Graduate J. Math. 3 (2018), 98 – 107

A Fast $O(N \log N)$ Second-Order Numerical Method For Space-Fractional Diffusion Equations

Treena Basu and Gregory Capra

Abstract

Fractional diffusion equations have been proven to accurately model anomalous diffusion processes in nature. However, numerical schemes applied to space-fractional diffusion equations result in dense or full coefficient matrices with computational complexity and storage capacity of $O(N^3)$ per time step and $O(N^2)$ respectively, which is increasingly problematic for larger N. This paper seeks to provide a more efficient and robust algorithm for numerically approximating a second-order accurate numerical solution to the discretized one-dimensional two-sided space-fractional diffusion equation that requires only $O(N \log N)$ computational work per time step and $O(N)$ memory by utilizing the Crank-Nicolson scheme and studying the structure of the resulting coefficient matrix. A fast iterative scheme is used to solve the resulting system of equations. Numerical results are shown to illustrate the second-order accuracy and efficiency of the new method.

MSC 2010. Primary: 65M06; Secondary: 65F05.

1 Introduction

In the recent times, fractional calculus has played a very important role in various fields like the natural sciences such as biology [4], geophysics [5], chemistry [8], social sciences such as finance [16, 23] and technology such as signal and image processing. Major topics include but are not limited to anomalous diffusion, continuous time random walk, Levy statistics, fractional Brownian motion, fractals, and groundwater problems [20]. The fractional order partial differential equation is a generalization of the classical partial differential equations. Using fractional order diffusion equations to model nonclassical phenomena such as anomalous diffusion has garnered great interest and is an emerging area of research that has many applications in the sciences; see for instance [9].

Closed-form analytic solutions have been developed for fractional partial diffusion equations using analytic methods such as the Fourier transform method, the Laplace transform method, and the Mellin transform method [3], but the fact remains that there are only a few instances where a closed-form analytic solution is available. This has lead to an urgent and growing need to design efficient and robust algorithms for numerical solutions of fractional partial differential equations. Due to the challenges that arise in terms of computational cost (and therefore CPU time) and high storage demand numerical methods for fractional partial differential equations such as finite difference methods [18], finite element methods [17], spectral methods [10] and discontinuous Galerkin methods [13, 24] remain an active area of research.

Numerical methods for solving these space-fractional diffusion equations are largely inefficient since the fractional partial derivative is a non-local operator and thus raise subtle stability issues on the corresponding numerical approximations. Further, numerical methods for fractional diffusion equations yield discrete systems with full or dense coefficient matrices that require computational cost of $O(N^3)$ per time step to numerically solve and a memory of $O(N^2)$ for a problem of size N. This is quite different from second-order diffusion equations which usually generate banded coefficient matrices with $O(N)$ non-zero entries and can be solved very efficiently by fast iterative methods with $O(N \log N)$ or $O(N)$ complexity. Thus, the question arises of whether a more efficient numerical method exists for solving these space-fractional diffusion equations.

Meershaert and Tadjeran showed in [11, 12] that a direct truncation of the Grunwald-Letnikov form of the fractional derivative, even though discretized implicitly in time, leads to unstable discretizations. They further established that by implementing a shifted Grünwald discretization to approximate the fractional diffusion equation the scheme is unconditionally stable and that the corresponding finite difference scheme convergences. Numerical experiments conducted showed that these methods yield satisfactory results. One of the issues that remain is that the shifted Grünwald discretization is only first-order accurate in space. In [19] Tadjeran, Meerschaert and Scheffler developed a Crank-Nicolson (CN) scheme which is second-order accurate in time. In order to recover second-order accuracy in space they utilized the Richardson extrapolation. But challenges still remain in that these methods generate full coefficient matrices which were inverted using Gaussian elimination and thus require $O(N^3)$ computational work per time step and $O(N^2)$ storage.

H.Wang, K.Wang and Sircar established in [22] that the stiffness matrix that results from application of a finite difference method on the two-sided space-fractional diffusion equation can be decomposed into a combination of sparse and structured dense matrices. They demonstrated that this matrix decomposition avoided the need to store the full coefficient matrix and thus allowed them to reduce the storage requirement of the coefficient matrix from $O(N^2)$ to $O(N)$. In addition to being able to reduce the storage requirement of the stiffness matrix, in [1] the authors were able to engineer a second-order accurate numerical solution using an operator-splitting technique to split the stiffness matrix as the sum of a banded matrix and a remaining matrix. The resulting algorithm had a computational cost of $O(N(\log N)^2)$ per time step, a significant improvement from the previous $O(N^3)$ per time step of traditional methods.

This paper proposes a numerical approach for the one-dimensional two-sided space-fractional diffusion equation that is second-order accurate in space and time and can overcome the obstacles of massive memory requirement of $O(N^3)$ per time step and enormous computational cost of $O(N^2)$ typically needed to develop a numerical solution. The two main goals of this work is to be able to design an algorithm that can drastically reduce the computational cost further from $O(N(\log N)^2)$ per time step that was proved in [1] to $O(N \log N)$ while still keeping keeping the reduced memory requirement of $O(N)$ as was done in previous work, see [22]. The fractional diffusion equation is discretized by the implicit Crank-Nicolson scheme with the shifted Grünwald formula and using an iterative scheme, namely the conjugate gradient squared method to solve the resulting system of equations as was done by the authors in [21]. This provides us with an opportunity to perform fast matrix-vector multiplication and thereby reduce the complexity cost from $O(N^3)$ to $O(N \log N)$ when using the iterative scheme.

The rest of this paper is organized as follows. In Section 2 we present the one-dimensional two-sided space-fractional diffusion equation and its Crank-Nicolson finite difference approximation. We will also write out the structure of the resulting coefficient matrix for future analysis and examination. We conclude this section with a discussion of the impact of computational cost and memory requirement of the finite difference method. In Section 3 we visit the conjugate gradient squared (CGS) iterative method for solving the system of equations resulting from the Crank-Nicolson scheme. We will also discuss the cost of performing the CGS method and its storage requirements. Next, in Section 4 we analyze the structure of the coefficient matrix and establish a technique to store the matrix in exactly $O(N)$ memory without the need for compression. This is in contrast to traditional methods that typically require $O(N^2)$ storage. Section 5 describes how to implement a fast matrix-vector multiplication approach to speed up the performance of the conjugate gradient squared iterative scheme. This section concludes with the fact that the fast method has a computational work of $O(N \log N)$ per iteration in contrast to traditional methods that require $O(N^3)$ computational cost per time step.

Numerical Experiments in Section 6 will investigate the performance of the newly developed fast algorithm, F2FDE, which consists of the Crank-Nicolson method together with the fast conjugate gradient squared method and Richardson extrapolation and compare it with the following four more standard techniques: the traditional Crank-Nicolson method with Gaussian elimination (CN), the traditional Crank- Nicolson method along with a Richardson extrapolation (CNE) (both of which were developed in [19]), the Crank-Nicolson scheme with the traditional conjugate gradient squared method (CN-CGS), and the Crank-Nicolson method with the traditional conjugate gradient squared method and Richardson extrapolation (CN-CGSE). In each of the above five methods mentioned we will present tables that analyze method accuracy and CPU time. Finally, in Section 7 we draw our conclusions and discuss possible future research.

2 The One-Dimensional Two-Sided Space-Fractional Diffusion Equation and its Finite Difference Approximation

In the last few decades, more and more anomalous diffusion phenomena have been found in the real world, which lead to fractional diffusion equations [9].

The initial-boundary value problem under consideration is the following two-sided space-fractional diffusion equation with the parameter $1 < \alpha < 2$ being the fractional order of the spatial derivative:

$$\frac{\partial u(x,t)}{\partial t} - d_+(x,t)\frac{\partial^\alpha u(x,t)}{\partial_+ x^\alpha} - d_-(x,t)\frac{\partial^\alpha u(x,t)}{\partial_- x^\alpha} = f(x,t). \tag{2.1}$$

subject to

$$x_L \le x \le x_R, \quad 0 < t \le T,$$

$$u(x_L,t) = 0, \quad u(x_R,t) = 0. \tag{2.2}$$

$$u(x,0) = u_0(x)$$

The functions $d_+(x,t)$ and $d_-(x,t)$ are the transport related coefficients and the function $f(x,t)$ is a source/sink term. The two-sided fractional diffusion equation allows modeling different flow regime impacts from either side. The left-sided (+) and right-sided (-) fractional derivatives $\dfrac{\partial^\alpha u(x,t)}{\partial_+ x^\alpha}$ and $\dfrac{\partial^\alpha u(x,t)}{\partial_- x^\alpha}$ in (2.1) are the Grünwald-Letnikov fractional derivatives of order α defined by [15, 14]:

$$\frac{\partial^\alpha u(x,t)}{\partial_+ x^\alpha} = \lim_{h\to 0^+} \frac{1}{h^\alpha} \sum_{k=0}^{\lfloor (x-x_L)/h \rfloor} g_k^{(\alpha)} u(x-kh,t),$$
$$\frac{\partial^\alpha u(x,t)}{\partial_- x^\alpha} = \lim_{h\to 0^+} \frac{1}{h^\alpha} \sum_{k=0}^{\lfloor (x_R-x)/h \rfloor} g_k^{(\alpha)} u(x+kh,t)$$

where $\lfloor x \rfloor$ represents the floor of x and the Grünwald weights $g_k^{(\alpha)}$ are defined as:

$$g_k^{(\alpha)} = (-1)^k \binom{\alpha}{k}$$

where $\binom{\alpha}{k}$ represents fractional binomial coefficients. We note that the Grünwald weights $g_k^{(\alpha)}$ have the recursive relation

$$g_0^{(\alpha)} = 1, \qquad g_k^{(\alpha)} = \left(1 - \frac{\alpha+1}{k}\right) g_{k-1}^{(\alpha)} \quad \text{for} \quad k \ge 1.$$

Moreover, for $1 < \alpha < 2$ the coefficients $g_k^{(\alpha)}$ satisfy the following properties [15, 11, 12]:

$$\begin{cases} g_0^{(\alpha)} = 1, \qquad g_1^{(\alpha)} = -\alpha < 0, \\ 1 \ge g_2^{(\alpha)} \ge g_3^{(\alpha)} \ge \cdots \ge 0, \\ \sum_{k=0}^{\infty} g_k^{(\alpha)} = 0, \qquad \sum_{k=0}^{m} g_k^{(\alpha)} < 0 \quad (m \ge 1). \end{cases} \tag{2.3}$$

Through these definitions one should observe the fact that these fractional derivatives are non-local operators. In other words, the left-sided fractional derivative of u at a point depends on all function values to the left of that point. Similarly, the right-sided fractional derivative of u at a point depends on all function values to the right of this point.

This paper focuses on the development of a second-order accurate numerical solution to Equation (2.1). To discretize this problem we will take as input two positive integers N and M, where the step size h is defined as $h = \dfrac{(x_R - x_L)}{N}$ and the time step Δt is defined as $\Delta t = \dfrac{T}{M}$. The spatial and temporal partitions therefore become $x_i = x_L + ih$ for $i = 0, 1, \ldots, N$ and $t^m = m\Delta t$ for $m = 0, 1, \ldots, M$. Let $u_i^m = u(x_i, t^m)$, $d_{+,i}^m = d_+(x_i,t^m)$, $d_{-,i}^m = d_-(x_i,t^m)$, and $f_i^m = f(x_i,t^m)$.

The shifted Grünwald approximations, which rid the diffusion Equation (2.1) of fractional partial derivatives, are as such:

$$\frac{\partial^\alpha u(x_i,t^m)}{\partial_+ x^\alpha} = \frac{1}{h^\alpha} \sum_{k=0}^{i+1} g_k^{(\alpha)} u_{i-k+1}^m + a_1 h + O(h^2),$$
$$\frac{\partial^\alpha u(x_i,t^m)}{\partial_- x^\alpha} = \frac{1}{h^\alpha} \sum_{k=0}^{N-i+1} g_k^{(\alpha)} u_{i+k-1}^m + b_1 h + O(h^2) \tag{2.4}$$

where a_1 and b_1 do not depend on the grid size h and $O(h^2)$ represents the remaining terms converging to zero at least as fast as some constant multiple of h^2. The use of the shifted Grüwald estimates is required to stabilize the finite difference scheme.

By discretizing the first-order time derivative in Equation (2.1) by the standard first-order time difference quotient and using the shifted Grünwald approximation in (2.4) above the Crank-Nicolson finite difference method, when applied to Equation (2.1) yields the following scheme which proven to be unconditionally stable and convergent by Tadjeran, Meerschaert and Scheffler in [19]:

$$\begin{aligned} &\frac{1}{2}\Big(f_i^{m+1} + f_i^m\Big) \\ &= \frac{u_i^{m+1} - u_i^m}{\Delta t} \\ &-\frac{1}{2}\Big(\frac{d_{+,i}^{m+1}}{h^\alpha} \sum_{k=0}^{i+1} g_k^{(\alpha)} u_{i-k+1}^{m+1} - \frac{d_{+,i}^{m}}{h^\alpha} \sum_{k=0}^{i+1} g_k^{(\alpha)} u_{i-k+1}^{m}\Big) \\ &-\frac{1}{2}\Big(\frac{d_{-,i}^{m+1}}{h^\alpha} \sum_{k=0}^{N-i+1} g_k^{(\alpha)} u_{i+k-1}^{m+1} - \frac{d_{-,i}^{m}}{h^\alpha} \sum_{k=0}^{N-i+1} g_k^{(\alpha)} u_{i+k-1}^{m}\Big) \end{aligned}$$

To illustrate the pattern of the stiffness matrices A^{m+1} and A^m, we list the corresponding first two equations for $i = 1$ and $i = 2$:

i=1

$$\begin{aligned} &\frac{u_1^{m+1} - u_1^m}{\Delta t} - \frac{d_{+,1}^{m+1}}{2h^\alpha} \sum_{k=0}^{2} g_k^{(\alpha)} u_{2-k}^{m+1} + \frac{d_{+,1}^{m}}{2h^\alpha} \sum_{k=0}^{2} g_k^{(\alpha)} u_{2-k}^{m} \\ &- \frac{d_{-,1}^{m+1}}{2h^\alpha} \sum_{k=0}^{N} g_k^{(\alpha)} u_k^{m+1} + \frac{d_{-,1}^{m}}{2h^\alpha} \sum_{k=0}^{N} g_k^{(\alpha)} u_k^{m} \\ &= \frac{1}{2}\left(f_1^{m+1} + f_1^m\right) \end{aligned} \tag{2.5}$$

i=2

$$\frac{u_2^{m+1}-u_2^m}{\Delta t} - \frac{d_{+,2}^{m+1}}{2h^\alpha}\sum_{k=0}^{3} g_k^{(\alpha)} u_{3-k}^{m+1} + \frac{d_{+,2}^{m}}{2h^\alpha}\sum_{k=0}^{3} g_k^{(\alpha)} u_{3-k}^{m} - \frac{d_{-,2}^{m+1}}{2h^\alpha}\sum_{k=0}^{N-1} g_k^{(\alpha)} u_{1+k}^{m+1} + \frac{d_{-,2}^{m}}{2h^\alpha}\sum_{k=0}^{N-1} g_k^{(\alpha)} u_{1+k}^{m} = \frac{1}{2}\left(f_2^{m+1}+f_2^m\right) \tag{2.6}$$

Expanding the summations in Equations (2.5) and (2.6), utilizing our knowledge of boundary conditions, and canceling and combining like terms one can establish how the matrices A^{m+1} and A^m below are obtained.

Let the vector $u^{m+1} = \left[u_1^{m+1} \quad u_2^{m+1} \ldots u_{N-1}^{m+1}\right]$ and likewise, $u^m = [u_1^m \quad u_2^m \ldots u_{N-1}^m]$. Also,

$$f^{m+1} = \left[f_1^{m+1} \quad f_2^{m+1} \ldots f_{N-1}^{m+1}\right]$$

and likewise, $f^m = [f_1^m \quad f_2^m \ldots f_{N-1}^m]$. The numerical scheme in matrix form is given by

$$\left(I + \frac{\Delta t}{2h^\alpha}A^{m+1}\right)u^{m+1} = \left(I - \frac{\Delta t}{2h^\alpha}A^{m}\right)u^{m} + \frac{\Delta t}{2}\left(f^m + f^{m+1}\right). \tag{2.7}$$

where I is the identity matrix of order $N-1$ and the entries of the coefficient matrix $A^{m+1} = [a_{i,j}^{m+1}]_{i,j=1}^{N-1}$ are given by

$$a_{i,j}^{m+1} = \begin{cases} -\left(d_{+,i}^{m+1} + d_{-,i}^{m+1}\right) g_1^{(\alpha)}, & j = i, \\ -\left(d_{+,i}^{m+1} g_2^{(\alpha)} + d_{-,i}^{m+1} g_0^{(\alpha)}\right), & j = i-1, \\ -\left(d_{+,i}^{m+1} g_0^{(\alpha)} + d_{-,i}^{m+1} g_2^{(\alpha)}\right), & j = i+1, \\ -d_{+,i}^{m+1} g_{i-j+1}^{(\alpha)}, & j < i-1, \\ -d_{-,i}^{m+1} g_{j-i+1}^{(\alpha)}, & j > i+1. \end{cases} \tag{2.8}$$

with A^m being defined similarly.

Using Equation (2.8) we conclude this section with an investigation into the nature of A^{m+1} from Equation (2.3) and on this note we observe the following [22]:

- the entries of A^{m+1} are such that $a_{ij}^{m+1} \leq 0$ for $i \neq j$;
- the coefficient matrix $I + \left(\frac{\Delta t}{2h^\alpha}\right)A^{m+1}$ is a non-singular, strictly diagonally dominant M-matrix;
- the coefficient matrix is full with a complicated structure; consequently it requires a storage of $O(N^2)$.

Due to its complex structure most numerical methods for fractional diffusion equations have been solved by Gaussian elimination, which requires computational work of $O(N^3)$ per time step [12, 19]. On the other hand, a traditional finite difference discretization of a second-order analogue of problem (2.1) yields a sparse coefficient matrix that requires far less storage and has reduced complexity. This significantly increased computational work and memory requirement of the numerical methods for fractional diffusion equations motivates the development of fast and robust numerical methods with an efficient storage mechanism for these problems.

It remains to be analyzed what kind of information and how much information is contained in the stiffness matrices A^{m+1} and A^m. The question regarding what kind of structure it possesses also remains to be answered and will be addressed in Section 4 where storage management of A^m and A^{m+1} will be discussed.

3 Applying the Conjugate Gradient Squared Method

When analyzing the computational cost and memory of the traditional numerical methods to approximate the solution of the space-fractional diffusion equation one will see there is much left to be desired. Taking up an enormous amount of time and space is the simple fact that inversion of a matrix takes $O(N^3)$ computation per time step and $O(N^2)$ memory. At this point in the paper we propose a popular iterative scheme for non-symmetric matrices, namely the Conjugate Gradient Squared (CGS) Method [6]. This method, an extension of the Conjugate Gradient method for symmetric matrices, will give us our desired numerical solution vector u^{m+1} with a significantly less computational cost of $O(N^2)$- (memory will remain the same as we still will have to store A^m and A^{m+1}) and therefore reduce our overall time complexity substantially.

The CGS algorithm applied to the matrix equation in (2.7) is listed step-by-step, below:

```
At each time step $t^{m+1}$, we choose $u^{(0)} = u^m$
Compute $r^{(0)} := \left(I - \frac{\Delta t}{2h^\alpha}A^m\right)u^m + \frac{\Delta t}{2}\left(f^m + f^{m+1}\right) - \left(I + \frac{\Delta t}{2h^\alpha}A^{m+1}\right)u^{(0)}$
Choose $\tilde{r}$ (for example, $\tilde{r} = r^{(0)}$).
for $i = 1, 2, \ldots$
    $\rho_{i-1} := \tilde{r}^T r^{(i-1)}$
    if $\rho_{i-1} = 0$ choose $\tilde{r} := r^{(i-1)}$
        if $i = 1$
            $w^{(1)} := r^{(0)}$
            $p^{(1)} := w^{(1)}$
        else
            $\nu_{i-1} := \rho_{i-1}/\rho_{i-2}$
            $w^{(i)} := r^{(i-1)} + \nu_{i-1} q^{(i-1)}$
            $p^{(i)} := w^{(i)} + \nu_{i-1}\left(q^{(i-1)} + \nu_{i-1} p^{(i-1)}\right)$
        end if
        $\hat{v} := \left(I + \frac{\Delta t}{2h^\alpha} A^{m+1}\right) p^{(i)}$
        $\mu_i := \rho_{i-1} / \tilde{r}^T \hat{v}$
        $q^{(i)} := w^{(i)} - \mu_i \hat{v}$
        $u^{(i)} := u^{(i-1)} + \mu_i \left(w^{(i)} + q^{(i)}\right)$
        $\hat{q} := \left(I + \frac{\Delta t}{2h^\alpha} A^{m+1}\right)\left(w^{(i)} + q^{(i)}\right)$
        $r^{(i)} := r^{(i-1)} - \mu_i \hat{q}$
        $\delta := \left\| \left(I - \frac{\Delta t}{2h^\alpha} A^m\right) u^m + \frac{\Delta t}{2}\left(f^m + f^{m+1}\right) - \left(I + \frac{\Delta t}{2h^\alpha} A^{m+1}\right) u^{(i)} \right\|$
        Check if convergence criteria < specified
        tolerance, continue if needed
end
$u^m := u^{(i)}$
```

Although the CGS method is, at its worst, $O(N^2)$ per time step, its computation time can be significantly reduced if the five matrix-vector multiplications $A^m u^m$, $A^{m+1}u^{(0)}$, $A^{m+1}p^{(i)}$, $A^{m+1}(w^{(i)} + q^{(i)})$, $A^{m+1}u^{(i)}$ are given attention. All other computational costs are of linear computational cost $O(N)$.

Note that the storage of the stiffness matrices still require $O(N^2)$ memory. In the next two sections we will investigate the ways in which we can efficiently store A^{m+1} and A^m and how we can accelerate the five matrix-vector multiplications mentioned above by reducing required operations from $O(N^2)$ to $O(N \log N)$.

4 An Efficient Storage of Coefficient Matrix of $O(N)$

In this section we explore our options to reduce the storage of the coefficient matrices. To execute scheme (2.7) requires storing the two matrices A^{m+1} and A^m which have $2(N-1)^2$ parameters. To this effect we note that an opportunity for memory savings arises if we decompose A^{m+1} and A^m as:

$$\begin{aligned} A^{m+1} &= -\mathrm{diag}\left(d_+^{m+1}\right) A_{L,N-1} - \mathrm{diag}\left(d_-^{m+1}\right) A_{R,N-1}, \\ A^m &= -\mathrm{diag}\left(d_+^{m}\right) A_{L,N-1} - \mathrm{diag}\left(d_-^{m}\right) A_{R,N-1}. \end{aligned} \tag{4.1}$$

where

- $\mathrm{diag}\left(d_+^{m+1}\right)$, $\mathrm{diag}\left(d_-^{m+1}\right)$, $\mathrm{diag}\left(d_+^{m}\right)$, and $\mathrm{diag}\left(d_-^{m}\right)$ are diagonal matrices of order $N-1$ with their ith entries $d_{+,i}^{m+1}$, $d_{-,i}^{m+1}$, $d_{+,i}^{m}$, and $d_{-,i}^{m}$ for $i = 1, 2, \ldots, N-1$, i.e., the diffusion coefficients line the main diagonal of their own matrices (zero everywhere else). Thus, they need not be stored in their entireity but rather can be stored as columns vectors $d_+^{m+1} = \left[d_{+,1}^{m+1}, d_{+,2}^{m+1}, \ldots, d_{+,N-1}^{m+1}\right]^T$,
  $d_-^{m+1} = \left[d_{-,1}^{m+1}, d_{-,2}^{m+1}, \ldots, d_{-,N-1}^{m+1}\right]^T$,
  $d_+^{m} = \left[d_{+,1}^{m}, d_{+,2}^{m}, \ldots, d_{+,N-1}^{m}\right]^T$,
  $d_-^{m} = \left[d_{-,1}^{m}, d_{-,2}^{m}, \ldots, d_{-,N-1}^{m}\right]^T$
  which would require only $4N-4$ parameters.
- and the matrices $A_{L,N-1}$ and $A_{R,N-1}$ are matrices of order $(N-1)$ and are defined by

$$A_{L,N-1} = \begin{bmatrix} g_1^{(\alpha)} & g_0^{(\alpha)} & 0 & \dots & 0 & 0 \\ g_2^{(\alpha)} & g_1^{(\alpha)} & g_0^{(\alpha)} & \ddots & \ddots & 0 \\ \vdots & g_2^{(\alpha)} & g_1^{(\alpha)} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ g_{N-2}^{(\alpha)} & \ddots & \ddots & \ddots & g_1^{(\alpha)} & g_0^{(\alpha)} \\ g_{N-1}^{(\alpha)} & g_{N-2}^{(\alpha)} & \dots & \dots & g_2^{(\alpha)} & g_1^{(\alpha)} \end{bmatrix},$$

$$A_{R,N-1} = \begin{bmatrix} g_1^{(\alpha)} & g_2^{(\alpha)} & \dots & \dots & g_{N-2}^{(\alpha)} & g_{N-1}^{(\alpha)} \\ g_0^{(\alpha)} & g_1^{(\alpha)} & g_2^{(\alpha)} & \dots & \ddots & g_{N-2}^{(\alpha)} \\ 0 & g_0^{(\alpha)} & g_1^{(\alpha)} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \ddots & g_1^{(\alpha)} & g_2^{(\alpha)} \\ 0 & 0 & \dots & 0 & g_0^{(\alpha)} & g_1^{(\alpha)} \end{bmatrix}.$$

Note that the matrices $A_{L,N-1}$ and $A_{R,N-1}$ are Toeplitz matrices. A Toepltiz matrix is a matrix in which each descending diagonal from left to right is constant. An $n \times n$ Toeplitz matrix T_n is completely determined by a sequence of $2n-1$ numbers $t_{i\,i=1-n}^{\,n-1}$ such that the (i,j)th entry of the matrix $T_n(i,j) = t_{j-i}$ for $i, j = 1, 2, \cdots, n$, see [7, 2]. Therefore, instead of storing the two matrices $A_{L,N-1}$ and $A_{R,N-1}$ we can store the vector fractional binomial coefficient vector $\mathrm{g}^{(\alpha)} = \left[g_0^{(\alpha)}, g_1^{(\alpha)}, \ldots, g_{N-1}^{(\alpha)}\right]^T$ which requires only N parameters.

Thus the total storage requirement for A^{m+1} and A^m comes down from $2(N-1)^2$ to a total of $4N - 4 + N = 5N - 4$ parameters, i.e. the memory requirement can be reduced from $O(N^2)$ to $O(N)$.

Also, note that $\mathrm{g}^{(\alpha)}$ depends only on the size of the spatial partition and the order of the anomalous diffusion but is independent of time or space and so it can be calculated and stored in advance.

5 A Fast Matrix-Vector Multiplication of $O(N \log N)$ for Evaluating $A^{m+1}u$ and $A^m u$

In the previous section we discussed the ways in which we can circumvent the need to ever store the full matrices A^{m+1} and A^m. In this section we establish the way in which we can achieve an $O(N \log N)$ matrix-vector multiplication algorithm that will speed up the CGS iterative method. Below we outline the steps required to efficiently multiply the matrices A^{m+1} and A^m that are of order $(N-1)$ with any vector u of length $(N-1)$ that utilizes the matrix decomposition in Equation (4.1).

1. An opportunity for speeding up the CGS algorithm arises which begins by embedding the Toeplitz matrices $A_{L,N-1}$ and $A_{R,N-1}$ of size $(N-1)\times(N-1)$ into Circulant matrices $C_{L,2N-2}$ and $C_{R,2N-2}$ of size $(2N-2)\times(2N-2)$. A Circulant matrix is a matrix in which each row vector is rotated one element to the right relative to the preceding row vector. An $n \times n$ Circulant matrix C_n is completely determined by a sequence of n numbers $c_{i\,i=0}^{\,n-1}$ such that the (i,j)-th entry of the matrix $C_n(i,j) = c_{(j-i) mod n}$ for $i,j = 1,2,\cdots,n$, see [7, 2]. Due to the cyclical structure of such a matrix, only one column or row of a Circulant matrix is needed to recreate the rest.

   The left and right Circulant matrices can be created with the following block structure [2]:

$$\begin{aligned} C_{L,2N-2} &= \begin{bmatrix} A_{L,N-1} & B_{L,N-1} \\ B_{L,N-1} & A_{L,N-1} \end{bmatrix}, \\ C_{R,2N-2} &= \begin{bmatrix} A_{R,N-1} & B_{R,N-1} \\ B_{R,N-1} & A_{R,N-1} \end{bmatrix} \end{aligned} \tag{5.1}$$

   where $B_{L,N-1}$ and $B_{R,N-1}$ are Toeplitz matrices of order $(N-1)$ defined to be

$$\begin{aligned} B_{L,N-1} &= \begin{bmatrix} 0 & g^{\alpha}_{N-1} & \cdots & \cdots & g^{\alpha}_3 & g^{\alpha}_2 \\ 0 & 0 & g^{\alpha}_{N-1} & \cdots & \ddots & g^{\alpha}_3 \\ 0 & 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \ddots & 0 & g^{\alpha}_{N-1} \\ g^{\alpha}_0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix}, \\ B_{R,N-1} &= \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & g^{\alpha}_0 \\ g^{\alpha}_{N-1} & 0 & 0 & \ddots & \ddots & 0 \\ \vdots & g^{\alpha}_{N-1} & 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ g^{\alpha}_3 & \ddots & \ddots & \ddots & 0 & 0 \\ g^{\alpha}_2 & g^{\alpha}_3 & \cdots & \cdots & g^{\alpha}_{N-1} & 0 \end{bmatrix} \end{aligned} \tag{5.2}$$

   We also create a $(2N-2)$ vector u_{2N-2} defined as $u_{2N-2} = \begin{bmatrix} u \\ 0 \end{bmatrix}$. In other words, we double the vector u by appending it with $(N-1)$ zeros. Now, instead of performing $A_{L,N-1}u$ and $A_{R,N-1}u$, we will seek to perform $C_{L,2N-2}u_{2N-2}$ and $C_{R,2N-2}u_{2N-2}$. Then the computations will yield:

$$C_{L,2N-2}u_{2N-2} = \begin{bmatrix} A_{L,N-1}u \\ B_{L,N-1}u \end{bmatrix} \tag{5.3}$$

$$\text{and} \quad C_{R,2N-2}u_{2N-2} = \begin{bmatrix} A_{R,N-1}u \\ B_{R,N-1}u \end{bmatrix}$$

   Thus, $A_{L,N-1}u$ and $A_{R,N-1}u$ can be obtained by capturing the first half of the computations in Equation (5.3).

2. Next we utilize the well known decomposition of Circulant matrices using the discrete Fourier transform [2, 7]

$$\begin{aligned} C_{L,2N-2} &= F^{-1}_{2N-2} diag(F_{2N-2}c_L)F_{2N-2}, \quad \text{and} \\ C_{R,2N-2} &= F^{-1}_{2N-2} diag(F_{2N-2}c_R)F_{2N-2} \end{aligned}$$

   where c_L and c_R are the first columns of the $(2N-2)$ x $(2N-2)$ Circulant matrix $C_{L,2N-2}$ and $C_{R,2N-2}$ defined as

$$\begin{aligned} c_L &= [g^{(\alpha)}_1, g^{(\alpha)}_2 \cdots, g^{(\alpha)}_{N-1}, 0, 0, \cdots g^{(\alpha)}_0]^T, \text{ and} \\ c_R &= [g^{(\alpha)}_1, g^{(\alpha)}_0, \cdots, 0, 0, g^{(\alpha)}_{N-1}, \cdots, g^{(\alpha)}_2]^T \end{aligned}$$

   F_{2N-2} is the $(2N-2)$ x $(2N-2)$ discrete Fourier transform matrix, and F^{-1}_{2N-2} is the inverse Fourier transform matrix of same size. Note how this computation only requires storing one column vector from each of the Circulant matrices $C_{L,2N-2}$ and $C_{R,2N-2}$ to be stored. We then execute the following actions to compute $C_{L,2N-2}u_{2N-2}$ and $C_{R,2N-2}u_{2N-2}$:

   (a) Perform the matrix-vector multiplication $w_{2N-2} = F_{2N-2}u_{2N-2}$, which can be achieved in $O((2N-2)\log(2N-2)) = O(N \log N)$ operations since $F_{2N-2}u_{2N-2}$ is the discrete Fourier transform of u_{2N-2} via the fast Fourier transform (FFT).

   (b) Likewise perform, $v_{L,2N-2} = F_{2N-2}c_L$ and $v_{R,2N-2} = F_{2N-2}c_R$, which can also be done in $O(N \log N)$ operations via FFT.

   (c) Perform element-wise multiplications

$$\begin{aligned} & z_{L,2N-2} \\ =\ & w_{2N-2} \cdot v_{L,2N-2} \\ =\ & [w_1 v_{L,1}, w_2 v_{L,2}, \cdots w_{2N-2} v_{L,2N-2}]^T \end{aligned}$$

   and

$$\begin{aligned} & z_{R,2N-2} \\ =\ & w_{2N-2} \cdot v_{R,2N-2} \\ =\ & [w_1 v_{R,1}, w_2 v_{R,2}, \cdots w_{2N-2} v_{R,2N-2}]^T \end{aligned}$$

   which requires $O(N)$ operations.

   (d) Evaluate $y_{L,2N-2} = F^{-1}_{2N-2} z_{L,2N-2}$ and $y_{R,2N-2} = F^{-1}_{2N-2} z_{R,2N-2}$ in $O(N \log N)$ operations using inverse FFT. Utilizing Equations (5.1) and (5.3) we get

$$y_{L,2N-2} = \begin{bmatrix} y_L \\ y_L' \end{bmatrix} = C_{L,2N-2}u_{2N-2} = \begin{bmatrix} A_{L,N-1}u \\ B_{L,N-1}u \end{bmatrix},$$

$$y_{R,2N-2} = \begin{bmatrix} y_R \\ y_R' \end{bmatrix} = C_{R,2N-2}u_{2N-2} = \begin{bmatrix} A_{R,N-1}u \\ B_{R,N-1}u \end{bmatrix}.$$

(e) Finally, execute the element-wise products $u_L^{m+1} = d_+^{m+1} \cdot y_L$, $u_R^{m+1} = d_-^{m+1} \cdot y_R$ and $u_L^m = d_+^m \cdot y_L$, $u_R^m = d_-^m \cdot y_R$ in $O(N)$ operations. Use the decompositions of Equation (4.1) to evaluate $A^{m+1}u = -u_L^{m+1} - u_R^{m+1}$ and evaluate $A^m u = -u_L^m - u_R^m$ in $O(N)$ operations.

We will apply the above steps to each of the five matrix-vector multiplications

$$A^m u^m,\ A^{m+1}u^{(0)},\ A^{m+1}p^{(i)},\ A^{m+1}(w^{(i)}+q^{(i)}),\ A^{m+1}u^{(i)}$$

of the CGS method. The overall computational complexity of specifically this method therefore changes from $O(N^2)$ to $O(N \log N)$ per time step. The memory requirement changes from $O(N^2)$ to $O(N)$, as previously demonstrated. The numerical experiment below puts all of the following together in our proposed fast second-order finite difference method with extrapolation (F2FDE) algorithm: the Crank-Nicolson scheme, the conjugate gradient squared method with fast matrix-vector multiplication algorithm and the Richardson extrapolation.

6 Numerical Experiments

In this section we will present numerical experiments to study the performance of our newly proposed fast second-order finite difference method with extrapolation (F2FDE) and compare it with the following four more standard techniques: the Crank-Nicolson method with Gaussian elimination (CN), the Crank-Nicolson method and Richardson extrapolation (CNE), the Crank-Nicolson scheme with conjugate gradient squared method (CN-CGS), and the Crank-Nicolson method with the conjugate gradient squared method and Richardson extrapolation (CN-CGSE).

We consider the fractional diffusion equation (2.1) with an anomalous diffusion of order $\alpha = 1.8$ and the left-sided and right-sided diffusion coefficients

$$d_+(x,t) = 1.32\Gamma(1.2)x^{1.8},$$

$$d_-(x,t) = 1.32\Gamma(1.2)(1-x)^{1.8}.$$

The spatial domain is $[x_L, x_R] = [0,1]$, the time interval is $[0,T] = [0,1]$. The source term and the initial condition are given by

$$\begin{aligned} f(x,t) &= -16e^{1-t}\Big[x^2(1-x^2) + 2.64(x^2 + (1-x)^2) \\ &\quad -13.2\Big(x^3 + (1-x)^3\Big) + 12\Big(x^4 + (1-x)^4\Big)\Big], \\ u_0(x) &= 16ex^2(1-x)^2. \end{aligned}$$

The true solution to the corresponding fractional diffusion equation (2.1) is given by [12]

$$u(x,t) = 16e^{1-t}x^2(1-x)^2.$$

We will test each of the methods in this paper with values of N & M from 2^6 through 2^9. We will also discuss the quantitative metrics memory efficiency and computational cost for evaluating the effectiveness for each of the five numerical schemes and even though we do not present a theoretical proof of the stability of the proposed numerical scheme, the numerical results presented in Table 1 and Table 2 below indicate that the new scheme has the same stability constraint as the standard finite difference scheme which was proven to be stable in [19].

6.1 Comparison of the Crank-Nicolson method with Gaussian elimination (CN) and the Crank-Nicolson method with CGS (CN-CGS)

Here we solve equation (2.7) first by using matrix inversion via Gaussian elimination (CN method) and denote the numerical solution at time step t^m as u_{CN}^m and followed by Crank-Nicolson Method in combination with the CGS method (CN-CGS method) and denote the corresponding numerical solution at time step t^m by u_{CN-CGS}^m. Let $u^m = u(x,t^m)$ be the true solution to problem (2.1) at time step t^m. We present the error norms $||u_{CN}^M - u^M||_{L^\infty}$ between the numerical solution and the true solution for the CN method and $||u_{CN-CGS}^M - u^M||_{L^\infty}$ for the CN-CGS method for varying mesh sizes in Table 1.

The Crank-Nicolson finite difference approximation introduces second-order convergence in time to both the CN method as well as the CN-CGS method but is first-order in space because of the computationally feasible Grünwald-Letnikov definition of fractional derivative we used for the space-fractional diffusion equation in (2.1). In other words, the overall convergence for these methods is $O(\Delta x) + O((\Delta t)^2)$. The table above shows that the actual error converges as theorized. The ratio of L_∞ norm errors as $N = M$ increases converges almost exactly to $\frac{1}{2}$. This demonstrates that, as $N = M$ is doubled, the error effectively reduces by a factor of two, which supports the theory that this method has linear convergence in regards to space.

Method	$N = M$	Error $\|\cdot\|_{L^\infty}$	Observed Order	CPU time
CN	2^6	1.546558×10^{-2}	-	$7.21 \times 10^0 = 7.21$ s
	2^7	7.765908×10^{-3}	0.502141	$5.84 \times 10^1 = 58.4$ s
	2^8	3.891277×10^{-3}	0.501071	$4.70 \times 10^2 = 7$ m 50 s
	2^9	1.947725×10^{-3}	0.500536	$3.72 \times 10^3 = 1$ h 2 m
	2^{10}	9.743850×10^{-4}	0.500268	$3.00 \times 10^4 = 8$ h 20 m
CN-CGS	2^6	1.546558×10^{-2}	-	$6.59 \times 10^0 = 6.59$ s
	2^7	7.765908×10^{-3}	0.502141	$5.20 \times 10^1 = 52$ s
	2^8	3.891277×10^{-3}	0.501071	$4.03 \times 10^2 = 6$ m 43 s
	2^9	1.947725×10^{-3}	0.500536	$3.19 \times 10^3 = 53$ m 10 s
	2^{10}	9.743850×10^{-4}	0.500268	$2.64 \times 10^4 =$7 h 20 m

Tab. 1: Comparison of the Crank-Nicolson (CN) method with the Crank-Nicolson with CGS (CN-CGS) method in the simulation of the fractional diffusion problem with a known analytic solution

The bulk of the memory used in these approaches comes from the storage of the two $(N-1) \times (N-1)$ matrices A^{m+1} and A^m, so the storage requirement for both methods is $O(N^2)$. Of course, there are a number of vectors that are stored in temporary memory in the process, but these all add simply $O(kN)$ to the memory requirement- where k is the number of size $(N-1)$ vectors being initialized in each method.

The computational complexity of the CN method rests primarily on the matrix inversion of A^{m+1} an $O(N^3)$ process, which is very inefficient for large values of N and a source of motivation to be improved upon. For the CN-CGS method there is a considerable computational benefit in that matrix inversion is no longer required, instead the computational cost rests on the matrix-vector multiplications within the CGS method. There are five of these, and they each bring in $O(N^2)$ computation. The overall computation cost is therefore $O(N^2)$ per time step, a large improvement from the CN method.

6.2 Comparison of the Crank-Nicolson with the Richardson Extrapolation (CNE), Crank-Nicolson Method with CGS and Extrapolation (CN-CGSE), and Fast Second-Order Finite Difference Method with Extrapolation (F2FDE)

In [19] the authors used a spatial recovery process- known as the Richardson Extrapolation. This involves finding the numerical solution $u_{\Delta x}$ on a course grid Δx and then finding the numerical solution $u_{\frac{\Delta x}{2}}$ on a fine grid $\frac{\Delta x}{2}$ and then computing the extrapolated solution on the coarse spatial grid Δx by $u_{\Delta x} = 2u_{\frac{\Delta x}{2}} - u_{\Delta x}$. They also demonstrated that the extrapolated solution has second-order accuracy in both space and time $O((\Delta x)^2) + O((\Delta t)^2)$.

Table 2 next lists the error norms $||u^M_{CNE} - u^M||_{L^\infty}$, $||u^M_{CN-CGSE} - u^M||_{L^\infty}$ and $||u^M_{F2FDE} - u^M||_{L^\infty}$ along with the CPU time for the three processes: the Crank-Nicolson with the Richardson Extrapolation (CNE), Crank-Nicolson Method with CGS and Extrapolation (CN-CGSE), and Fast Second-Order Finite Difference Method with Extrapolation (F2FDE) from $N = M = 64$ to $N = M = 512$. In the case of the newly proposed fast method F2FDE we also present results for the case $N = M = 1024$.

The errors for all three methods CNE, CN-CGSE and F2FDE that utilize the Richardson extrapolation converge almost exactly as predicted: $O((\Delta x)^2)+O((\Delta t)^2)$. The ratio of L_∞ norm errors as $N = M$ doubles converges to $\frac{1}{4}$. In other words, as the inputs N and M double, the error reduces by a factor of 4. Thus, we can see the $O((\Delta x)^2)$ convergence.

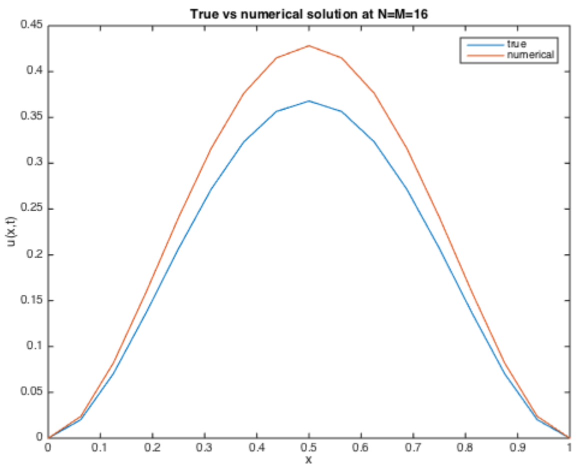
The Crank-Nicolson method with extrapolation (CNE) still requires overall $O(N^2)$ storage for memory and the overall computation cost remains $O(N^3)$ per time step. In the CN-CGSE method the overall computational cost and memory remains unchanged from the CN-CGS method as the Richardson Extrapolation acts to improve only the rate of convergence in the spatial dimension. Thus, the overall computational cost per time step is $O(N^2)$ and memory $O(N^2)$.

In the case of the new fast second-order finite difference method F2FDE with Richardson extrapolation memory requirement is now $O(N)$ due to the decomposition that utilizes the properties of Circulant matrices. Computation cost within the CGS component of the method was reduced from $O(N^2)$ to $O(N \log N)$, which is verified in the table by looking at the ratio of the completion time.

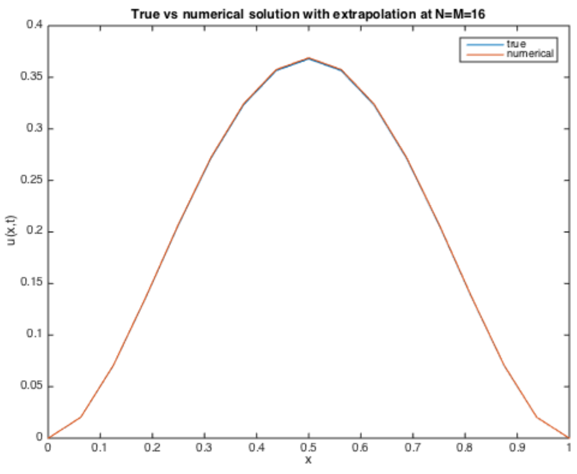
Below is a representative plot with $N = M = 16$ in Figure 1, which shows that the fast second-order finite difference method numerical solution and the standard Crank-Nicolson without extrapolation solution both sit on the curve of the true solution without stark differences. All numerical calculations were carried out using MATLAB. The reader should note that implementing

Method	N=M	Error $\|\cdot\|_{L^\infty}$	Observed Order	CPU (time)
CNE	2^6	6.623621×10^{-5}	-	$6.06 \times 10^1 = 60.1$ s
	2^7	1.664738×10^{-5}	0.251333	$4.82 \times 10^2 = 8$ m 2 s
	2^8	4.172939×10^{-6}	0.250666	$3.83 \times 10^3 = 1$ h 3 m 50 s
	2^9	1.044624×10^{-6}	0.250333	$2.97 \times 10^4 = 8$ h 15 m
CN-CGSE	2^6	6.623622×10^{-5}	-	$5.91 \times 10^1 = 59.1$ s
	2^7	1.664739×10^{-5}	0.251333	$4.58 \times 10^2 = 7$ m 38 s
	2^8	4.172940×10^{-6}	0.250666	$3.65 \times 10^3 = 1$ h 50 s
	2^9	1.044624×10^{-6}	0.250333	$2.91 \times 10^4 = 8$ h 5 m
F2FDE	2^6	6.623622×10^{-5}	-	$2.60 \times 10^0 = 2.60$ s
	2^7	1.664739×10^{-5}	0.251333	$9.19 \times 10^0 = 9.19$ s
	2^8	4.172940×10^{-6}	0.250666	$3.52 \times 10^1 = 35$ s
	2^9	1.044625×10^{-6}	0.250333	$1.42 \times 10^2 = 2$ m 22 s
	2^{10}	2.613301×10^{-7}	0.250166	$6.05 \times 10^2 = 10$ m 5 s

Tab. 2: Comparison of the Crank-Nicolson method with Extrapolation (CNE), the Crank-Nicolson with CGS and Extrapolation (CN-CGSE) and Fast Second-Order Finite Difference Method with Extrapolation (F2FDE)



(a) CN run on N=M=16



(b) F2FDE run on N=M=16

Fig. 1: Plot of the true solution versus the numerical solution

the fast second-order finite difference method only requires that all of the matrix-vector multiplications of the standard conjugate gradient squared method be replaced by fast matrix-vector multiplication code which is based on FFT, a call function readily available in MATLAB.

In summary, numerical experiments presented above show a significant reduction in computational time, which coincides with our theoretical analysis. For instance, to obtain a second-order accurate solution in space and time with $N = M = 512$ nodes, the fast second-order finite difference method F2FDE developed in this paper has about 209 times of CPU reduction than the standard Crank-Nicolson scheme solved with Gaussian elimination. This is in addition to the significant reduction in memory needed to develop a numerical solution to Equation (2.7). This demonstrates the utility and potential of the method.

7 Concluding Remarks

Fractional partial differential equations have applications in biology, chemistry, finance, and more. In the recent past many numerical methods have been developed for space-fractional diffusion equations, but techniques for obtaining accurate approximations to these equations are often computationally expensive and require a lot of data storage in the form of the stiffness matrix, and impose serious challenges in the case of higher dimension problems.

This paper develops a fast second-order accurate solution method for the implicit finite difference scheme given by Equation (2.7) for the one-dimensional space-fractional diffusion equation developed by Meerschaert, Tadjeran and Scheffler in [19]. The newly proposed fast method F2FDE reduces the computational cost from $O(N^3)$ to $O(N \log N)$ per time step and memory requirement from $O(N^2)$ to $O(N)$ while the Richardson extrapolation allows us to recover second-order spatial accuracy improving upon the typical convergence rate of the Crank-Nicolson scheme of $O(\Delta x) + O((\Delta t)^2)$ to second-order convergence rate in space and time of $O((\Delta x)^2) + O((\Delta t)^2)$.

The fast and efficient algorithm developed in this paper can be applied to other numerical methods. Also, corresponding fast versions can be implemented with relative ease for other conjugate gradient or Krylov subspace types of iterative methods.

Acknowledgments:
The authors want to thank the anonymous reviewers for the careful reading of the manuscript and the fruitful remarks.

References

[1] T. S. Basu and H. Wang. *A Fast Second-Order Finite Difference Method for Space-Fractional Diffusion Equations.* International Journal of Numerical Analysis and Modeling, Volume 9, Number 3, pp.658-666, 2011.

[2] P. J. Davis. *Circulant Matrices.* John Wiley & Sons, New York, 1979.

[3] L. Debnath, *Linear Partial Differential Equations for Scientists and Engineers, fourth edition*

[4] N. Gal, D. Weihs, Experimental evidence of strong anomalous diffusion in living cells, *Phys. Rev. E 81 (2010) 020903 (4pp)*

[5] V. Ganti, M.M. Meerschaert, E.F. Georgiou, E. Viparelli, G. Parker, Normal and anomalous diffusion of gravel tracer particles in rivers, *J. Geophysics.Res. 115 (2010) F00A07 (15pp)*

[6] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Third Edition, Oct 1996.

[7] R. M. Gray. *Toeplitz and circulant matrices: A review.* Foundations and Trends in Communications and Information Theory, 2, pp. 155-239, 2006.

[8] J.W. Kirchner, X. Feng, and C. Neal, Fractal stream chemistry and its implications for containant transport in catchments, *Nature* 403 (2000), 524-526.

[9] J. Klafter and I. M. Sokolov. *Anomalous diffusion spreads its wings.* Physics World. *physicsweb.org*, August, 2005.

[10] Y. Lin and C. Xu, Finite difference/spectral approximations for the time-fractional diffusion equation, *J. Comput. Phys.*, 225 (2007) 1533-1552.

[11] M.M. Meerschaert and C. Tadjeran, Finite difference approximations for fractional advection-dispersion flow equations, *J. Comput. Appl. Math.*, 172 (2004) 65-77.

[12] M.M. Meerschaert and C. Tadjeran, Finite difference approximations for two-sided space-fractional partial differential equations, *Appl. Numer. Math.*, 56 (2006) 80-90.

[13] K. Mustapha, W. McLean, Superconvergence of a discontinuous Galerkin method for fractional diffusion and wave equations, *SIAM J. Numer. Anal. 51 (1) (2013) 491-515*

[14] K.B. Oldham and J. Spanier, *The Fractional Calculus*, Academic Press, New York, 1974.

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

[16] M. Raberto, E. Scalas, and F. Mainardi, Waiting-times and returns in high-frequency financial data: an empirical study, *Physica* 314 (2002), 749-755.

[17] J.P. Roop, Computational aspects of FEM approximation of fractional advection dispersion equations on bounded domains in R^2, *J. Comput. Appl. Math.* 193, (2006) 243-268

[18] E. Sousa, Finite difference approximates for a fractional advection diffusion problem, *J. Comput. Phys.* 228 (2009), 4038-4054.

[19] C. Tadjeran, M.M. Meerschaert, and H.P. Scheffler, A second-order accurate numerical approximation for the fractional diffusion equation, *J. Comput. Phys.*, 213 (2006) 205-213.

[20] G. E. Uhlenbeck and L. S. Ornstein. *On the Theory of the Brownian Motion.* Physics Review, Volume 36, Number 5, pp. 823-841, Sep 1930.

[21] H. Wang and T.S. Basu. *A Fast Finite Difference Method for Two-Dimensional Space-Fractional Diffusion Equations.* SIAM J. Sci. Comput. Vol. 34, No. 5, pp. A2444-A2458.

[22] H. Wang, K. Wang, and T. Sircar, A direct $O(N \log^2 N)$ finite difference method for fractional diffusion equations, *J. Comput. Phys.*, 229:8095-8104, 2010.

[23] H. O. A. Wold and P. Whittle. *A Model Explaining the Pareto Distribution of Wealth.* Econometrica, Volume 25, Number 4, pp. 591-595, Oct 1957.

[24] Q. Xu, J.S. Hesthaven, Discontinuous Galerkin method for fractional convection-diffusion equations, *SIAM J. Numer. Anal. 52(1) (2013) 405-423*

Treena Basu, Gregory Capra
1600 Campus Road, Department of Mathematics,
Occidental College, Los Angeles, CA, USA.
E-mail: `basu@oxy.edu` , `gregoryrcapra@gmail.com`