)^m \begin{pmatrix} 1-\gamma \\ m \end{pmatrix}$. Podlubny ([13]) shows that this expression can also be defined as the coefficients of the power series for the function $(1-z)^{\,1-\gamma}$:

$$(1-z)^{1-\gamma} = \sum_{m=0}^{\infty} (-1)^m \begin{pmatrix} 1-\gamma \\ m \end{pmatrix} z^m = \sum_{m=0}^{\infty} \psi(\gamma,m) z^m \tag{2.10}$$

Applying this identity to $B_1(\gamma,n)$ and considering the limit $n \to \infty$, we can derive

$$B_2(\gamma) = 2^{\gamma} \tag{2.11}$$

As Yuste and Acedo describe in their considerations of stability in the one-dimensional case, we could consider the second-order approximation of the Riemann-Liouville definition by making use of a different set coefficients to define our memory function. This option would result in a slightly lower bound on our parameters, but for the purposes of estimating appropriate parameter values, we consider the first-order approximation to be sufficient.

From (2.9) we can see that we have three degrees of freedom in parameters that we are free to choose for our simulations : Δ_t, Δx^2, Δy^2 . To determine the most conservative bounds on those degrees of freedom (minimum values for grid size and maximum value for time step), we assume the maximum possible value for the trigonometric terms, $sin^2\left(\frac{k_c \Delta x}{2}\right) = sin^2\left(\frac{k_d \Delta y}{2}\right) = 1$. The resulting inequality determining parameter bounds is

$$\Delta_t^{\gamma}\left(\frac{4\alpha}{\Delta x^2} + \frac{4\beta}{\Delta y^2}\right) \leq B_2(\gamma)$$

If we assume a simple case with a square grid ($\Delta x = \Delta y = \Delta$) and equal diffusion coefficients in both spatial directions ($\alpha = \beta$), the expression becomes

$$r = \frac{\alpha \Delta t^{\gamma}}{\Delta^2} \leq \frac{B_2(\gamma)}{8} = 2^{\gamma-3} = B_{full}(\gamma) \tag{2.12}$$

As a final check, when we set $\gamma = 1$ (classical diffusion), we recover the well known traditional bound $r \leq \frac{1}{4}$ that results from the classical 2D diffusion equation using the FTCS discretization.

2.1.3 Error Propagation

An alternative approach to stability is to consider roundoff error from floating point or finite precision arithmetic, and ensure that this does not propagate in time by requiring error $\left|\varepsilon_{j,l}^{n+1}\right| \leq \left|\varepsilon_{j,l}^{n}\right|$. A mathematical derivation based on this interpretation would progress in much the same manner as in Section 2.1.2 and result in similar parameter boundaries.

2.2 Adaptive Memory Algorithm

As discussed in [7], while the full implementation represented by Eq. 2.1 is an accurate finite difference approximation to the solution of Eq. 1.1, using the Grünwald-Letnikov definition of the fractional derivative involves a summation that takes into account the entire past history of the simulation. As the simulation progresses, calculating the summation term becomes increasingly cumbersome, time consuming, and memory intensive. Therefore we have developed an adaptive memory method (introduced in [7]) that improves on computational efficiency while maintaining a level of accuracy comparable to the original full implementation. This is achieved by recognizing that the further back a time point is in the history of the simulation, the less it contributes to the calculation of the solution at the next time point. Therefore we sample the history of the system more frequently for recent time points (which contribute more to the solution at the next time step) and less often for time points further back in the history of the system, weighted by an appropriate amount to compensate for less frequent sampling; this approach significantly reduces actual computational

time. For smooth functions, the convolution of $\psi(\gamma,m)$ and the function in question changes slowly enough that the weighting of the sampled time points compensates for the less frequent sampling, and maintains overall accuracy. The motivation and derivation of this adaptive algorithm is discussed in further detail in [7]. Here we reproduce a modified version of the algorithm and recognize that analyzing stability requires only a few modifications from the process described in the last section.

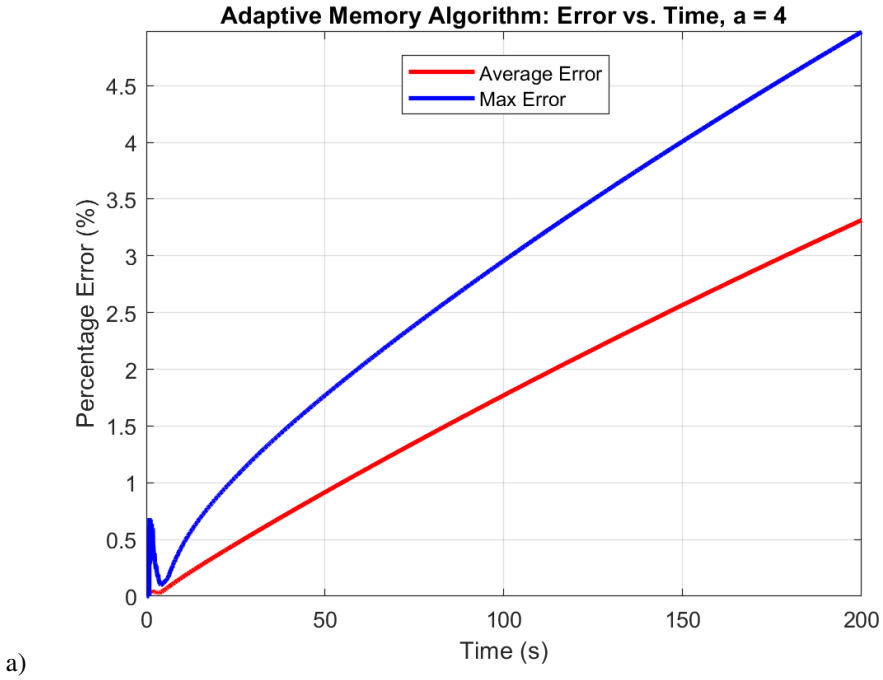
$$\begin{aligned}
\frac{u_{j,l}^{n+1}-u_{j,l}^{n}}{\Delta_t^{\gamma}} &= \overbrace{\sum_{m=0}^{a}\psi(\gamma,m)\left(\frac{\alpha}{\Delta x^2}\delta x_{j,l}^{n-m}+\frac{\beta}{\Delta y^2}\delta y_{j,l}^{n-m}\right)}^{R1} \\
&+\sum_{s=2}^{s_{max}(n)}\left\{\overbrace{\sum_{\eta=1}^{\eta_{max}(s,n)}(2s-1)\,\psi\left(\gamma,M\left(s,\eta\right)\right)\left(\frac{\alpha}{\Delta x^2}\delta x_{j,l}^{n-M(s,\eta)}+\frac{\beta}{\Delta y^2}\delta y_{j,l}^{n-M(s,\eta)}\right)}^{R2}\right. \\
&\left.+\overbrace{\sum_{p=M(s,\eta_{max})+s}^{min(a^s,n)}\psi(\gamma,p)\left(\frac{\alpha}{\Delta x^2}\delta x_{j,l}^{n-p}+\frac{\beta}{\Delta y^2}\delta y_{j,l}^{n-p}\right)}^{R3}\right\} \qquad (2.13)\\
M\left(s,\eta\right) &= a^{s-1}+(2s-1)\,\eta-s+1 \\
\eta_{max}\left(s,n\right) &= min\left(\left\lfloor\frac{a^s-a^{s-1}}{2s-1}\right\rfloor,\left\lfloor\frac{n-a^{s-1}}{2s-1}\right\rfloor\right) \\
\delta x_{j,l}^{n} &= u_{j+1,l}^{n}-2u_{j,l}^{n}+u_{j-1,l}^{n} \\
\delta y_{j,l}^{n} &= u_{j,l+1}^{n}-2u_{j,l}^{n}+u_{j,l-1}^{n} \qquad (2.14)
\end{aligned}$$

where a is the predefined base interval, $\lfloor\ \rfloor$ denotes the floor function, and s_{max} is determined by the current time step n such that $a^{s_{max}-1}+1\leq n\leq a^{s_{max}}$. The terms $R1,R2,R3$ are referred to in more detail during the complexity analysis in later sections.

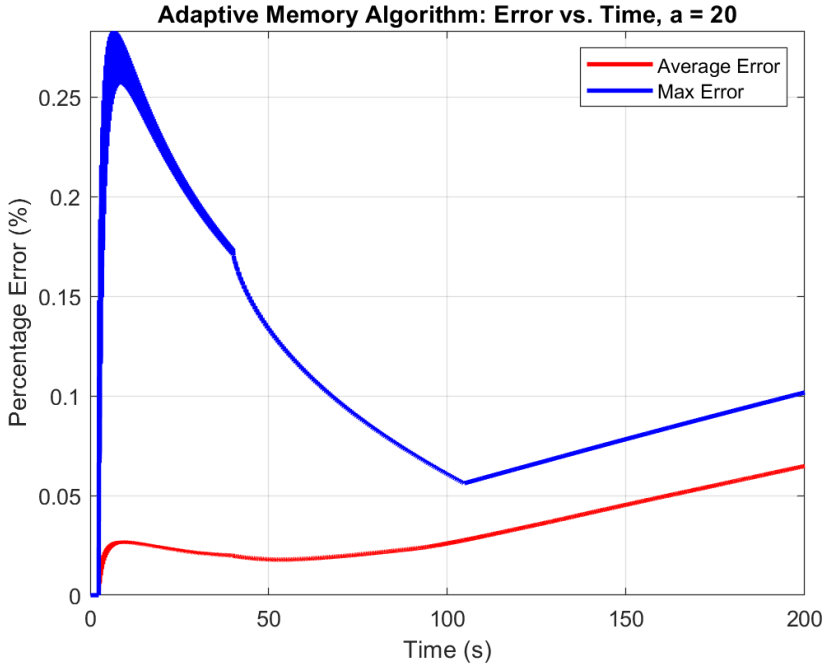
There are various ways to interpret how a is translated into a numerical algorithm, depending on whether one assumes that it is with respect to m (mathematical series beginning at 0), time step n (which may need to be adjusted to begin from index 0 or 1, depending on the programming language), or time t. For example, if one assumes a is a time, we can define $a_{effective}=a/dt$ where a is converted to a time step. For $dt<1$ this has the same effect as increasing a and interpreting it with respect to time step n. The larger the value of a, the more accurate the scheme presented in Eq. 2.13, when compared against the full algorithm in Eq. 2.1. Fig. 2.1 shows the error plots where the adaptive step algorithm in Eq. 2.13 is compared against the full implementation (the reference case) Eq. 2.1, for the adaptive step parameter values $a=4$ and $a=20$. Note that the initial error curve early in the simulation (before the dip and monotonic rise) is a function of a.

As with the full implementation, we apply a von Neumann stability analysis where we assume the solution is separable in the form of Eq. 2.4, substitute into Eq. 2.13 and simplify in the same fashion as described in Section 2.1.2 to yield

$$\begin{aligned}
0 &= A_{cd}^{n+1}-A_{cd}^{n}+\Delta_t^{\gamma}\left(\frac{4\alpha}{\Delta x^2}sin^2\left(\frac{k_c\Delta x}{2}\right)+\frac{4\beta}{\Delta y^2}sin^2\left(\frac{k_d\Delta y}{2}\right)\right)\left\{\sum_{m=0}^{a}\psi\left(\gamma,m\right)A_{cd}^{n-m}\right. \\
&\left.+\sum_{s=2}^{s_{max}(n)}\left(\sum_{\eta=1}^{\eta_{max}(s,n)}(2s-1)\,\psi\left(\gamma,M(s,\eta)\right)A_{cd}^{n-M(s,\eta)}+\sum_{p=M(s,\eta_{max})+s}^{min(a^s,n)}\psi\left(\gamma,p\right)A_{cd}^{n-p}\right)\right\} \qquad (2.15)
\end{aligned}$$



a)



b)

Figure 2.1: **Effect of a on Accuracy.** **a)** With adaptive step parameter $a = 4$, we see the error (compared against the full 2D discretization defined in Eq. 2.1) grows significantly. **b)** For $a = 20$, the error remains consistently below 0.3%, even as long as 200 s into the simulation. The tradeoff for this increase in accuracy is a longer computation time. For both cases the following parameters are used: $\gamma = 0.6$, $\alpha = \beta = 50\ \frac{units^2}{s^\gamma}$, $dt = 0.1s$, $\Delta x = \Delta y = 10\ units$, $N_x = N_y = 20$.. We include both maximum error at each timepoint, and average error over the whole solution, at each timepoint.

As before, we once again assume $A_{cd}^{n+1} = \sigma_{cd} A_{cd}^{n}$ to get the following expression:

$$\begin{aligned}\sigma_{cd} &= 1 - \Delta_t^{\gamma}\left(\frac{4\alpha}{\Delta x^2} sin^2\left(\frac{k_c \Delta x}{2}\right) + \frac{4\beta}{\Delta y^2} sin^2\left(\frac{k_d \Delta y}{2}\right)\right)\left[\sum_{m=0}^{a} \psi(\gamma, m)\sigma_{uv}^{-m}\right. \\ &+ \sum_{s=2}^{s_{max}(n)}\left\{\sum_{\eta=1}^{\eta_{max}(s,n)} (2s-1)\,\psi\left(\gamma, M\left(s,\eta\right)\right)\sigma_{cd}^{-M(s,\eta)}\right. \\ &\left.\left. + \sum_{p=M_(s,\eta_{max})+s}^{min(a^s,n)} \psi(\gamma, p)\sigma_{cd}^{-p}\right\}\right] \end{aligned} \tag{2.16}$$

Again, for stability we require that $\mid \sigma_{cd} \mid < 1$ for all values of σ. If we assume the "worst-case" scenario ($\sigma_{cd} = -1$), we can infer the following inequality to provide bounds on the time step and spatial steps

$$B_1(\gamma, n, a) = \frac{2}{\Xi} \geq \Delta_t^{\gamma}\left(\frac{4\alpha}{\Delta x^2} sin^2\left(\frac{k_c \Delta x}{2}\right) + \frac{4\beta}{\Delta y^2} sin^2\left(\frac{k_d \Delta y}{2}\right)\right) \tag{2.17}$$

$$\begin{aligned}\Xi &= \sum_{m=0}^{a} \psi(\gamma, m)\,(-1)^m \\ &+ \sum_{s=2}^{s_{max}(n)}\left\{\sum_{\eta=1}^{\eta_{max}(s,n)} (2s-1)\,\psi\left(\gamma, M(s,\eta)\right)(-1)^{M(s,\eta)}\right. \\ &\left. + \sum_{p=M(s,\eta_{max}+s)}^{min(a^s,n)} \psi(\gamma, p)\,(-1)^p\right\}\end{aligned} \tag{2.18}$$

If we assume a simple case with a square grid and equal diffusion coefficients, the most conservative bound occurs when the trigonometric terms are equal to 1 and we get

$$r = \frac{\alpha \Delta t^{\gamma}}{\Delta^2} \leq \frac{B_1\left(\gamma, n, a\right)}{8} = \frac{1}{4\Xi} = B_{adap}\left(\gamma, n, a\right) \tag{2.19}$$

Unlike the full implementation in Section 2.1 where we could use the convergent nature of the series to approximate the bound B_{adap} with an analytical expression, the adaptive memory algorithm involves the parameter a which makes it difficult to take a limit approach to the expression Ξ. Fig. 2.2a shows the value of Ξ for a simulation with parameters $\gamma = 0.6$, and a varying from $a = 4$ to $a = 12$. We can see that for each value of s, the interval oscillates around some equilibrium value and if extended over infinite timesteps, would converge to said value. As would be expected, we also observe that as a increases, the value of Ξ approaches the value of the summation in Eq. 2.9 in the analogous full 2D case.

From Ξ in Eq. 2.18, we can take the sum over η for a given value of s

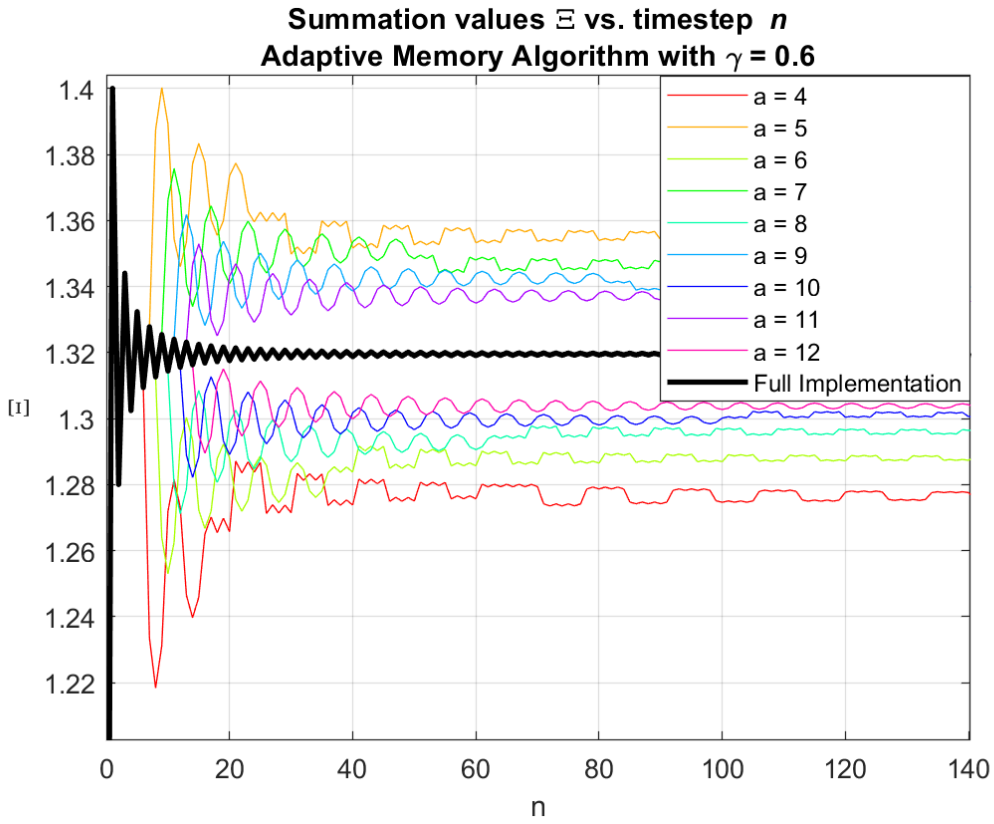
$$\sum_{\eta=1}^{\eta_{max}(s,n)} (2s-1)\,\psi\left(\gamma, M\left(s,\eta\right)\right)(-1)^{M(s,\eta)} \tag{2.20}$$

and by virtue of the nature of the memory function ψ,

$$|\psi\left(\gamma, M_{\eta+1}\right)| \leq |\psi\left(\gamma, M_{\eta}\right)|$$

which makes Eq. 2.20 an alternating series that, in the limit $\eta_{max} \to \infty$, converges. However, η never approaches infinity in any of these sums, and depending on several of these parameters, the next s interval can see the sum Ξ oscillating around a different equilibrium value (see discontinuities in Fig.2.2). In many cases it seems that as $n \to \infty$ and the amplitude of oscillations becomes smaller and smaller for each subsequent interval, we do approach a limit or at least a narrow range of values. However, all of these complications make it difficult to further simplify the expression for Ξ as an analytical expression.

On the other hand we do note that because the modulus of the memory function $|\psi|$ decreases quickly as n increases, especially for the first few terms, it is the value of the first summation term in Eq. 2.18 that largely determines the final



a)

Figure 2.2:
For a range of values for a, we plot Ξ as a function of time step n. There are several obvious discontinuities in the pattern of the oscillations, which occur at the breaks between consecutive s intervals.

value of Ξ (and in turn B_{adap}). Let $\Xi_{approx} = \sum_{m=0}^{a} \psi(\gamma, m)\,(-1)^m$. We see from Figure 2.3 that for various values of a, the error between Ξ (after it approaches a clear equilibrium, which occurs at least by $n \sim 200$ steps in most cases) and Ξ_{approx} is within 2%, and as expected, it decreases as a increases. Since we are unable to extract a simple analytical expression for Ξ, we will instead use Ξ_{approx} which, while numerically based, is accurate, simple, and quick to calculate for a reasonable range of a value.

We will now show with numerical examples how B_{adap} varies with γ, n, a, and compare to B_{full}, derived in Section 2.1.2. Figures 2.4 and 2.5 show that the values for the B functions change minimally as $n \to \infty$. We can also see that $B_{adap}(\gamma, n, a)$ approaches B_{full} as a increases (which is consistent with Fig. 2.2). As expected, the actual magnitude of the bounds depend significantly on the order of the fractional operator, γ. Still, for both the full implementation and the adaptive memory algorithms, it is easy to precompute the B functions based on γ (and a in the adaptive case) and then decide how to define parameters $\Delta t, \Delta x$, and Δy as needed to maintain stability during the simulation. Since the stability dependence on n is weak in most cases (here we are assuming that time step n is reasonably larger than a), the length of simulation has little influence on the parameters. It is also clear from these results that the adaptive memory bounds B_{adap} are pretty close to B_{full} in many cases, suggesting that the adaptive memory algorithm significantly saves on computational time and power, without adverse effect on the stability regime of the parameters.

2.3 Simulation Results

Here we will show that our stability analyses agree well with numerical simulations. For all plots in the results section, we are using the adaptive memory algorithm defined in Eq. 2.13 and have set $\alpha = 50 \frac{units^2}{s^\gamma}$, $\Delta x = \Delta y = 10$ *units*, and base interval $a = 8$ (which insures a reasonable level of accuracy with maximum error of the adaptive memory algorithm remaining under 1% for the duration of the simulation). Our initial condition is a narrow two-dimensional Gaussian:

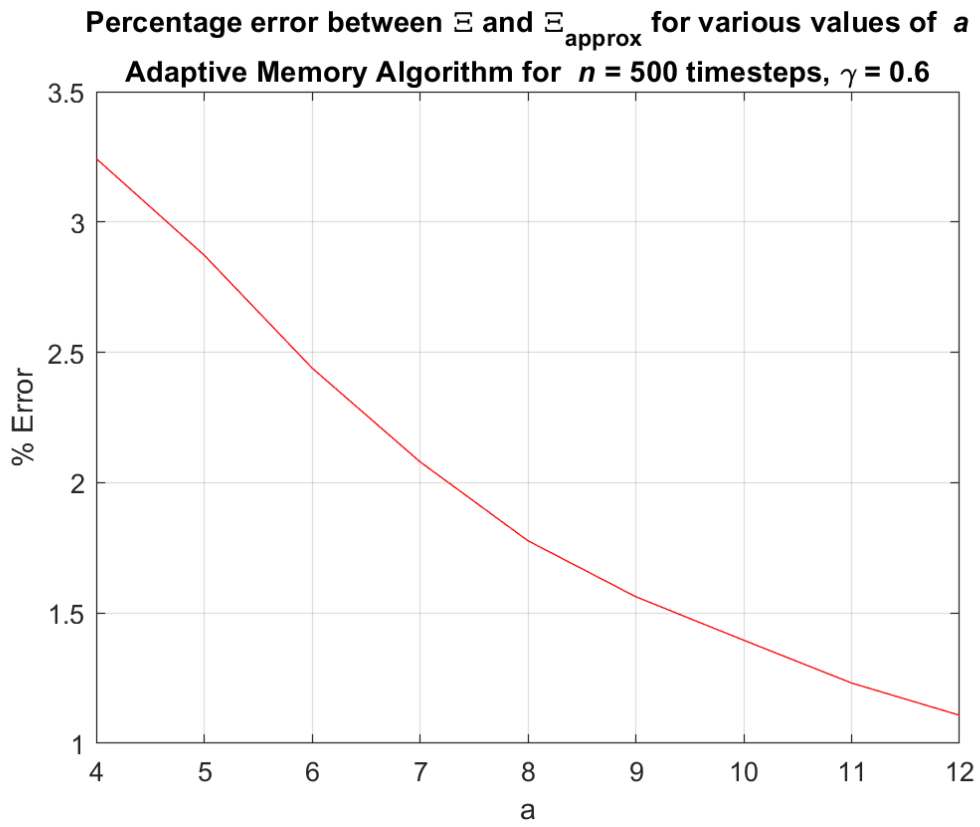


Figure 2.3: Error values between Ξ and Ξ_{approx} for various values of a. These values correspond to $n = 500$ timesteps, and $\gamma = 0.6$.

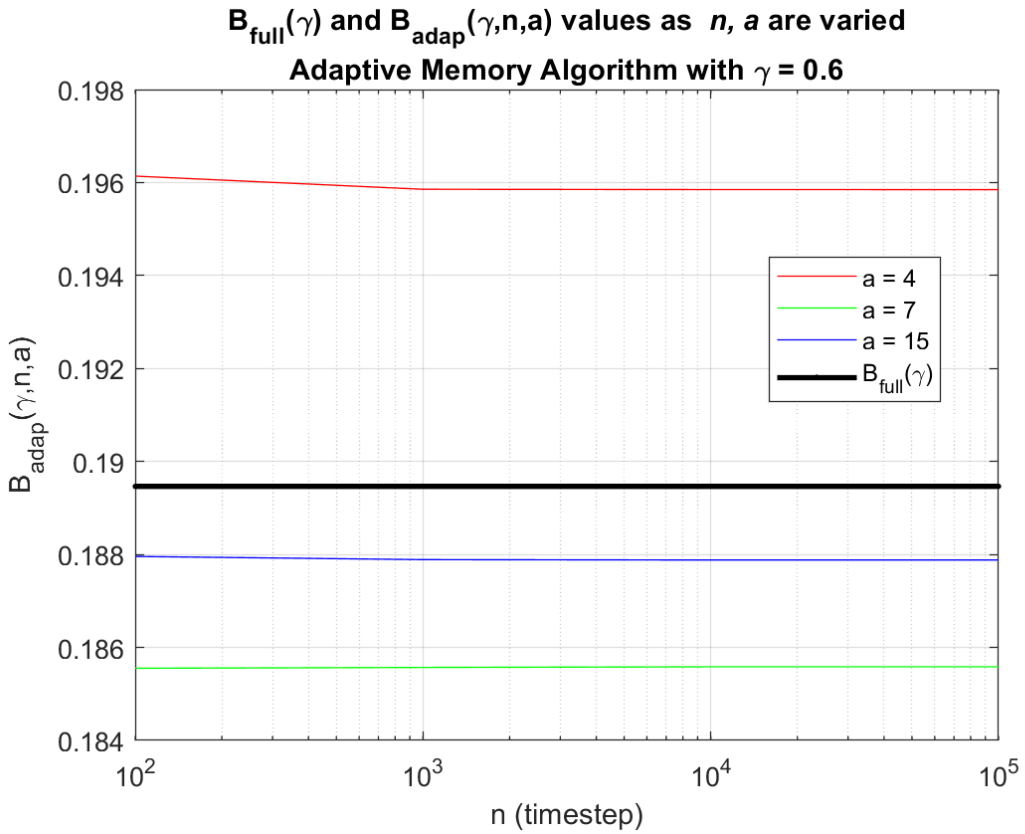


Figure 2.4: Adaptive Memory Algorithm with $\gamma = 0.6$: $B_{full}(\gamma)$ compared to $B_{adap}(\gamma, n, a)$ values as parameters a, n are varied. The dependence of B_{adap} on n is very weak. As expected, as a increases, B_{adap} approaches B_{full}.

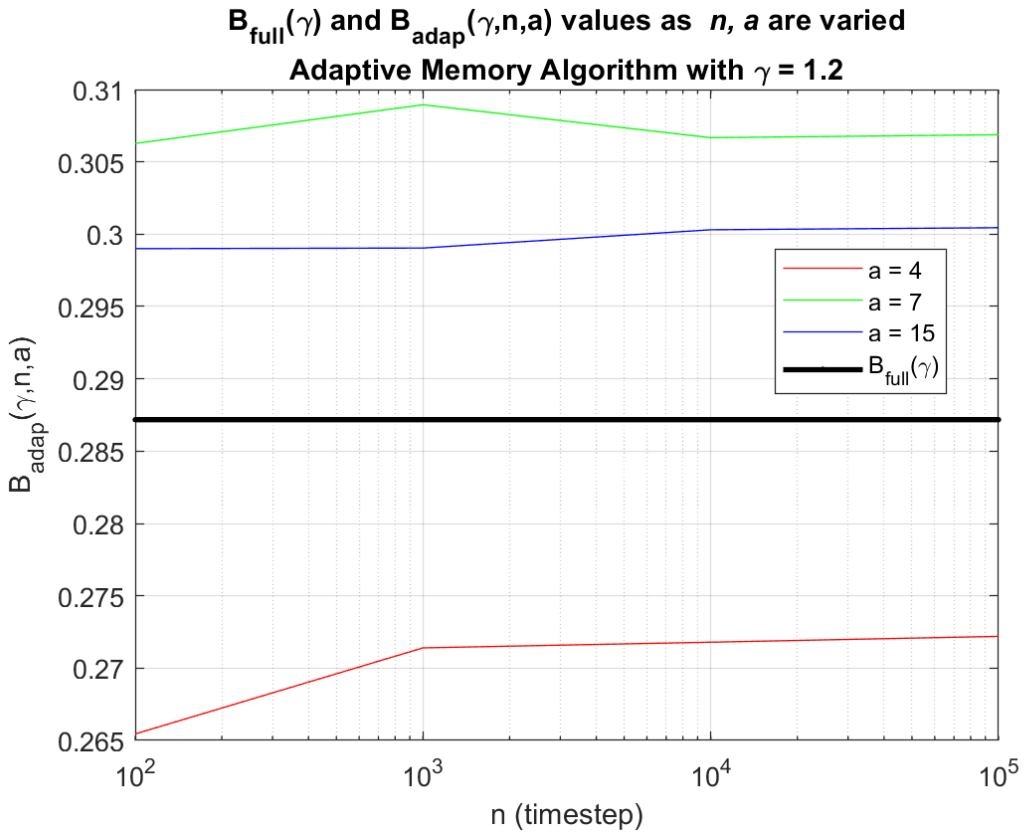


Figure 2.5: Adaptive Memory Algorithm with $\gamma = 1.2$: $B_{full}(\gamma)$ compared to $B_{adap}(\gamma, n, a)$ values as parameters a, n are varied. B_{adap} a bit more with n compared to the $\gamma = 0.6$ case (Fig. 2.4), but the dependence is still generally weak. As expected, as a increases, B_{adap} approaches B_{full}.

$u(x,y,0) = e^{-x^2/2\sigma_1^2} e^{-y^2/2\sigma_2^2}$, $\sigma_1 = 5$ *units*, $\sigma_2 = 5$ *units*.

We begin by setting $\gamma = 0.6$, which puts our simulations in the subdiffusion regime where the true solution to the fractional diffusion equation is composed of only decaying modes. Recall that $r = \frac{\alpha \Delta t^\gamma}{\Delta^2}$, and with the parameters we have already set, we find that according to Eq. 2.19, for stable conditions, $r \leq B_{adap} = 0.1929$ using Ξ, and $r \leq B_{adap} = 0.1905$ if we use Ξ_{approx}. Our only degree of freedom is time step Δt such that $\Delta t \leq \left(\frac{r\Delta^2}{\alpha}\right)^{(1/\gamma)} = 0.20024s$ using the more conservative B_{adap} estimate.

In Fig. 2.6a,b, we have chosen $\Delta t = 0.1s$ which sets $r = 0.12559$, fulfills the stability inequality criteria and puts our parameters in the stable region of parameter space and relatively far from the boundary. Both the intensity maps and surface plots confirm that the numerical solution is clearly stable at early and late time points in the simulation, as there is no oscillatory behavior.

In Fig. 2.6c,d, we have chosen $\Delta t = 0.2s$ ($r = 0.19037$), which puts our parameters in the stable region but close to the boundary. The plots show that there is an oscillatory component that is evident early in the simulation. However, at a later time in the same simulation, the oscillatory component has been suppressed and the solution has decayed in time and space as expected of subdiffusive behavior. It is reasonable that the oscillatory component was present in the beginning of the simulation since r is close to the boundary between the stable and unstable regimes. However r is still strictly in the stable regime and this is verified by the fact that the oscillations do decay with time. Recalling the discussion in Section 2.1.2 relating to high frequency Fourier modes, we can see clearly from these plots that the oscillations causing instability are indeed on the order of the grid discretization.

In Fig. 2.7a,b, we have chosen $\Delta t = 0.21s$, which sets $r = 0.19602$ and puts our parameters in the unstable region, but close to the boundary between stable and unstable regions. As expected of unstable simulations, the numerical solution includes oscillatory components that appear early in the simulation and persist as the simulation continues (they would remain even as $n \to \infty$). But because r is close to the boundary with the stable regime, the oscillations do not drastically overwhelm the decaying modes, suggesting that the solution is only mildly unstable.

In Fig. 2.7c, we have chosen $\Delta t = 0.3s$, which sets $r = 0.2428$, putting our parameters clearly in the unstable region and further from the boundary. The plots depicting a later time in the simulation confirm this by showing that the oscillatory behavior of the solution grows unbounded with time and completely overwhelms the decaying modes of the true solution, as observed by scale of the z axis, which is orders of magnitude larger than that of the true solution.

Next, we will consider the superdiffusion cases with $\gamma = 1.2$. According to Eq. 2.19, for stable numerical solutions, $r \leq B_{adap} = 0.272$ using Ξ, and $r \leq B_{adap} = 0.2809$ if we use Ξ_{approx}. In addition, it is less straightforward to characterize the superdiffusion regime because in addition to decaying modes characteristic of diffusion, the true solution also has oscillatory components (which is logical because as γ increases towards 2, we approach the classical wave equation). However, we can still observe the difference in behavior between unstable and stable numerical solutions.

In Fig. 2.8a, we have chosen $\Delta t = 0.4s$, which sets $r = 0.16651$, putting our parameters in the stable region. The plots show a numerical solution with low spatial frequency oscillatory components (oscillating on a much larger scale than the scale of the grid elements), and the presence of high frequency components characteristic of noise, is absent. The numerical solution remains bounded. In Fig. 2.8b, $\Delta t = .55s$, which sets $r = 0.24401$, putting our parameters still in the stable region but close to the boundary between stable and unstable regions. The presence of the high frequency noise components is apparent, especially compared to Fig. 2.8a at the same simulation time, but even so late into the simulation, they do not overwhelm the overall solution and are clearly bounded, which verifies stability. Finally, in Fig. 2.8c,d, we have chosen $\Delta t = 0.7s$, setting $r = 0.3259$ and putting our parameters clearly in the unstable region. The plots show high frequency oscillatory behavior that appears early in the simulation when decaying modes still dominate the solution. At a later timestamp the oscillatory components have grown drastically and in an unbounded manner, which, as also noted in Fig. 2.7, is a good indicator of numerical instability.

All our results verify the conclusions made during our stability analysis in Sections 2.1.2 and 2.2. It's also clear from Figs. 2.6 and 2.7 that the boundary between stable and unstable regimes is quite a sharp one, as changing our time step from $\Delta t = 0.2s$ to $\Delta t = 0.21s$ was enough to change our simulations from being stable to unstable.

3 Complexity Analysis

In the following sections we analyze the complexity of the full two-dimensional implementation of the fractional diffusion equation, the adaptive memory algorithm, and the linked list alternative version defined in [7].

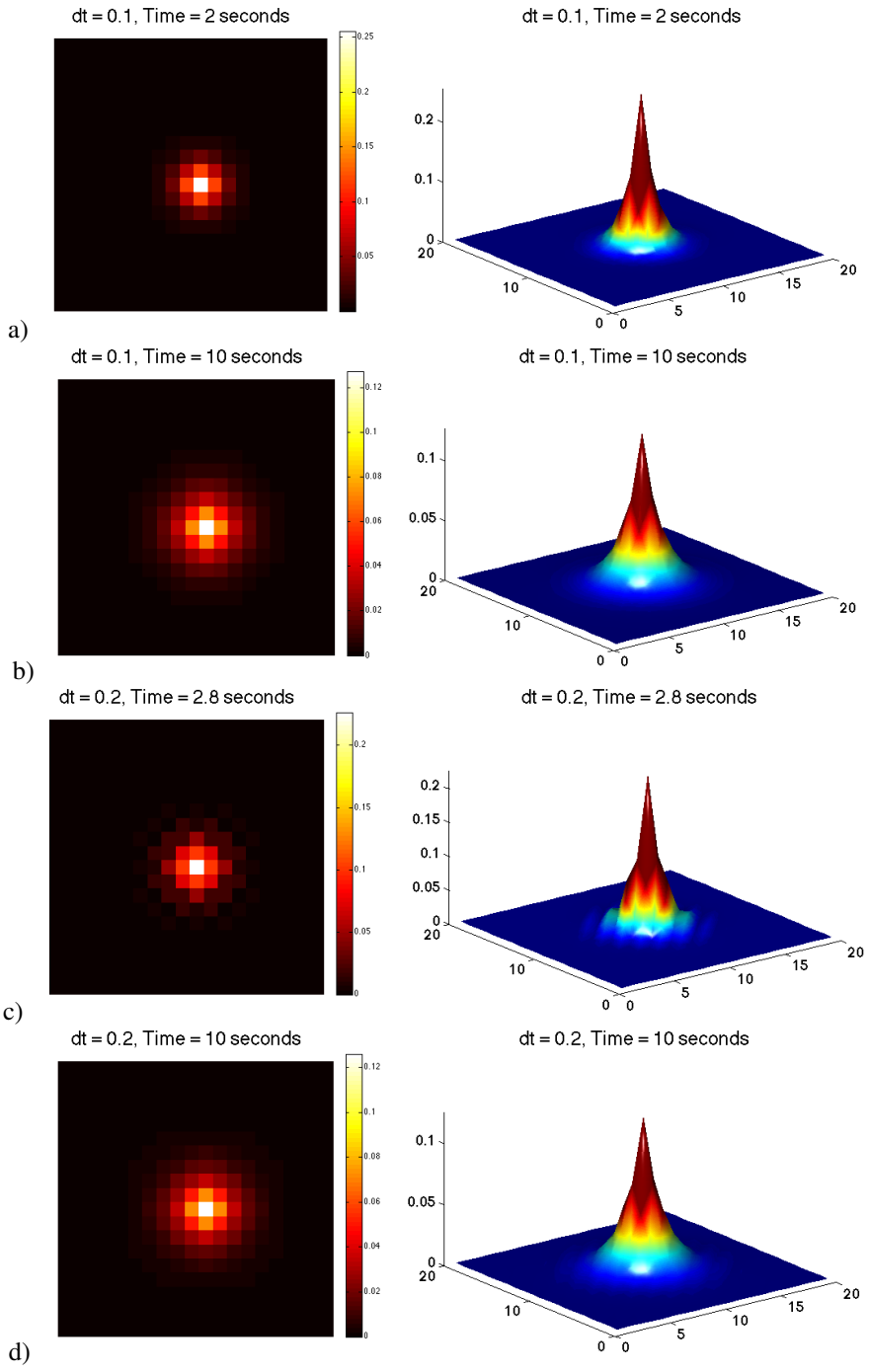


Figure 2.6: **Stable Simulations in the Subdiffusion Region**. $\alpha = 50 \frac{units^2}{s^\gamma}$, $\Delta x = \Delta y = 10$ *units*, $\gamma = 0.6$, left column: 2D intensity plots, right column: 3D surface plots. **a,b)** $\Delta t = 0.1$ *s*, r is in the stable region and far from the bound, and it is clear that there is no oscillatory component at any point in the simulation. **c,d)** $\Delta t = 0.2s$, r in the stable regime, but very close to the boundary. It is clear in the snapshots, that there is a tiny oscillatory component that appears early in the simulation (away from the center of the grid), but decays as the simulation progresses, and so the solution remains bounded.

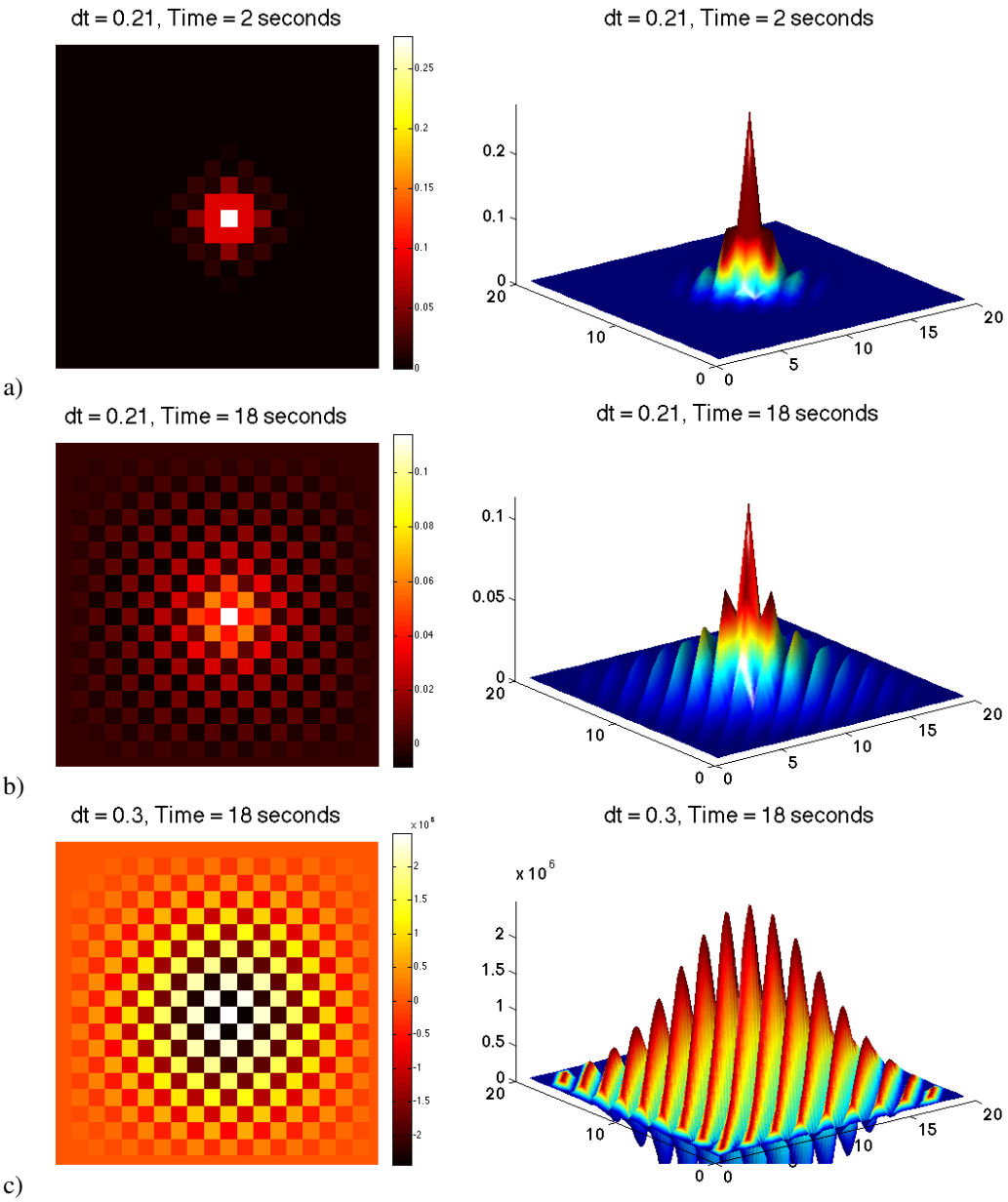


Figure 2.7: **Unstable Simulations in the Subdiffusion Region**. $\alpha = 50 \frac{units^2}{s^\gamma}$, $\Delta x = \Delta y = 10\ units$, $\gamma = 0.6$, left column: 2D intensity plots, right column: 3D surface plots. **a,b)** $\Delta t = 0.21s$, r is in the unstable region and close to the boundary. The plots reflect an oscillatory component that appears early in the simulation and is sustained through the duration of the simulation. Even though the oscillations do not increase, and do not overwhelm the decaying modes of the true solution, they do not disappear and would remain even as $n \to \infty$, which is an indication of instability. **c)** $\Delta t = 0.3s$, r is in the unstable region, and far from the bounds between the unstable and stable regions. The plots reflects oscillatory components that remain at a later time in the simulation and have increased in an unbounded manner. If we note the axis scale of both plots, we also observe that those components have completely overwhelmed the decaying modes of the true solution. These are a hallmarks of strong instability in numerical simulations.

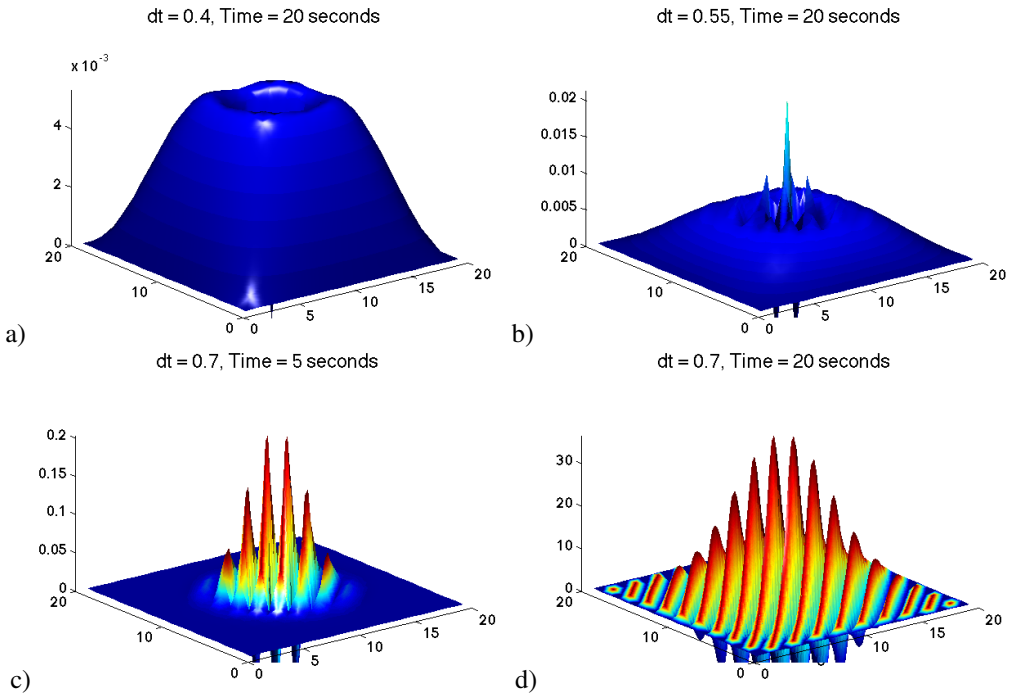


Figure 2.8:

Stable and Unstable Superdiffusion Simulations. $\alpha = 50 \frac{units^2}{s^\gamma}$, $\Delta x = \Delta y = 10$ *units*, $\gamma = 1.2$. **a)** $\Delta t = 0.4s$, r is in the stable regime. Because we are in the superdiffusion regime, the true solution does have oscillatory components alongside decaying modes. However, the numerical solution is obviously bounded, even later in the simulation, and there is no sign of high frequency oscillations (visible on the plots **c** and **d**), so the plots seem to verify that the simulation is indeed stable. **b)** $\Delta t = 0.55s$, r is in the stable regime. We can clearly see high frequency components, especially contrasted the same plot at the same time in the simulation, in part **a**. However, as observed by the scale bar, the high frequency components don't overwhelm the decaying modes of the solution and cause the solution to grow unbounded, so this verifies stability. **c,d)** $\Delta t = 0.7s$, r is in the unstable regime. It is clear from the plots that the high frequency oscillatory components is present early in the simulation, but the decaying modes still dominate the overall solution. However, it's clear that later in the simulation the high frequency oscillations continue to grow unbounded in time, which verifies the unstable nature of the numerical solution.

Algorithm 1 Full 2D Implementation

1: **for** $n = 0 : N$ **do** ▷ N is the total # of timesteps the algorithm will run
2: **for** $j = 0 : N_x - 1$ **do** ▷ iterate over all grid points in x direction
3: **for** $l = 0 : N_y - 1$ **do** ▷ iterate over all grid points in y direction
4: **for** $m = 0 : 1 : n$ **do**
5: Calculate $\psi(\gamma, m)\left(\frac{\alpha}{\Delta x^2}\delta x_{j,l}^{n-m} + \frac{\beta}{\Delta y^2}\delta y_{j,l}^{n-m}\right)$
6: **end for**
7: Calculate $u_{j,l}^{n+1}$
8: **end for**
9: **end for**
10: **end for**

3.1 Full Implementation

We begin with the pseudocode for the full implementation defined in Eq. 2.1. In our pseudocode we generally omit instructions not essential to showing the logical structure of the algorithm and note that the most important aspect of time complexity analysis are data structures involving loops. Lines of code that are run a constant number of times and are independent of variables like timestep, can be ignored in the overall Big-O functional form.

From the pseudocode in Algorithm 1 we can see that for each n, the total approximate number of instructions (ignoring things like constant number of instructions) is $N_x N_y (n+1)$ where n varies from $0 : N$. Therefore:

$$\begin{aligned} total\ instructions &= N_x N_y (1 + ... + N + 1) \\ &= N_x N_y \left(\frac{(N+1)(N+2)}{2}\right) \\ &\Rightarrow O\left(N_x N_y N^2\right) \end{aligned}$$

This gives us the growth in execution time as a function of the grid dimensions N_x and N_y, as well as the number of timesteps N. While execution time grows linearly with the size of each spatial dimension of the grid we are using, it grows as the square of the number of timesteps in the simulation, which is not the most efficient numerical implementation.

Here we note that in the one-dimensional case, we can easily vectorize the loop iterating over grid points, increasing efficiency with regards to grid size. However, with respect to the time dimension, complexity still grows as $O\left(N^2\right)$.

3.2 Adaptive Memory Algorithm

In Algorithm 2 we provide the pseudocode for the implementation of the adaptive memory algorithm described in Eq. 2.13. As before, we omit the instructions detailing the calculations themselves because counting these instructions involve multiplying constants against the number of times the loops are run, and do not affect the form of the Big-O expressions in terms of the input variable of interest (N_x, N_y, N). As with the full implementation, we briefly note that in the one-dimensional analog, we can vectorize the loop iterating over spatial gridpoints.

Analysis of the total number of instructions in Part 1 of Algorithm 2 proceeds in exactly the same way as in Section 3.1. The total instruction count is approximately

$$\begin{aligned} N_x N_y (1 + 2 + ... + a + 1) &= N_x N_y \frac{(a+1)(a+2)}{2} \\ &\Rightarrow O\left(N_x N_y a^2\right) \end{aligned}$$

Analysis of Part 2 of Algorithm 2 is more complicated because of the additional parameters s_{max}, η_{max}, and M_{max}. For loop {L1} (lines 21-23), η_{max} depends on the current s interval, but the maximum value is always $\frac{a^s - a^{s-1}}{2s-1}$. For loop {L2} (lines 24-26), the loop serves to sample timesteps when the current interval $[a^{s-1} : a^s]$ (or $[a^{s-1} : n]$ for the latest interval corresponding to s_{max}), is not evenly divisible by the weight for that interval, $(2s-1)$, and therefore there are a few timesteps that aren't taken into account with loop {L1}. The maximum number of times loop {L2} is run, is always

Algorithm 2 Adaptive Memory Algorithm

▷ **Part 1**

for $n = 0 : a$ **do**
  for $j = 0 : N_x - 1$ **do** ▷ iterate over all grid points in x direction
    for $l = 0 : N_y - 1$ **do** ▷ iterate over all grid points in y direction
      for $m = 0 : 1 : n$ **do**
        Calculate $\psi(\gamma, m)\left(\frac{\alpha}{\Delta x^2}\delta x_{j,l}^{n-m} + \frac{\beta}{\Delta y^2}\delta y_{j,l}^{n-m}\right)$
      end for
      At this location, calculate $u_{j,l}^{n+1}$
    end for
  end for
end for

▷ **Part 2**

for $n = a+1 : N$ **do** ▷ N is the total # of timesteps the algorithm will run
  Determine s_{max} based on n
  for $j = 0 : N_x - 1$ **do** ▷ iterate over all grid points in x direction
    for $l = 0 : N_y - 1$ **do** ▷ iterate over all grid points in y direction
      for $m = 0 : 1 : a$ **do** ▷ Calculate Sum 1 (base interval)
        Calculate R1: $\psi(\gamma, m)\left(\frac{\alpha}{\Delta x^2}\delta x_{j,l}^{n-m} + \frac{\beta}{\Delta y^2}\delta y_{j,l}^{n-m}\right)$ ▷ R1 from Eq. 2.13
      end for
      for $s = 2 : s_{max}$ **do**
        for $\eta = 1 : \eta_{max}$ **do**
          Calculate R2 in Eq. 2.13 } $\{L1\}$
        end for
        for $p = M_{max} + s : min\,(a^s, n)$ **do**
          Calculate R3 in Eq. 2.13 } $\{L2\}$
        end for
      end for
      Use R1-R3 to calculate $u_{j,l}^{n+1}$
    end for
  end for
end for

$\{L3\}$

$< 2s-1$. Therefore in the large *for* loop {L3} in Algorithm 2 encompassing lines 13-31, for a given value of n, the number of instructions run is, at most:

$$total\ instructions \leq N_x N_y \left\{ a+1+\sum_{s=2}^{s_{max}} \left(\underbrace{\frac{a^s-a^{s-1}}{2s-1}}_{\{1a\}} + \underbrace{2s-1}_{\{1b\}} \right) \right\} \tag{3.1}$$

To simplify this expression further we need to make some assumptions about worst case scenarios. We can simplify the first term {1a} of the summation in Eq. 3.1 using telescoping series:

$$\begin{aligned}
\sum_{s=2}^{s_{max}} \frac{a^s-a^{s-1}}{2s-1} &= \frac{a^2-a}{3} + \frac{a^3-a^2}{5} + \ldots + \frac{a^{s_{max}}-a^{s_{max}-1}}{2s_{max}-1} \\
&< \frac{a^2-a}{3} + \frac{a^3-a^2}{3} + \ldots + \frac{a^{s_{max}}-a^{s_{max}-1}}{3} \\
&= \frac{1}{3}\left\{ \left(a^2-a\right) + \left(a^3-a^2\right) + \ldots + \left(a^{s_{max}-1}-a^{s_{max}-2}\right) + \left(a^{s_{max}}-a^{s_{max}-1}\right) \right\} \\
&= \frac{1}{3}\left(a^{s_{max}}-a\right)
\end{aligned}$$

The second term {1b} of the summation can be reduced to

$$\begin{aligned}
\sum_{s=2}^{s_{max}} 2s-1 &= 2\sum_{s=2}^{s_{max}} s - \sum_{s=2}^{s_{max}} 1 \\
&= 2\left(\frac{s_{max}\left(s_{max}+1\right)}{2} - 1 \right) - \left(s_{max}-1\right) \\
&= s_{max}^2 - 1
\end{aligned}$$

The expression in Eq. 3.1 can now be reduced to

$$total\ instructions \leq N_x N_y \left(a + \frac{1}{3}a^{s_{max}} - \frac{1}{3}a + s_{max}^2 - 1 \right) \tag{3.2}$$

s_{max} for a given timestep n is determined by $a^{s_{max}-1}+1 \leq n \leq a^{s_{max}}$, or $\frac{ln(n)}{ln(a)} \leq s_{max} \leq \frac{ln(n-1)}{ln(a)} + 1$. Since we are concerned with worst case scenarios, we use the maximum value of this interval, $s_{max} = \frac{ln(n-1)}{ln(a)} + 1$ and substitute into Eq. 3.2 . Now we must consider the *for* loop {L3} in Algorithm 2 and how the total number of instructions relates to the total number of timesteps N:

$$total\ instructions \leq N_x N_y \sum_{n=a+1}^{N} \left(\underbrace{\frac{2}{3}a}_{\{2a\}} + \underbrace{\frac{1}{3}a^{ln(n-1)/ln(a)}}_{\{2b\}} a + \underbrace{\left(\frac{ln(n-1)}{ln(a)}\right)^2}_{\{2c\}} + 2\underbrace{\frac{ln(n-1)}{ln(a)}}_{\{2d\}} \right) \tag{3.3}$$

We now simplify the four components of the summation in Eq. 3.3.

For {2a}, $\sum_{n=a+1}^{N} \frac{2}{3}a = \left(N-a\right)\frac{2}{3}a$. Since a is a constant parameter chosen by the user prior to the simulation, the dominant term in this expression is clearly N.

For {2b}:

$$\begin{aligned}\frac{a}{3}\sum_{n=a+1}^{N} a^{\frac{ln(n-1)}{ln(a)}} &= \frac{a}{3}\sum_{n=a+1}^{N} a^{\frac{log_a(n-1)}{log_a e}\frac{1}{lna}} \\ &= \frac{a}{3}\sum_{n=a+1}^{N}\left[a^{log_a(n-1)}\right]^{\frac{1}{(log_a e)(log_e a)}} \\ &= \frac{a}{3}\sum_{n=a+1}^{N} n-1 \\ &= \frac{a}{3}\left(\frac{N^2+N}{2}-(1+2+...+a)-(N-a)\right)\end{aligned}$$

The dominant term here is clearly N^2.

For {2c} in equation Eq. 3.3:

$$\begin{aligned}\frac{1}{(ln(a))^2}\sum_{n=a+1}^{N}(ln(n-1))^2 &= \frac{1}{(ln(a))^2}\left[(ln(a))^2+(ln(a+1)^2+...+(ln(N-1))^2\right] \\ &\leq \frac{1}{(ln(a))^2}\left[(ln(N-1))^2+(ln(N-1))^2+...+(ln(N-1))^2\right] \\ &= \frac{1}{(ln(a))^2}\left[(N-a)(ln(N-1))^2\right]\end{aligned}$$

This expression in Big-O notation is $O\left(N\left(ln\left(N\right)\right)^2\right)$. Alternatively, we can simplify {2c} using integration by parts, to get the same result.

For {2d}:

$$\begin{aligned}\frac{2}{ln(a)}\sum_{n=a+1}^{N} ln(n-1) &= \frac{2}{ln(a)}ln\left[(a)(a+1)...(N-1)\right] \\ &= \frac{2}{ln(a)}ln\left(\frac{(N-1)!}{(a-1)!}\right) \\ &= \frac{2}{ln(a)}\left(ln\left((N-1)!\right)-ln\left((a-1)!\right)\right)\end{aligned}$$

Using Stirling's approximation, our final expression is

$$\frac{2}{ln(a)}\left(Nln(N-1)-N+1+O\left(ln\left(N-1\right)\right)-ln((a-1)!)\right) \tag{3.4}$$

which is dominated by the $Nln\left(N-1\right)$term.

Combining the simplified expressions for {2a}-{2d} in Eq. 3.3, it is clear that the N^2 term dominates the behavior of the overall expression as N becomes large, and so we conclude that the worst case behavior of the adaptive step algorithm, is still bounded by $O\left(N_x N_y N^2\right)$. While the adaptive time step algorithm improves raw execution time in relation to the full memory implementation, the complexity analysis shows that when it comes to scaling with large N, the algorithm is as inefficient as the full implementation.

3.3 Linked List Adaptive Timestep Algorithm

In [7] we describe an alternative version of the adaptive memory algorithm which is based on a power law that enables us to eliminate past history timepoints that will never again need to be referenced, thus saving us execution time and

Algorithm 3 Linked List Implementation

for $n = 0 : N$ **do** ▷ N is the total # of timesteps the algorithm will run
  for $j = 0 : N_x - 1$ **do** ▷ iterate over all grid points in x direction
    for $l = 0 : N_y - 1$ **do** ▷ iterate over all grid points in y direction
      for $node = 1$:length(linked list) **do**
        From each node calculate $\psi(\gamma, n-i) w^i \left(\frac{\alpha}{\Delta x^2} \delta x^i_{j,l} + \frac{\beta}{\Delta y^2} \delta y^i_{j,l} \right)$ } {L1}
      end for
      Calculate $u^{n+1}_{j,l}$
    end for
  end for
  Create a new node in the list to store $u^{n+1}_{j,l}$
  If-else statements to determine whether we need to condense the linked list according to Algorithm 4.1 in [7]. If so:
  while it continues to be necessary to condense the linked list **do**
    Constant number of instructions to delete nodes and update weights } {L2}
  end while
end for

Table 1: Linked list data for timestep $n = 25$

Timestep	0	4	8	12	14	16	18	20	22	23	24	25
Weight	$2^2 = 4$	4	4	$2^1 = 2$	2	2	2	2	$2^0 = 1$	1	1	1

memory. Here we summarize the algorithm and refer to [7, 2] for additional background technical details and derivation.

$$\frac{u^{n+1}_{j,l} - u^n_{j,l}}{\Delta t^\gamma} = \sum_{\left\{u^i_{j,l} \in U^n\right\}} \psi(\gamma, n-i) w^i \left(\frac{\alpha}{\Delta x^2} \delta x^i_{j,l} + \frac{\beta}{\Delta y^2} \delta y^i_{j,l} \right) \tag{3.5}$$

$$w^{n+1} = 1$$
$$W^{n+1} := \left\{w^i \in W^n\right\} + \left\{w^{n+1}\right\}$$
$$U^{n+1} := \left\{u^i \in U^n\right\} + \left\{u^{n+1}\right\}$$

The data associated with each timestep n is stored in the elements constituting a doubly linked list, and i represents the timestep in each node/element in the linked list. Additionally, when there are more than η points in the set W^i of any given weight, the elements of the set are condensed according to Algorithm 4.1 in [7]. We present the pseudocode for the linked list implementation in Algorithm 3.

To determine how many times loop {L1} runs in relation to timestep n, we can consider the following. For a given timestep n, we are interested in a summation of terms involving weights (powers of two), each set of which never exceeds η nodes. For example, for timestep $n = 25$ and $\eta = 5$, Table 1 shows the timestep and weight for all nodes currently in the linked list at this point in the simulation.

Since the dynamically changing weight is directly related to how often the history of the simulation is sampled, we can construct the following relationship between time step and weight based on the data in Table 1:

$$2^0 \cdot 5 + 2^1 \cdot 5 \leq n+1 = 26 \leq 2^0 \cdot 5 + 2^1 \cdot 5 + 2^2 \cdot 5$$

The inequality arrives from the fact that a particular weight category has η or fewer nodes.

For a general timestep n we can write the inequalities in terms of geometric series:

$$\sum_{l=0}^{x-1} 2^l \leq \frac{n+1}{\eta} \leq \sum_{l=0}^{x} 2^l \tag{3.6}$$

where $x+1$ is the number of weight categories represented in the linked list at timestep n. In the example in Table 1, the number of weight categories is 3, with weights 2^0, 2^1, and 2^2. Substituting the geometric series with their total

sums and taking log_2 of both sides, we can rewrite equation Eq. 3.6 as

$$x \leq log_2\left(\frac{n+1}{\eta}+1\right) \leq x+1 \Rightarrow$$
$$log_2\left(\frac{n+1}{\eta}+1\right) \leq x+1 \leq log_2\left(\frac{n+1}{\eta}+1\right)+1 \tag{3.7}$$

where $x+1$ is the number of weight categories.

We now return to Algorithm 3 and refer to loop {L1}. The maximum possible number of times this loop runs for any given timestep n, is the length of the linked list, which is η times the maximum number of weight categories at that time. Using Eq. 3.7, we get

$$\eta\left(log_2\left(\frac{n+1}{\eta}+1\right)+1\right)$$

Therefore the total number of instructions in lines 2-9 of Algorithm 3 is $N_xN_y\eta\left(log_2\left(\frac{n+1}{\eta}+1\right)+1\right)$. We now look at loop {L2} (lines 11-13). This *while* loop runs whenever the number of nodes belonging to a particular weight category, exceeds the user-set parameter η, and we need to condense the linked list. The maximum number of times this may run within a single timestep, is the number of weight categories in the linked list. For example, if the number of nodes with weight 2^0 exceeds η, we delete the second "least" node in the set of nodes with this weight, and double the weight of the "least" element (node) in this same set, to 2^1. (See the derivation of Algorithm 4.1 in [7] for details about the ordering of these sets). For some particular timestep this may mean that the number of nodes with weight 2^1 now exceeds η and we continue to condense in the same fashion until we have addressed all weight categories currently represented in the linked list. The maximum number of weight categories is given by Eq. 3.7, and this is the maximum number of times loop {L2} is run for a given time step n. Now we can combine all loops in Algorithm 3 and write that the maximum number of instructions (ignoring constants) is given by

$$\begin{aligned} total\ instructions &\leq \sum_{n=0}^{N}\left[N_xN_y\eta\left(log_2\left(\frac{n+1}{\eta}+1\right)+1\right)+log_2\left(\frac{n+1}{\eta}+1\right)+1\right] \\ &= (N_xN_y\eta+1)\left[\underbrace{\sum_{n=0}^{N} log_2\left(\frac{n+1}{\eta}+1\right)}_{\{1\}}+N+1\right] \end{aligned} \tag{3.8}$$

We can simplify the summation term {1}:

$$\sum_{n=0}^{N} log_2\left(\frac{n+1+\eta}{\eta}\right) = \sum_{n=0}^{N}\left(log_2(n+1+\eta)-log_2(\eta)\right)$$

With a change of variable $d = n+1+\eta$ we can rewrite the summation and once again utilize Sterling's approximation to arrive at:

$$\begin{aligned} \sum_{d=1+\eta}^{N+1+\eta=N'} log_2(d) - Nlog_2(\eta) &\leq log_2(\eta+1)+log_2(\eta+2)+...+log_2\left(N'\right)-(N+1)log_2(\eta) \\ &= log_2\left(\frac{N'!}{\eta!}\right)-(N+1)log_2(\eta) \\ &= N'log_2\left(N'\right)-N'log_2e+O\left(log_2\left(N'\right)\right)-(N+1)log_2(\eta)-log_2\eta! \end{aligned}$$

We can now write Eq. 3.8 as

$$total\ instructions \leq (N_xN_y\eta+1)\left[N'log_2N'-N'log_2e-(N+1)log_2\eta+N+O\left(log_2N'\right)-log_2\eta!+1\right]$$

It is clear that the dominant term here is $N'\left(log_2\left(N'\right)\right)$and since N' is simply equal to N offset by a constant, we can conclude the overall complexity is

$$O\left(N_xN_yNlog_2(N)\right)$$

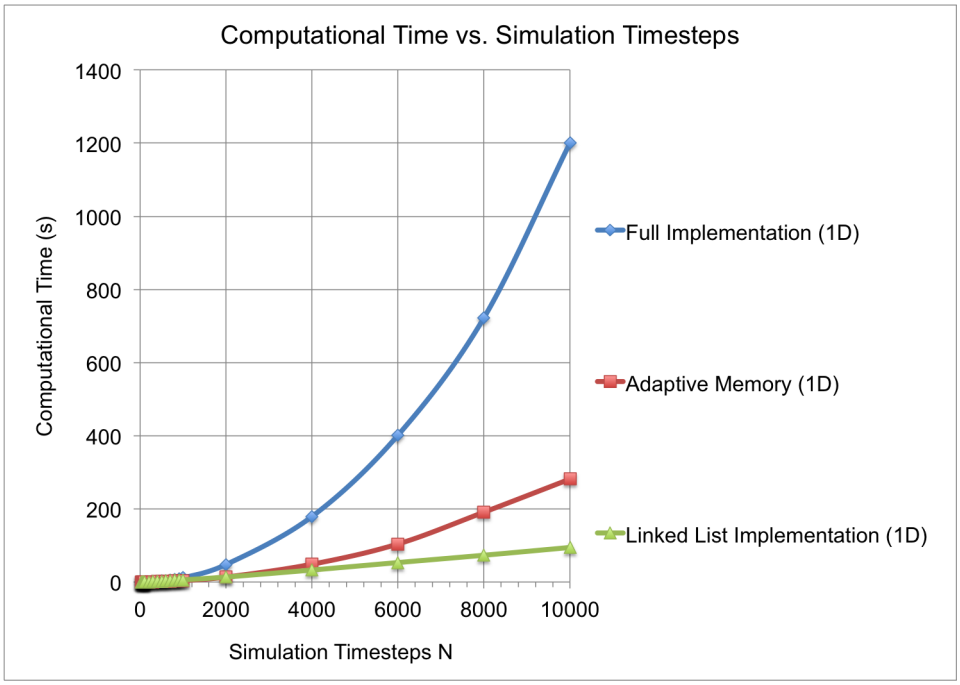


Figure 3.1: Simulated data verifying theoretical complexity results. For all simulations we set $\gamma = 0.8$, $D = 1\frac{units^2}{s^\gamma}$, $dt = 0.1s$, $dx = 0.6875\ units$. We used adaptive step adaptive step parameter a and η values of 20.

Compared to the full and adaptive step algorithms analyzed in this section, the linked list implementation is clearly the most efficient in terms of how execution time scales as number of timesteps N, grows large. An added benefit (as noted in [7], Section 4) is the lowered memory requirements from $O(N)$ (data stored for all N time steps) to $O(log_2 N)$, since we no longer require keeping all time points in the history of the simulation.

As before, we also note that in the one-dimensional analogous case of the linked list pseudocode, the loop iterating over spatial grid points can be vectorized.

3.4 Simulation Results

We verified our theoretical complexity results with simulated data. In Fig. 3.1 we show computation run times for simulations of various number of steps N and demonstrate the complexity of the three numerical methods explored in this section (using one-dimensional versions for simplicity). For all algorithms we set $\gamma = 0.8$, $D = 1\frac{units^2}{s^\gamma}$, $dt = 0.1s$, $dx = 0.6875\ units$. We used adaptive step parameter a and η values of 20.

Trendlines were fit to the data using Matlab's curve fitting toolbox. For the data corresponding to the full and adaptive memory implementations, the best fit lines were polynomials of second order (modeled with three coefficients). For the linked list implementation, the best fit trendline with minimum number of coefficients was a $Nlog_2 N$ function.

- full implementation simulation time = $p_1 N^2 + p_2 N + p_3$ with $p_1 = 1.106e-5$, $p_2 = 5.5e-4$, $p_3 = -0.0423$ and goodness of fit measure $R^2 = 0.9999$
- adaptive memory simulation time = $p_1 N^2 + p_2 N + p_3$ with $p_1 = 2.671e-5$, $p_2 = 1.441e-3$, $p_3 = -0.05873$ and goodness of fit measure $R^2 = 0.9999$
- linked list simulation time = $p_1 Nlog_2(N)$ with $p_1 = 7.047e-4$ and goodness of fit measure $R^2 = 0.9992$. This is the simplest trendline that was found to be a good fit. There were other functions that fit the data for the linked list implementation but required additional coefficients.

The data fitting and simulated data in Fig. 3.1 thus supports the theoretical analysis done in this section.

It is worth mentioning that there is additional overhead with the linked list implementation, involving the actual handling of the linked list (e.g. deleting and inserting new nodes, etc.). For this reason, for shorter simulations, the

linked list algorithm may actually take longer to execute than the full or adaptive memory implementations. It is only when simulation length exceeds a certain point (and this threshold is determined by the details of the simulation, including parameter values) that the $NlogN$ time scaling complexity become more noticeable and the advantages of the linked list approach become more evident.

4 Error

In the last section we found that the algorithmic complexity for the linked list implementation is generally more computationally efficient than the full and adaptive memory implementations, with the exception of shorter simulation lengths where the overhead of dealing with linked list operations, may overwhelm the algorithmic scaling in time. However, with numerical algorithms, considerable advantages in one area usually come with some drawback or cost, in another. In this case, the accuracy of the linked list implementation is the drawback. In Fig. 4.1 we compare the error of the adaptive memory and linked list approaches, with the full implementation. We include both maximum error at each timepoint, and average error over the whole solution, at each timepoint. The reason we compare with the full implementation is because there is no closed form solution for the 2D fractional diffusion equation except for very specific circumstances, and these two algorithms were derived in part based on the basic finite difference scheme used in the full implementation. In Fig. 4.1 we see that the adaptive memory algorithm using the given parameters, converges over time, but the error with the linked list approach increases quickly and in an unbounded manner. There is some flexibility on setting parameters, and the larger η is, the more accurate the simulation is, but the longer it takes (similar to the adaptive step parameter a). We leave as an open problem the analysis of the error resulting from the linked implementation, including whether it converges over time and its exact dependence on η. Even in the case that error does not ever converge, one can set the value of η depending on the required length of the simulation and how much error one is willing to tolerate during the simulation.

In Fig. 4.2 we also include a comparison of computational times for the simulations used in Fig. 4.1. It is clear that the adaptive memory and linked list approaches are much faster than the full implementation. However, the linked list simulation was not considerably faster than the adaptive step approach. If the simulation time was extended further, we would see this gap widen, however, as the $NlogN$ vs N^2 scaling effects came into play. Considering both Fig. 4.1 and 4.2, we may come to the conclusion that in many practical cases, the adaptive memory approach may provide the most reasonable trade-off between execution time and accuracy.

5 Conclusion

We have successfully characterized the stability of two finite difference algorithms used in solving the time-fractional diffusion equation. We provide our rationale for selecting a von Neumann analysis while also exploring a few alternative interpretations and approaches to analyzing stability. Using a von Neumann analysis we have developed bounding expressions for parameters like time step, spatial discretization, and grid size, that can be used to appropriately set parameters to ensure accurate simulations that reflect the true solution of the fractional diffusion equation. Our simulation results verify that our bounding expressions resulting from our stability analysis are valid and accurate. We also note here that the stability analysis of the linked list implementation should be explored, and we leave this as an open problem for further consideration.

We have also successfully characterized the algorithmic complexity of three finite difference algorithms introduced in [7]. We find that the full and adaptive step algorithms have a Big-O complexity of $O\left(N^2\right)$ while the linked list scheme performs much better with $O\left(Nlog_2N\right)$ complexity. We compared our analytical results with simulated data and find they are in good agreement.

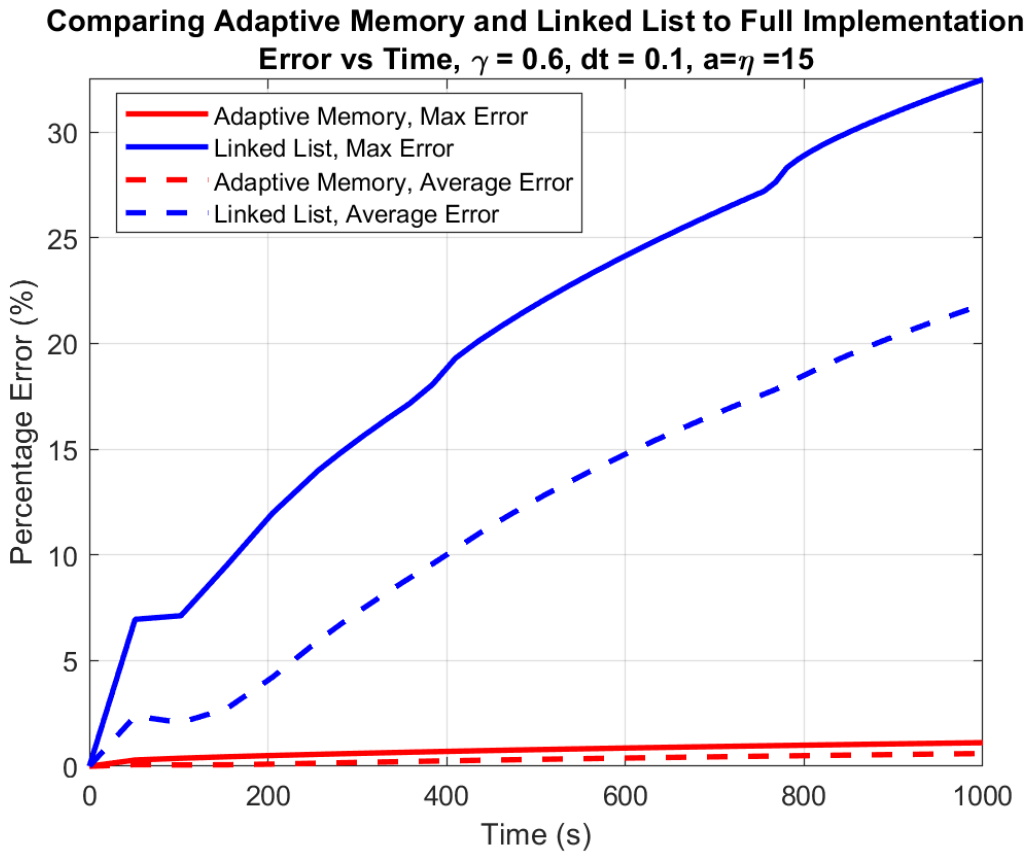


Figure 4.1: Comparing error of adaptive memory and linked list algorithms, using the full implementation as the reference. For simulations, the following parameters were used: $\gamma = 0.6$, $\alpha = \beta = 50\ \frac{units^2}{s^3}$, $dt = 0.1s$, $\Delta x = \Delta y = 8.64\ units$, $N_x = N_y = 20$, $a = \eta = 15$. We include both maximum error at each timepoint, and average error over the whole solution, at each timepoint.

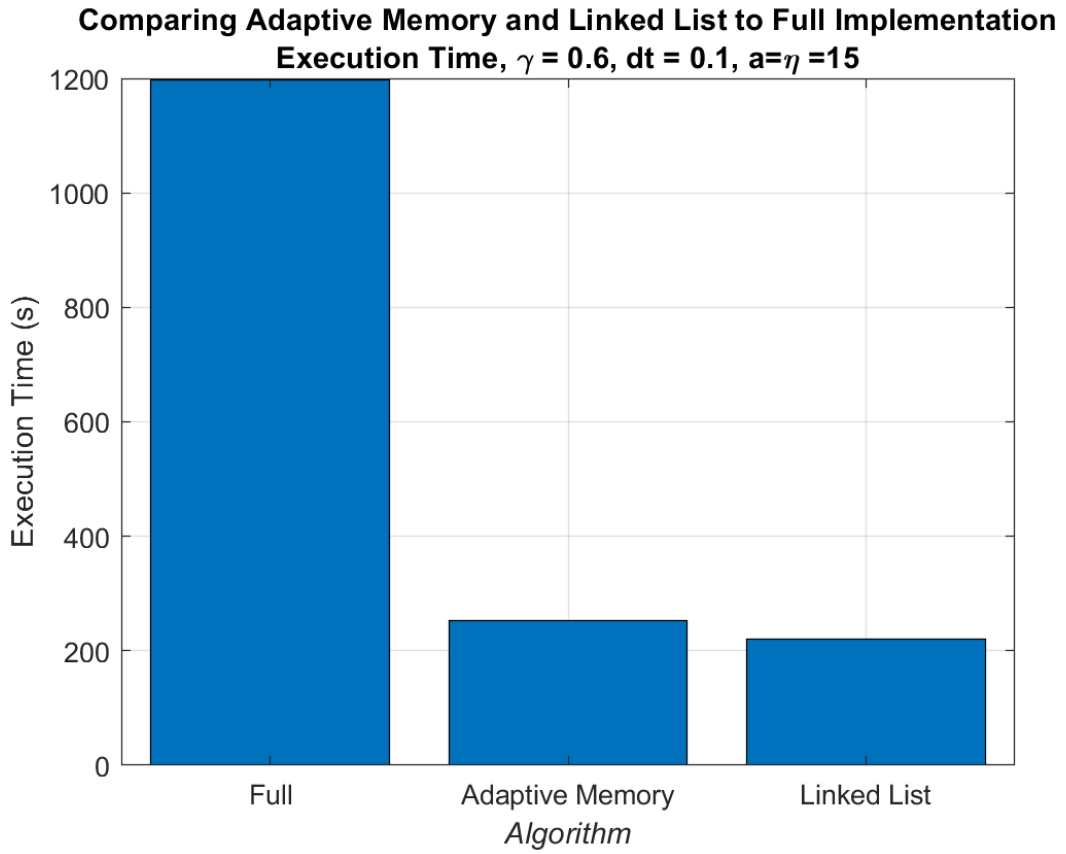


Figure 4.2: Comparing execution times between all three algorithms. The following parameters were used (same as Fig. 4.1 $\gamma = 0.6$, $\alpha = \beta = 50\ \frac{units^2}{s^4}$, $dt = 0.1s$, $\Delta x = \Delta y = 8.64\ units$, $N_x = N_y = 20$, $a = \eta = 15$.

References

[1] Boris Baeumer, *Subordinated advection-dispersion equation for contaminant transport*, Water Resources . . . **37** (2001), no. 6, 1543–1550.

[2] N Bhattacharya, *Fractional Diffusion : Numerical Methods and Applications in Neuroscience*, Ph.D. Dissertation in Bioengineering, UC San Diego (2014).

[3] ZEA Fellah, C Depollier, and M Fellah, *Application of fractional calculus to the sound waves propagation in rigid porous materials: Validation via ultrasonic measurements*, Acta Acustica united with . . . **88** (2002), 34–39.

[4] Joseph E. Flaherty, *Course notes - partial differential equations*, University Lecture, http://www.cs.rpi.edu/ flaherje/.

[5] B. Henry, T. Langlands, and S. Wearne, *Fractional cable models for spiny neuronal dendrites*, Physical Review Letters **100** (2008), no. 12, 128103.

[6] Charles Hirsch, *Numerical Computation of Internal and External Flows, Vol 1: Fundamentals of Numerical Discretization*, vol. 1, John Wiley & Sons Ltd., 1988.

[7] Christopher L MacDonald, Nirupama Bhattacharya, Brian P Sprouse, and Gabriel A Silva, *Efficient computation of the Grünwald-Letnikov fractional diffusion derivative using adaptive time step memory*, Journal of Computational Physics (2015).

[8] Richard L Magin, *Fractional Calculus in Bioengineering*, Begell House Publishers, January 2006.

[9] R.L. Magin and M. Ovadia, *Modeling the Cardiac Tissue Electrode Interface Using Fractional Calculus*, Journal of Vibration and Control **14** (2008), no. 9-10, 1431–1442.

[10] B. Mathieu, P. Melchior, a. Oustaloup, and Ch. Ceyral, *Fractional differentiation for edge detection*, Signal Processing **83** (2003), no. 11, 2421–2432.

[11] Parviz Moin, *Fundamentals of Engineering Numerical Analysis*, Cambridge University Press, 2010.

[12] Keith B. Oldham and Jerome Spanier, *The Fractional Calculus: Theory and Applications of Differentiation and Integration to Arbitrary Order*, Dover Books on Mathematics, April 2006.

[13] I. Podlubny, *Fractional Differential Equations*, Academic Press New York, 1999.

[14] N. Sebaa, Z.E.a. Fellah, W. Lauriks, and C. Depollier, *Application of fractional calculus to ultrasonic wave propagation in human cancellous bone*, Signal Processing **86** (2006), no. 10, 2668–2677.

[15] Eugeniusz Soczkiewicz, *Application of Fractional Calculus in the Theory of Viscoelasticity*, Molecular and Quantum Acoustics **23** (2002), 397–404.

[16] SB Yuste and L. Acedo, *On an explicit finite difference method for fractional diffusion equations*, Arxiv preprint cs/0311011 (2003).