Electronic Transactions on Numerical Analysis.
Volume 65, pp. 281–307, 2026.

ISSN 1068–9613.
DOI: 10.1553/etna_vol65s281

ETNA
Kent State University and
Johann Radon Institute (RICAM)

NUMERICAL APPROXIMATION OF CAPUTO-TYPE ADVECTION-DIFFUSION EQUATIONS VIA SHIFTED CHEBYSHEV POLYNOMIALS*

FRANCISCO DE LA HOZ† AND PERU MUNIAIN‡

Abstract. In this paper, using a pseudospectral approach, we develop operational matrices based on the shifted Chebyshev polynomials to numerically approximate Caputo fractional derivatives and Riemann–Liouville fractional integrals. To make the generation of these matrices stable, we use variable precision arithmetic. Then, we apply the Caputo differentiation matrices to numerically solve Caputo-type advection-diffusion equations in one and in multiple spatial dimensions, which involves transforming the discretization of the equation under consideration into a Sylvester (tensor) equation. We provide complete Matlab codes, whose implementation is carefully explained. The numerical experiments involving highly oscillatory functions in time confirm the effectiveness of this approach.

Key words. Caputo fractional derivative, Riemann–Liouville fractional integral, Caputo-type advection-diffusion equations, pseudospectral methods, shifted Chebyshev polynomials, Sylvester equations, Sylvester tensor equations

AMS subject classifications. 15A24, 15A69, 26A33, 35R11, 65M70

1. Introduction.

Fractional calculus (see, e.g., [24] for a complete overview) generalizes the ideas of differentiation and integration to non-integer orders. In this paper we consider the Caputo fractional derivative, which is one possible definition of a fractional derivative. The Caputo fractional derivative is defined as follows: Let the order $\alpha \in (0,\infty)$ be such that $0 \le n-1 < \alpha < n$, for $n \in \mathbb{N}$. Then, the Caputo fractional derivative of order α of a function $f(t) \in \mathcal{C}^n(0,\infty)$ is given by

$$D_t^\alpha f(t) = \frac{1}{\Gamma(n-\alpha)} \int_0^t \frac{f^{(n)}(\tau)}{(t-\tau)^{1-n+\alpha}} d\tau, \tag{1.1}$$

where $\Gamma(\cdot)$ denotes Euler's gamma function. Unlike the integer-order derivatives $f'(t)$, $f''(t)$, etc., which are local, fractional derivatives are nonlocal. However, it is immediate to see that

$$\begin{aligned} \lim_{\alpha \to n^-} D_t^\alpha f(t) &= f^{(n)}(t), \quad n \in \mathbb{N}, \\ \lim_{\alpha \to (n-1)^+} D_t^\alpha f(t) &= f^{(n-1)}(t) - f^{(n-1)}(0), \quad n \in \mathbb{N}, \end{aligned} \tag{1.2}$$

where, in order to get the second expression, we have integrated (1.1) by parts; in particular, $\lim_{\alpha\to 0^+} D_t^\alpha f(t) = f(t) - f(0)$. Therefore, the operator D_t^α is not continuous at any $\alpha \in \mathbb{N}$.

Likewise, let $\alpha \in (0,\infty)$. Then, the Riemann–Liouville fractional integral of a function $f(t) \in \mathcal{C}^0(0,\infty)$ is given by

$$I_t^\alpha f(t) = \frac{1}{\Gamma(\alpha)} \int_0^t \frac{f(\tau)}{(t-\tau)^{1-\alpha}} d\tau. \tag{1.3}$$

*Received December 11, 2025. Accepted April 10, 2026. Published online on July 3, 2026. Recommended by Mariarosa Mazza. Francisco de la Hoz was partially supported by the research group grant IT1615-22 funded by the Basque Government, and by the projects PID2021-126813NB-I00 and PID2024-158099NB-I00 funded by MICIU/AEI/10.13039/501100011033 and by ERDF, EU. Peru Muniain was partially supported by the research group grant IT1461-22 funded by the Basque Government, and by the project PID2022-139458NB-I00 funded by MICIU/AEI/10.13039/501100011033 and by ERDF, EU.

‡Department of Mathematics, Faculty of Science and Technology, University of the Basque Country UPV/EHU, Barrio Sarriena, S/N, 48980 Leioa, Spain.

‡Corresponding author. Department of Applied Mathematics, Escuela de Ingeniería de Bilbao, University of the Basque Country UPV/EHU, Plaza Ingeniero Torres Quevedo, 1, 48013 Bilbao, Spain (peru.muniain@ehu.eus).

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Note that $I_t^\alpha f(t)$ is not defined at $\alpha = 0$, but, integrating (1.3) by parts, it follows immediately that $\lim_{\alpha\to 0^+} I_t^\alpha f(t) = f(t)$.

In this paper we are mainly interested in numerically approximating (1.1) in $t \in [0, T]$. However, since the numerical treatment of (1.3) is quite similar, we also consider it. Let us mention that often other notation is used to denote (1.1) and (1.3). For instance, it is common to add, respectively, the super- or subscripts C (which stands for Caputo) or RL (which stands for Riemann–Liouville) by writing, e.g., ${}^C D_t^\alpha f(t)$ and ${}^{RL} I_t^\alpha f(t)$. However, since there is no risk of confusion here, we simply write $D_t^\alpha f(t)$ and $I_t^\alpha f(t)$.

Among the existing methods (see, e.g., the review [4] and its references), we focus on methods based on polynomial interpolation. This can be done, e.g., by means of Lagrange interpolating polynomials, as in [5, 20, 21], where $D_t^\alpha f(t)$ is approximated in the context of numerically solving Caputo-type advection-diffusion equations or, more recently, as in [10], where an FFT-based convolution algorithm applied to a modification of the methods in [5, 20, 21] allowed us to numerically approximate $D_t^\alpha f(t)$ for an extremely large number of nodes very efficiently. Moreover, in [10], after discretizing the time and space variables by means of operational matrices, Caputo-type advection-diffusion equations are transformed into a Sylvester equation of the form $\mathbf{A}\cdot\mathbf{X} + \mathbf{X}\cdot\mathbf{B} = \mathbf{C}$, which enabled us to consider the solution simultaneously at all time instants and across all spatial nodes.

On the other hand, it is also possible to perform the interpolation by means of a spectral method, and, in this regard, the so-called shifted Chebyshev polynomials $T_k^*(t)$, $t \in [0, 1]$, which will be used in this paper, seem a natural option. These polynomials are just the first-kind Chebyshev polynomials $T_k(t) = \cos(k \arccos(t))$ (see, e.g., [3, 25] for a complete exposition of their properties) evaluated at $2t - 1$, i.e.,

$$T_k^*(t) = T_k(2t-1) = \cos(k \arccos(2t-1)).$$

Then, given $N \in \mathbb{N} \cup \{0\}$, we approximate $f(t)$ as

$$f(t) \approx \sum_{k=0}^{N} \hat{f}_k T_k^*\left(\frac{t}{T}\right), \qquad t \in [0, T], \tag{1.4}$$

for a certain $T > 0$ and apply (1.1) and (1.3) to (1.4), getting, respectively,

$$D_t^\alpha f(t) \approx \sum_{k=0}^{N} \hat{f}_k D_t^\alpha \left(T_k^*\left(\frac{t}{T}\right)\right), \qquad t \in [0, T], \tag{1.5}$$

$$I_t^\alpha f(t) \approx \sum_{k=0}^{N} \hat{f}_k I_t^\alpha \left(T_k^*\left(\frac{t}{T}\right)\right), \qquad t \in [0, T]. \tag{1.6}$$

With regard to the actual implementation of (1.5) and/or (1.6) and differential equations involving them, it is possible to do it by constructing operational matrices, as in, e.g., [14, 16, 26], but even if such matrices are not explicitly defined, as in, e.g., [17], it is necessary to solve linear systems of equations.

The main inconvenience of using shifted Chebyshev polynomials for the numerical approximation of fractional derivatives and/or integrals is that the resulting methods are, in general, unstable, except for small values of N, which limits their applicability. Moreover, the existing theoretical results are rather involved; for instance, in [1, Theorem 1], the authors proved the following:

$$D_t^\alpha T_k(t) \approx \sum_{p=0}^{M} \Delta_{k,p}^{(\mu)} T_p(t), \qquad M \gg 1,$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

where

$$\Delta_{k,p}^{(\mu)} = \frac{2k(-1)^{k+\lceil\alpha\rceil}\Gamma(k+\lceil\alpha\rceil)\Gamma\left(\lceil\alpha\rceil-\alpha+\frac{1}{2}\right)\theta_p}{\Gamma(k-\lceil\alpha\rceil+1)} \times {}_4\tilde{F}_3\left(\begin{matrix}1, \lceil\alpha\rceil-k, k+\lceil\alpha\rceil, \lceil\alpha\rceil-\alpha+\frac{1}{2} \\ \lceil\alpha\rceil+\frac{1}{2}, \lceil\alpha\rceil-\alpha-p+1, \lceil\alpha\rceil-\alpha+p+1\end{matrix}\middle| 1\right).$$

Here, $\lceil\cdot\rceil$ denotes the ceiling function, which maps a real number to the smallest integer greater than or equal to it, and θ_p is given by

$$\theta_p = \begin{cases} \frac{1}{2}, & p = 0, \\ 1, & p \geq 1. \end{cases}$$

With regard to ${}_4F_3$ and ${}_4\tilde{F}_3$, they are particular cases of ${}_pF_q$ and ${}_p\tilde{F}_q$, respectively, taking $p = 4$ and $q = 3$. The hypergeometric function ${}_pF_q$ is defined by

$$(1.7) \qquad {}_pF_q(a_1,\ldots,a_p;b_1,\ldots,b_q;z) = \sum_{k=0}^{\infty} \frac{(a_1)_k\cdots(a_p)_k}{(b_1)_k\cdots(b_q)_k}\frac{z^k}{k!}, \qquad p,q \in \mathbb{N}\cup\{0\},$$

where $(a)_k$ denotes the Pochhammer symbol,

$$(a)_k = \begin{cases} a(a+1)\cdots(a+k-1), & k \in \mathbb{N}, \\ 1, & k = 0, \end{cases}$$

and the regularized hypergeometric function ${}_p\tilde{F}_q$ is defined by

$${}_p\tilde{F}_q(a_1,\ldots,a_p;b_1,\ldots,b_q;z) = \frac{{}_pF_q(a_1,\ldots,a_p;b_1,\ldots,b_q;z)}{\Gamma(b_1)\cdots\Gamma(b_q)}.$$

Bearing in mind the previous arguments, the main aim of this work is to generate fractional differentiation matrices and fractional integration matrices that are stable for any N, and for this we use arbitrary-precision arithmetic (vpa) in the intermediate steps. Let us remark that vpa has been used successfully in [6] to numerically evaluate the Gaussian hypergeometric function ${}_2F_1$ (which is also a particular case of (1.7), taking $p = 2$ and $q = 1$) in the context of the fractional Laplacian, which is a related operator but, to the best of our knowledge, has not been previously used to generate fractional operational matrices associated with the shifted Chebyshev polynomials.

The structure of this paper is as follows. In Section 2, we approximate a given function $f(t)$ in terms of shifted Chebyshev polynomials, as in (1.4), and express (1.5) and (1.6) by means of operational matrices. In Section 3, we carefully explain how to implement the matrices developed in Section 2 in Matlab [23] by means of vpa. In this regard, special care is taken in choosing the correct number of digits. Note that once the matrices have been obtained by means of vpa, they can be safely cast to 64-bit precision, but, except for small values of N, it is not possible to compute the matrices correctly without vpa. In Section 4, we perform several numerical tests, considering different values of α and N, obtaining very satisfactory results, especially for $\alpha \in (0,1)$. Finally, in Section 5, as a practical application, following the ideas in [9] and [10], we solve Caputo-type advection-diffusion equations in one and multiple spatial dimensions.

To help in understanding the implementation and reproduction of the numerical experiments, complete Matlab codes are offered. The codes are also available in the GitHub repository [11].

All the codes have been run on a Mainstream A+ Server AS-2024S-TR SUPERMICRO AMD, with 56 cores, 112 threads, 2.75 GHz, and 128 GB of RAM.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

2. A matrix-based pseudospectral method to numerically approximate $D_t^\alpha f(t)$ and $I_t^\alpha f(t)$. Given a sufficiently regular function $f(t)$ defined on $[0,T]$, with $T>0$, in order to numerically approximate (1.1) and (1.3) by means of (1.5) and (1.6), respectively, we consider a pseudospectral method, where, in order to determine the coefficients $\{\hat{f}_k\}$ in (1.4), we must impose the equality in (1.4) at $N+1$ given nodes.

2.1. Computation of $\{\hat{f}_k\}$. In order to obtain $\{\hat{f}_k\}$ in (1.4), we impose equality at the $N+1$ extreme points of $T_k^*(t/T)$:

$$t_j = \frac{T}{2}\left[1+\cos\left(\frac{j\pi}{N}\right)\right], \qquad 0\le j\le N. \tag{2.1}$$

Therefore, denoting by f_j the numerical approximation of $f(x_j)$, we get

(2.2)
$$\begin{aligned} f_j \equiv \sum_{k=0}^{N}\hat{f}_k T_k^*\left(\frac{t_j}{T}\right) = \sum_{k=0}^{N}\hat{f}_k T_k\left(2\frac{t_j}{T}-1\right) &= \sum_{k=0}^{N}\hat{f}_k T_k\left(\cos\left(\frac{j\pi}{N}\right)\right) \\ &= \sum_{k=0}^{N}\hat{f}_k\cos\left(\frac{jk\pi}{N}\right), \qquad 0\le j\le N, \end{aligned}$$

i.e., we have a discrete cosine transform. Then, representing the cosines in exponential form yields

$$\begin{aligned} f_j &= \sum_{k=0}^{N}\hat{f}_k\frac{e^{\mathrm{i}jk\pi/N}+e^{-\mathrm{i}jk\pi/N}}{2} = \frac{1}{2}\sum_{k=0}^{N}\hat{f}_k e^{2\mathrm{i}jk\pi/(2N)} + \frac{1}{2}\sum_{k=-N}^{0}\hat{f}_{-k}e^{2\mathrm{i}jk\pi/(2N)} \\ &= \hat{f}_0 + \frac{1}{2}\sum_{k=1}^{N-1}\hat{f}_k e^{2\mathrm{i}jk\pi/(2N)} + (-1)^j\hat{f}_N + \frac{1}{2}\sum_{k=N+1}^{2N-1}\hat{f}_{2N-k}e^{2\mathrm{i}jk\pi/(2N)}, \quad 0\le j\le N. \end{aligned}$$

On the other hand, if we replace j by $2N-j$ in the last equation, then

$$\begin{aligned} f_{2N-j} &= \hat{f}_0 + \frac{1}{2}\sum_{k=1}^{N-1}\hat{f}_k e^{2\mathrm{i}(2N-j)k\pi/(2N)} \\ &\qquad + (-1)^{2N-j}\hat{f}_N + \frac{1}{2}\sum_{k=N+1}^{2N-1}\hat{f}_{2N-k}e^{2\mathrm{i}(2N-j)k\pi/(2N)} \\ &= \hat{f}_0 + \frac{1}{2}\sum_{k=N+1}^{2N-1}\hat{f}_{2N-k}e^{-2\mathrm{i}j(2N-k)\pi/(2N)} \\ &\qquad + (-1)^j\hat{f}_N + \frac{1}{2}\sum_{k=1}^{N-1}\hat{f}_k e^{-2\mathrm{i}j(2N-k)\pi/(2N)} = f_j, \end{aligned}$$

where we have substituted k by $2N-k$ in the sums of the second line. Therefore, defining

$$g_j = \begin{cases} f_j, & 0\le j\le N, \\ f_{2N-j}, & N+1\le j\le 2N-1, \end{cases} \tag{2.3}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

and

$$\hat{g}_k = \begin{cases} \hat{f}_0, & k = 0, \\ \frac{1}{2}\hat{f}_k, & 1 \le k \le N-1, \\ \hat{f}_N, & k = N, \\ \frac{1}{2}\hat{f}_{2N-k}, & N+1 \le k \le 2N-1, \end{cases} \tag{2.4}$$

we conclude that

$$g_j = \sum_{k=0}^{2N-1} \hat{g}_k e^{2\mathrm{i}jk\pi/(2N)} \Longleftrightarrow \hat{g}_k = \frac{1}{2N} \sum_{j=0}^{2N-1} g_j e^{-2\mathrm{i}jk\pi/(2N)}, \tag{2.5}$$

i.e., $\{g_j\}$ and $\{\hat{g}_k\}$ are obtained from each other by means of an inverse discrete Fourier transform (IDFT) and a discrete Fourier transform (DFT), respectively. Therefore, to get $\{\hat{f}_k\}$ from $\{f_j\}$, we define $\{g_j\}$ according to (2.3), perform a DFT on these coefficients yielding $\{\hat{g}_k\}$ and then apply (2.4) to get $\{\hat{f}_k\}$. Likewise, to obtain $\{f_j\}$ from $\{\hat{f}_k\}$, we define $\{\hat{g}_k\}$ according to (2.4), perform an IDFT on these coefficients, which yields $\{g_j\}$, and then apply (2.3) to get $\{f_j\}$ but keeping only the indices $0 \le j \le N$.

2.1.1. Matrix-based computation of $\{\hat{f}_k\}$ from $\{f_j\}$. Bearing in mind the previous arguments, it is fairly straightforward to create a matrix $\mathbf{M} = [m_{ij}] \in \mathbb{R}^{(N+1)\times(N+1)}$ such that

$$\hat{\mathbf{f}} = \mathbf{M} \cdot \mathbf{f} \Longleftrightarrow \mathbf{f} = \mathbf{M}^{-1} \cdot \hat{\mathbf{f}}, \tag{2.6}$$

where

$$\mathbf{f} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_N \end{bmatrix}, \qquad \hat{\mathbf{f}} = \begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \vdots \\ \hat{f}_N \end{bmatrix}. \tag{2.7}$$

Indeed, combining (2.3) and (2.5),

$$\hat{g}_k = \frac{f_0}{2N} + \frac{1}{N} \sum_{j=1}^{N-1} f_j \cos\left(\frac{jk\pi}{N}\right) + \frac{(-1)^k f_N}{2N},$$

and hence, from (2.4),

$$\hat{f}_k = \begin{cases} \dfrac{f_0}{2N} + \dfrac{1}{N} \displaystyle\sum_{j=1}^{N-1} f_j + \dfrac{f_N}{2N}, & k = 0, \\ \dfrac{f_0}{N} + \dfrac{2}{N} \displaystyle\sum_{j=1}^{N-1} f_j \cos\left(\dfrac{jk\pi}{N}\right) + \dfrac{(-1)^k f_N}{N}, & 1 \le k \le N-1, \\ \dfrac{f_0}{2N} + \dfrac{1}{N} \displaystyle\sum_{j=1}^{N-1} (-1)^j f_j + \dfrac{1}{2N} f_N (-1)^N, & k = N. \end{cases}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Therefore, the entries of the symmetric matrix $\mathbf{M}$, with $m_{jk} = m_{kj}$, are given by

(2.8)
$$m_{jk} = \frac{2}{N}\varepsilon_{jk}\cos\left(\frac{jk\pi}{N}\right), \qquad \text{where } \varepsilon_{jk} = \begin{cases} \frac{1}{4}, & j,k \in \{0,N\}, \\ \frac{1}{2}, & j \in \{0,N\} \text{ and } 1 \le k \le N-1, \\ \frac{1}{2}, & 1 \le j \le N-1 \text{ and } k \in \{0,N\}, \\ 1, & 1 \le j,k \le N-1. \end{cases}$$

Note that the inverse of $\mathbf{M}$ is trivially obtained from (2.2):

$$[\mathbf{M}^{-1}]_{jk} = \cos\left(\frac{jk\pi}{N}\right), \qquad 0 \le j,k \le N.$$

2.2. Construction of the fractional differentiation and integration matrices. Coming back to the shifted Chebyshev polynomials $\{T_k^*(t)\}$, we are interested in obtaining their coefficients explicitly, i.e., finding $\{a_{k\ell}\}$, such that

(2.9)
$$T_k^*(t) = \sum_{\ell=0}^{k} a_{k\ell} t^\ell.$$

Although there are different ways to do this, we opt for using a recurrence to obtain $\{T_k^*(t)\}$ recursively, in a similar way as with the standard Chebyshev polynomials:

(2.10)
$$\begin{cases} T_0(t) = 1, \\ T_1(t) = t, \\ T_{k+1}(t) = 2tT_k(t) - T_{k-1}(t), \quad k \in \mathbb{N}. \end{cases}$$

Hence, replacing t by $2t-1$ in (2.10) and bearing in mind that $T_k^*(t) \equiv T_k(2t-1)$, we have

(2.11)
$$\begin{cases} T_0^*(t) = 1, \\ T_1^*(t) = 2t - 1, \\ T_{k+1}^*(t) = (4t-2)T_k^*(t) - T_{k-1}^*(t), \quad k \in \mathbb{N}. \end{cases}$$

In this way, we obtain the values of the coefficients $\{a_{k\ell}\}$ of $\{T_0^*, T_1^*, \ldots, T_N^*\}$ as defined in (2.9). We store them in an upper triangular matrix $\mathbf{C} = [c_{ij}] \in \mathbb{Z}^{(N+1)\times(N+1)}$ such that $c_{\ell+1,k+1} \equiv a_{k\ell}$, i.e., the coefficients of the kth polynomial $T_k^*(t)$ are stored at the $(k+1)$th column, and the constant coefficient a_{k0} of $T_k^*(t)$ occupies the first position of that column, the coefficient a_{k1} in front of t the second position, the coefficient a_{k2} in front of t^2 the third position, and so on. For instance, if $N = 5$, then

$$\begin{aligned} T_0(t) &= 1, \\ T_1(t) &= 2t - 1, \\ T_2(t) &= 8t^2 - 8t + 1, \\ T_3(t) &= 32t^3 - 48t^2 + 18t - 1, \\ T_4(t) &= 128t^4 - 256t^3 + 160t^2 - 32t + 1, \\ T_5(t) &= 512t^5 - 1280t^4 + 1120t^3 - 400t^2 + 50t - 1, \end{aligned}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

from which we get

$$\mathbf{C} = \begin{bmatrix} a_{00} & a_{10} & a_{20} & a_{30} & a_{40} & a_{50} \\ 0 & a_{11} & a_{21} & a_{31} & a_{41} & a_{51} \\ 0 & 0 & a_{22} & a_{32} & a_{42} & a_{52} \\ 0 & 0 & 0 & a_{33} & a_{43} & a_{53} \\ 0 & 0 & 0 & 0 & a_{44} & a_{54} \\ 0 & 0 & 0 & 0 & 0 & a_{55} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 0 & 2 & -8 & 18 & -32 & 50 \\ 0 & 0 & 8 & -48 & 160 & -400 \\ 0 & 0 & 0 & 32 & -256 & 1120 \\ 0 & 0 & 0 & 0 & 128 & -1280 \\ 0 & 0 & 0 & 0 & 0 & 512 \end{bmatrix};$$

note that the structure of the first row comes from the fact that $c_{1,k+1} = a_{k0}$ and that $a_{k0} = T_k^*(0) = T_k(-1) = (-1)^k$, for $0 \le k \le N$.

From a practical point of view, in order to generate $\mathbf{C}$ by using (2.11), we observe that if we multiply by t the polynomial corresponding to a given column, then this is equivalent to shifting the column one position downwards. Therefore, we first create an all-zero matrix $\mathbf{C}$ of order $N+1$, and the first two columns corresponding, respectively, to the first two equations in (2.11) are completely determined by assigning $c_{11} = 1$, $c_{12} = -1$, $c_{22} = 2$. Then, the jth column, for $3 \le j \le N+1$, is generated from the jth and $(j-1)$th columns according to the third equation in (2.11) by the following recurrence:

$$\begin{cases} c_{1j} = (-1)^{j-1}, \\ c_{ij} = 4c_{i-1,j-1} - 2c_{i,j-1} - c_{i,j-2}, \qquad 2 \le i \le j \le N+1. \end{cases} \tag{2.12}$$

After obtaining $\mathbf{C}$, and hence determining the coefficients $\{a_{k\ell}\}$ in (2.9), we introduce (2.9) in (1.4) getting

$$\begin{aligned} f(t) \approx \sum_{k=0}^{N} \sum_{\ell=0}^{k} a_{k\ell} \hat{f}_k \left(\frac{t}{T}\right)^{\ell} &= \sum_{k=0}^{N} \sum_{\ell=0}^{k} c_{\ell+1,k+1} \hat{f}_k \left(\frac{t}{T}\right)^{\ell} \\ &= \sum_{k=0}^{N} \sum_{\ell=0}^{N} c_{\ell+1,k+1} \hat{f}_k \left(\frac{t}{T}\right)^{\ell}, \end{aligned} \tag{2.13}$$

where we have used in the last equality that $\mathbf{C}$ is upper triangular. Therefore, in order to apply (1.1) and (1.3) to (2.13), we first consider their action on $f(t) = (t/T)^{\ell}$. In the case of (1.1), we recall that, when $n, \ell \in \mathbb{N} \cup \{0\}$,

$$\frac{d^n(t^{\ell})}{dt^n} = \begin{cases} \ell(\ell-1)\cdots(\ell+1-n)t^{\ell-n} = \dfrac{\Gamma(\ell+1)}{\Gamma(\ell+1-n)} t^{\ell-n}, & n \le \ell, \\ 0, & n > \ell. \end{cases}$$

Hence, introducing $f(t) = (t/T)^{\ell}$ in (1.1), with $\ell \in \mathbb{N} \cup \{0\}$, it follows that, when $\alpha > \ell$, $D_t^{\alpha}(t/T)^{\ell} = 0$, and, when $\alpha \le \ell$,

$$\begin{aligned} D_t^{\alpha} \left(\frac{t}{T}\right)^{\ell} &= \frac{\Gamma(\ell+1)}{T^{\ell}\Gamma(n-\alpha)\Gamma(\ell+1-n)} \int_0^t \frac{\tau^{\ell-n}}{(t-\tau)^{1-n+\alpha}} d\tau \\ &= \frac{\Gamma(\ell+1)t^{\ell-\alpha}}{T^{\ell}\Gamma(n-\alpha)\Gamma(\ell+1-n)} \int_0^1 z^{\ell+1-n-1}(1-z)^{n-1-\alpha} dz \\ &= \frac{\Gamma(\ell+1)t^{\ell-\alpha}}{T^{\ell}\Gamma(n-\alpha)\Gamma(\ell+1-n)} B(\ell+1-n, n-\alpha) \\ &= \frac{\Gamma(\ell+1)}{T^{\ell}\Gamma(\ell+1-\alpha)} t^{\ell-\alpha}, \end{aligned} \tag{2.14}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

where we have performed the change of variable $\tau = tz$ and where $\Gamma(\cdot)$ and $B(\cdot,\cdot)$ denote Euler's gamma and beta function, respectively. Note that (2.14) is formally valid when $\ell + 1 - \alpha \notin -\mathbb{Z} \cup \{0\}$. In particular, if $\alpha \leq \ell$ is a nonnegative integer, then

$$D_t^\alpha \left(\frac{t}{T}\right)^\ell = \frac{\ell(\ell-1)\cdots(\ell+1-\alpha)}{T^\ell} t^{\ell-\alpha},$$

which is precisely the standard, integer-order derivative of order α of $f(t) = (t/T)^\ell$.

With regard to (1.3), introducing $f(t) = (t/T)^\ell$ in (1.3), with $\ell \in \mathbb{N} \cup \{0\}$,

$$\begin{aligned} I_t^\alpha \left(\frac{t}{T}\right)^\ell &= \frac{1}{T^\ell \Gamma(\alpha)} \int_0^t \frac{\tau^\ell}{(t-\tau)^{1-\alpha}} d\tau = \frac{t^{\ell+\alpha}}{T^\ell \Gamma(\alpha)} \int_0^1 z^{\ell+1-1}(1-z)^{-1+\alpha} dz \\ &= \frac{t^{\ell+\alpha}}{T^\ell \Gamma(\alpha)} B(\ell+1,\alpha) = \frac{\Gamma(\ell+1)}{T^\ell \Gamma(\ell+1+\alpha)} t^{\ell+\alpha}; \end{aligned} \tag{2.15}$$

note that (2.15) is formally valid when $\ell + 1 + \alpha \notin -\mathbb{Z} \cup \{0\}$. Moreover, (2.15) is precisely (2.14) after replacing $-\alpha$ by α.

Bearing in mind the previous arguments, we numerically approximate (1.1) and (1.3) by applying the operators D_t^α and I_t^α to (2.13), respectively, and then evaluating the resulting expressions at $t = t_j$:

$$\begin{aligned} D_t^\alpha f(t_j) &\approx \sum_{k=0}^N \sum_{\ell=0}^N c_{\ell+1,k+1} \hat{f}_k D_t^\alpha \left(\frac{t_j}{T}\right)^\ell \\ &= \sum_{k=0}^N \left[\sum_{\ell=\lceil\alpha\rceil}^N t_j^{\ell-\alpha} \frac{\Gamma(\ell+1)}{T^\ell \Gamma(\ell+1-\alpha)} c_{\ell+1,k+1} \right] \hat{f}_k, \qquad 0 \leq j \leq N, \end{aligned} \tag{2.16}$$

where $\lceil\alpha\rceil \leq N$ and

$$\begin{aligned} I_t^\alpha f(t_j) &\approx \sum_{k=0}^N \sum_{\ell=0}^N c_{\ell+1,k+1} \hat{f}_k I_t^\alpha \left(\frac{t_j}{T}\right)^\ell \\ &= \sum_{k=0}^N \left[\sum_{\ell=0}^N t_j^{\ell+\alpha} \frac{\Gamma(\ell+1)}{T^\ell \Gamma(\ell+1+\alpha)} c_{\ell+1,k+1} \right] \hat{f}_k, \qquad 0 \leq j \leq N. \end{aligned} \tag{2.17}$$

Observe that, if $p(x)$ is a polynomial of degree smaller than or equal to N, then (2.13) is exact for $f(x) = p(x)$ and so are (2.16) and (2.17). Moreover, even if the operators D_t^α and I_t^α are defined in (1.1) and (1.3) for positive non-integer values of α, both approximations (2.16) and (2.17) are well defined for $\alpha \in \mathbb{N} \cup \{0\}$. More precisely, when $\alpha = 0$, $D_t^\alpha f(t_j) = I_t^\alpha f(t_j)$, and when $\alpha \in \mathbb{N} \cup \{0\}$, $D_t^\alpha f(t_j) \approx f^{(\alpha)}(t_j)$. The latter is coherent with (1.2) and is due to the fact that we are taking $\lceil\alpha\rceil$ as the lower limit of the inner sum in (2.16).

Coming back to (2.16) and (2.17), in order to express these identities in matrix form, we define, respectively, $\hat{\mathbf{D}}_t^\alpha = [d_{ij}] \in \mathbb{C}^{(N+1)\times(N+1)}$ and $\hat{\mathbf{E}}_t^\alpha = [e_{ij}] \in \mathbb{C}^{(N+1)\times(N+1)}$ as

$$d_{j+1,k+1} = \sum_{\ell=\lceil\alpha\rceil}^N t_j^{\ell-\alpha} \frac{\Gamma(\ell+1)}{T^\ell \Gamma(\ell+1-\alpha)} c_{\ell+1,k+1}, \qquad 0 \leq j,k \leq N, \tag{2.18}$$

$$e_{j+1,k+1} = \sum_{\ell=0}^N t_j^{\ell+\alpha} \frac{\Gamma(\ell+1)}{T^\ell \Gamma(\ell+1+\alpha)} c_{\ell+1,k+1}, \qquad 0 \leq j,k \leq N, \tag{2.19}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

where, to avoid burdening the notation, we omit the subscript $_t$ and the superscript $^\alpha$ in $d_{j+1,k+1}$ and $e_{j+1,k+1}$. Then,

$$\begin{bmatrix} D_t^\alpha f(t_0) \\ D_t^\alpha f(t_1) \\ \vdots \\ D_t^\alpha f(t_N) \end{bmatrix} \approx \hat{\mathbf{D}}_t^\alpha \cdot \hat{\mathbf{f}}, \qquad \begin{bmatrix} I_t^\alpha f(t_0) \\ I_t^\alpha f(t_1) \\ \vdots \\ I_t^\alpha f(t_N) \end{bmatrix} \approx \hat{\mathbf{E}}_t^\alpha \cdot \hat{\mathbf{f}}, \tag{2.20}$$

where $\hat{\mathbf{f}}$ is defined in (2.7). Note that it is possible to approximate $\{D_t^\alpha f(t_j)\}$ and $\{I_t^\alpha f(t_j)\}$ directly from $\{f_j\}$ by means of $\mathbf{M}$ defined in (2.8). Indeed, from (2.6) we get

$$\begin{bmatrix} D_t^\alpha f(t_0) \\ D_t^\alpha f(t_1) \\ \vdots \\ D_t^\alpha f(t_N) \end{bmatrix} \approx \mathbf{D}_t^\alpha \cdot \mathbf{f}, \qquad \begin{bmatrix} I_t^\alpha f(t_0) \\ I_t^\alpha f(t_1) \\ \vdots \\ I_t^\alpha f(t_N) \end{bmatrix} \approx \mathbf{E}_t^\alpha \cdot \mathbf{f}, \tag{2.21}$$

where $\mathbf{f}$ is defined in (2.7), and

$$\mathbf{D}_t^\alpha = \hat{\mathbf{D}}_t^\alpha \cdot \mathbf{M}, \qquad \mathbf{E}_t^\alpha = \hat{\mathbf{E}}_t^\alpha \cdot \mathbf{M}. \tag{2.22}$$

3. Implementation in Matlab. Given a sufficiently regular function $f(t)$, in order to implement (2.20), we need to compute, on the one hand, the coefficients $\{\hat{f}_k\}$, for $0 \le k \le N$, and, on the other hand, generate the matrices $\hat{\mathbf{D}}_t^\alpha$ and $\hat{\mathbf{E}}_t^\alpha$. We remark that we can safely operate with 64-bit precision to obtain $\{\hat{f}_k\}$ and that the final version of $\hat{\mathbf{D}}_t^\alpha$ and $\hat{\mathbf{E}}_t^\alpha$ can also be stored with 64-bit precision, as vpa is only necessary in the intermediate steps of the construction of $\hat{\mathbf{D}}_t^\alpha$ and $\hat{\mathbf{E}}_t^\alpha$. Furthermore, once $\hat{\mathbf{D}}_t^\alpha$ and $\hat{\mathbf{E}}_t^\alpha$ have been obtained, it is straightforward to construct $\mathbf{D}_t^\alpha$ and $\mathbf{E}_t^\alpha$ from (2.22) and then apply (2.21). Note that $\mathbf{M}$ is a nonsingular, well-conditioned matrix, and it can be safely generated with 64-bit precision, although using vpa to generate $\hat{\mathbf{D}}_t^\alpha \cdot \mathbf{M}$ and $\hat{\mathbf{E}}_t^\alpha \cdot \mathbf{M}$ and then converting the result into 64-bit precision yields higher accuracy.

3.1. Computation of the coefficients $\{\hat{f}_k\}$. The computation of $\{\hat{f}_k\}$ is a straightforward application of (2.3), (2.4), and (2.5). Therefore, given the values $\{f(x_j)\}$, for $0 \le j \le N$, we compute $\{g_j\}$, for $0 \le j \le 2N-1$, by means of (2.3), $\{\hat{g}_k\}$, for $0 \le k \le 2N-1$, by means of (2.5), and finally, $\{\hat{f}_k\}$, for $0 \le k \le N$, by means of (2.4):

$$\hat{f}_k = \begin{cases} \hat{g}_0, & k = 0, \\ 2\hat{g}_k, & 1 \le k \le N-1, \\ \hat{g}_N, & k = N. \end{cases}$$

Note that the DFT in (2.5) is performed by means of a fast Fourier transform (FFT) [15], which reduces the computational cost from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \ln N)$. However, Matlab's implementation of the FFT multiplies the result by the number of input values, i.e., it actually calculates $\{2N\hat{g}_k\}$. Therefore, we have to divide the FFT result by $2N$:

$$\hat{g}_k = \frac{1}{2N}\,\mathrm{FFT}(g_0, g_1, \ldots, g_{2N-1}) = \frac{1}{2N}\left[2N \sum_{j=0}^{2N-1} g_j e^{-2\mathrm{i}jk\pi/(2N)}\right].$$

Bearing in mind the previous arguments, the Matlab code is as follows:

ETNA
Kent State University and
Johann Radon Institute (RICAM)


```
g=[f;f(N:-1:2)];
hatg=fft(g)/N;
hatf=hatg(1:N+1);
hatf([1 N+1])=hatf([1 N+1])/2;
hatf(abs(hatf)<eps)=0;
```

Observe that in the last line we have applied the so-called Krasny's filter [18], i.e., we have set to zero those f_k for which $|f_k| < \varepsilon$, where $\varepsilon = 2^{-52}$. Applying this filter is very common in FFT-based pseudospectral methods and prevents the infinitesimally small rounding errors that appear when applying the FFT from being amplified. Therefore, in general, we apply this filter systematically whenever it is possible.

3.2. Computation of $\hat{\mathbf{D}}_t^\alpha$ and $\hat{\mathbf{E}}_t^\alpha$. In order to efficiently generate $\hat{\mathbf{D}}_t^\alpha$ and $\hat{\mathbf{E}}_t^\alpha$, we observe that, thanks to the structure of (2.18) and (2.19), they can both be expressed as a product of three matrices. More precisely, in the case of $\hat{\mathbf{D}}_t^\alpha$,

$$\hat{\mathbf{D}}_t^\alpha = \mathbf{D}_1 \cdot \operatorname{diag}(\mathbf{d}_2) \cdot \mathbf{C}(\lceil\alpha\rceil + 1 : N+1, 1 : N+1), \tag{3.1}$$

where $\mathbf{C}(\lceil\alpha\rceil + 1 : N+1, 1 : N+1) \in \mathbb{C}^{(N+1-\lceil\alpha\rceil)\times(N+1)}$ is the matrix $\mathbf{C}$ without its first $\lceil\alpha\rceil$ columns,

$$\mathbf{D}_1 = \begin{bmatrix} t_0^{\lceil\alpha\rceil-\alpha} & t_0^{\lceil\alpha\rceil+1-\alpha} & \cdots & t_0^{N-\alpha} \\ t_1^{\lceil\alpha\rceil-\alpha} & t_1^{\lceil\alpha\rceil+1-\alpha} & \cdots & t_1^{N-\alpha} \\ \vdots & \vdots & \ddots & \vdots \\ t_N^{\lceil\alpha\rceil-\alpha} & t_N^{\lceil\alpha\rceil+1-\alpha} & \cdots & t_N^{N-\alpha} \end{bmatrix} \in \mathbb{C}^{(N+1)\times(N+1-\lceil\alpha\rceil)},$$

and $\operatorname{diag}(\mathbf{d}_2) \in \mathbb{C}^{(N+1-\lceil\alpha\rceil)\times(N+1-\lceil\alpha\rceil)}$ is the diagonal matrix whose diagonal entries are precisely the entries of the vector $\mathbf{d}_2 \in \mathbb{C}^{N+1-\lceil\alpha\rceil}$:

$$\mathbf{d}_2 = \left(\frac{\Gamma(\lceil\alpha\rceil+1)}{T^{\lceil\alpha\rceil}\Gamma(\lceil\alpha\rceil+1-\alpha)}, \frac{\Gamma(\lceil\alpha\rceil+2)}{T^{\lceil\alpha\rceil+1}\Gamma(\lceil\alpha\rceil+2-\alpha)}, \ldots, \frac{\Gamma(N+1)}{T^N\Gamma(N+1-\alpha)} \right). \tag{3.2}$$

Likewise, in the case of $\hat{\mathbf{E}}_t^\alpha$,

$$\hat{\mathbf{E}}_t^\alpha = \mathbf{E}_1 \cdot \operatorname{diag}(\mathbf{e}_2) \cdot \mathbf{C}, \tag{3.3}$$

where

$$\mathbf{E}_1 = \begin{bmatrix} t_0^\alpha & t_0^{1+\alpha} & \cdots & t_0^{N+\alpha} \\ t_1^\alpha & t_1^{1+\alpha} & \cdots & t_1^{N+\alpha} \\ \vdots & \vdots & \ddots & \vdots \\ t_N^\alpha & t_N^{1+\alpha} & \cdots & t_N^{N+\alpha} \end{bmatrix} \in \mathbb{C}^{(N+1)\times(N+1)},$$

and $\operatorname{diag}(\mathbf{e}_2) \in \mathbb{C}^{(N+1)\times(N+1)}$ is the diagonal matrix whose diagonal entries are precisely the entries of the vector $\mathbf{e}_2$:

$$\mathbf{e}_2 = \left(\frac{\Gamma(1)}{T^0\Gamma(1+\alpha)}, \frac{\Gamma(2)}{T^1\Gamma(2+\alpha)}, \ldots, \frac{\Gamma(N+1)}{T^N\Gamma(N+1+\alpha)} \right) \in \mathbb{C}^{N+1}. \tag{3.4}$$

Regarding the actual implementation in Matlab, we observe that the matrix $\mathbf{C}$ generated by (2.12) is characterized by the fact that adjacent nonzero entries have opposite signs, and

ETNA
Kent State University and
Johann Radon Institute (RICAM)

the absolute value of those entries can take extremely large values as N grows. For instance, when $N = 100$, the exact values of $c_{71,101}$ and $c_{72,101}$ are

$$\begin{aligned} c_{71,101} &= 16663353313943991298474502833578948335 \\ &\quad 63446572325067822868496043008453509120, \\ c_{72,101} &= -16977944641117641718553583947907828690 \\ &\quad 78728901979391848292743945528541184000, \end{aligned}$$

i.e., they both are of the order of $\mathcal{O}(10^{75})$ and have opposite signs. Therefore, working with standard 64-bit floating-point precision (IEEE 754) causes most digits to be lost:

$$\begin{aligned} c_{71,101} &= 1.666335331394399 \times 10^{75}, \\ c_{72,101} &= -1.697794464111764 \times 10^{75}, \end{aligned}$$

which has disastrous effects in the computation of (3.1) and (3.3). In order to avoid this problem, we have found that the use of vpa, which allows working with an arbitrary number of digits and computing (3.1) and (3.3) accurately, is effective. In Matlab, we have to fix the number of digits (by default, 32) by means of the function `digits`; then, the function `vpa`, applied to a number, returns the number with the required digits. For instance, if we type

```
digits(1000)
vpa(pi)
```

then the first 1000 digits of π are shown. Note that we could have also typed `vpa(pi,1000)`, but if we have not set `digits(1000)` previously, then, e.g., `5*vpa(pi,1000)` returns only 32 digits by default, so it is necessary to invoke `digits` before using `vpa` to maintain the desired accuracy throughout all the operations. On the other hand, sometimes it will be necessary to convert a given value into a symbolic variable before applying `vpa`. For instance, if we want 1000 digits of π^2 and type `vpa(pi^2,1000)`, then we are not calculating π^2 but rather its 64-bit floating-point representation, so the result is incorrect. Therefore, we have to type commands like `vpa(sym(pi)^2,1000)` or, even better, `vpa(str2sym('pi^2'),1000)`, which turns the array of characters `'pi^2'` into a symbolic variable. In general, the use of `str2sym` is always a safe option. A final observation with respect to `vpa` is that if we want to apply it to an all-zero matrix, then this is equivalent to applying `sym`. Therefore, `(vpa([0 0]) + 5).^1000` and `(sym([0 0]) + 5).^1000` produce the same result, whereas `(vpa(0) + 5)^100` and `(sym(0) + 5)^1000` do not because `(sym(0) + 5)^1000` returns the exact result, whereas `(vpa(0) + 5)^1000` truncates the result to the number of digits set by the function `digits`.

Bearing in mind the previous arguments, we first generate $\mathbf{C}$, which appears in both (3.1) and (3.3), by implementing (2.12) using `vpa`. The resulting Matlab code is the following:

```
C=sym(zeros(N+1));
C(1,1)=1;
C(1:2,2)=[-1;2];
for j=3:N+1
  C(1:j,j)=[(-1)^(j-1);...
    4*C(1:j-1,j-1)-2*C(2:j,j-1)-C(2:j,j-2)];
end
```

Note that the only difference with respect to working with 64-bit floating-point precision is the use of `sym` in the first line. Indeed, if we type `C=zeros(N+1);`, then all the entries are

ETNA
Kent State University and
Johann Radon Institute (RICAM)

generated with 64 bits with the resulting loss of accuracy. On the other hand, we have used `sym` in this piece of code instead of `vpa` because, as we have said above, there is no difference in the result. If we want to ensure that `C` is really stored using `vpa` and with a specific number of digits, then we must write, e.g., `C(1,1)=vpa(1);` in the second line, but there is no significant variation in the elapsed time, so we have computed `C` with `sym`. However, even if it is possible to generate (3.1) and (3.3) completely by means of `sym` without using `vpa` at all, the computational cost is much higher, so we have only used `sym` in the generation of `C`.

The generation of $\mathbf{D}_1$ and $\mathbf{E}_1$ poses no problem. More precisely, after computing the time instants $\{t_j\}$ with `vpa`,

```
T_vpa=vpa(T);
t=T_vpa*(1+cos(vpa(pi)*(0:N)/N)')/2;
```

where `T_vpa` contains the vpa version of the final time T, then it is enough to type

```
a_vpa=vpa(a);
ceila=double(ceil(a_vpa));
D1=t.^((ceila:N)-a_vpa);
E1=t.^((0:N)+a_vpa);
```

where `a_vpa` contains the vpa version of α and `ceila` corresponds to $\lceil\alpha\rceil$.

In order to compute $\mathbf{e}_2$, we observe that

$$\frac{\Gamma(\ell+1)}{\Gamma(\ell+1+\alpha)} = \frac{\ell}{\ell+\alpha}\cdot\frac{\ell-1}{\ell-1+\alpha}\cdot\ldots\cdot\frac{1}{1+\alpha}\cdot\frac{1}{\Gamma(1+\alpha)}, \qquad 0\le\ell\le N,$$

where we have used that $\Gamma(1+z)=z\Gamma(z)$, for all $z\in\mathbb{C}$. Therefore, $\mathbf{e}_2$ in (3.4) can be generated very efficiently as a cumulative product:

```
e2=cumprod([1/gamma(1+a_vpa),...
   ((1:N)./((1:N)+a_vpa))/T_vpa]).';
```

Observe that we store `e2` as a column vector because Matlab allows writing `e2.*C`, which is equivalent to but faster than `diag(e2)*C`, so $\hat{\mathbf{E}}_t^\alpha$ in (3.3) is created by typing

```
hatEta=double(E1*(e2.*C));
```

where we have converted the result from `vpa` to `double` so that `hatEta` is stored in 64-bit precision. Observe that even though we require $\alpha\notin\mathbb{N}\cup\{0\}$ in the definition of $\hat{\mathbf{D}}_t^\alpha$ and $\hat{\mathbf{E}}_t^\alpha$, it is possible to consider also $\alpha\in\mathbb{N}\cup\{0\}$ to create `hatEta` without modifying the codes, but this is not true in the case of $\hat{\mathbf{D}}_t^\alpha$. Indeed, when $\alpha\notin\mathbb{N}\cup\{0\}$, we generate `d2` in a way similar to `e2`. More precisely, bearing in mind that

$$\frac{\Gamma(\ell+1)}{\Gamma(\ell+1-\alpha)} = \frac{\ell}{\ell-\alpha}\cdot\frac{\ell-1}{\ell-1-\alpha}\cdots\frac{1}{1-\alpha}\cdot\frac{1}{\Gamma(1-\alpha)}, \qquad 0\le\ell\le N, \tag{3.5}$$

we write

```
d2aux=cumprod([1/gamma(1-a_vpa),...
   ((1:N)./((1:N)-a_vpa))/T_vpa]).';
d2=d2aux(ceila+1:N+1);
```

i.e., we have ignored the first $\lceil\alpha\rceil$ entries of `d2aux`. However, when $\alpha\in\mathbb{N}$, (3.5) cannot be used because $\Gamma(1-\alpha)$ is not defined. Therefore, in that case, we write

$$\frac{\Gamma(\ell+1)}{\Gamma(\ell+1-\alpha)} = \frac{\ell}{\ell-\alpha}\cdot\frac{\ell-1}{\ell-1-\alpha}\cdot\ldots\cdot\frac{2+\alpha}{2}\cdot\frac{1+\alpha}{1}\cdot\Gamma(1+\alpha), \qquad \alpha\le\ell\le N,$$

and hence, to generate $\mathbf{d}_2$ in (3.2), we write

ETNA
Kent State University and
Johann Radon Institute (RICAM)


```
d2=cumprod([prod(1:a_vpa)/T_vpa^a_vpa,...
  ((a_vpa+1):N)./(1:(N-a_vpa))/T_vpa]).';
```

where `prod(1:(1+a_vpa))` corresponds to $(1+\alpha)\Gamma(1+\alpha) = \Gamma(2+\alpha) = (1+\alpha)!$. Observe that, when $\alpha = 0$, `d2` can be generated in both ways. Finally, in all cases, $\hat{\mathbf{D}}_t^\alpha$ in (3.1) is created by typing

```
hatDta=double(D1*(d2.*C(ceila+1:N+1,:)));
```

where we have removed the first $\lceil \alpha \rceil$ columns of $\mathbf{C}$ and have stored `hatDta` in 64-bit precision.

3.3. Computation of $\mathbf{D}_t^\alpha$ and $\mathbf{E}_t^\alpha$. In order to obtain $\mathbf{D}_t^\alpha$ and $\mathbf{E}_t^\alpha$, we only need to compute $\mathbf{M}$ and apply (2.22). We remark that it is numerically safe to construct $\mathbf{M}$ using 64-bit precision and then multiply, respectively, the 64-bit versions of $\hat{\mathbf{D}}_t^\alpha$ and $\hat{\mathbf{E}}_t^\alpha$ by it. However, using vpa gives slightly more accurate results. The code is an immediate application of (2.8):

```
M=2*cos(vpa(pi)*(0:N)'*(0:N)/N);
M(:,[1 N+1])=M(:,[1 N+1])/2;
M([1 N+1],:)=M([1 N+1],:)/2;
M=M/N;
```

and the non-vpa version would follow by just replacing `vpa(pi)` by `pi`. Then we multiply the vpa versions of `hatDta` and `hatEta` by it and afterwards apply `double` to all the matrices, i.e.,

```
hatEta=E1*(e2.*C);
Eta=double(hatEta*M);
hatEta=double(hatEta);
```

and

```
hatDta=(D1*(d2.*C(ceila+1:N+1,:)));
Dta=double(hatDta*M);
hatDta=double(hatDta);
```

3.4. Matlab codes to generate $\hat{\mathbf{D}}_t^\alpha$, $\hat{\mathbf{E}}_t^\alpha$, $\mathbf{D}_t^\alpha$, and $\mathbf{E}_t^\alpha$. In Listing 1 we present the code of the function `CaputoMatrix.m`, which generates $\hat{\mathbf{D}}_t^\alpha$ and $\mathbf{D}_t^\alpha$, and in Listing 2 the code of the function `RiemannLiouvilleMatrix.m`, which generates $\hat{\mathbf{E}}_t^\alpha$ and $\mathbf{E}_t^\alpha$. Moreover, both functions also return the 64-bit discretization of $t \in [0, T]$. Even if, for the sake of clarity, we present them separately, it is straightforward to combine them into a single function, which would imply some reduction of the execution time because $\mathbf{C}$ and $\mathbf{M}$ would then be generated only once instead of twice. In fact, if we increase the size of $\mathbf{C}$, then new columns and rows are added, but the previously generated ones do not change. Therefore, it is perfectly possible to pregenerate a matrix $\mathbf{C}$ having a larger size than required, store it, and then introduce it as a parameter, a global variable, etc. In that case, we have to type `hatDta=D1*(d2.*C(ceila+1:N+1,1:N+1));` and `hatEta=E1*(e2.*C(1:N+1,1:N+1));`, respectively.

Note also that the variables `a_vpa` and `T_vpa` store, respectively, a vpa version of the parameters `a` (corresponding to α) and `T` (corresponding to the final time T). Therefore, when invoking `CaputoMatrix.m` and `RiemannLiouvilleMatrix.m`, it is possible to introduce `a` and `T` as 64-bit precision numbers, as vpa variables, or even as symbolic variables; for instance, if $\alpha = \exp(-1)$ and $T = \pi^2$, then we must introduce `a` and `T` as `str2sym('exp(-1)')` and `str2sym('pi^2')`, respectively. On the other hand, in practice, it is not necessary to consider a vpa version of `N` (which corresponds to N) because it represents a natural number that is correctly handled by Matlab.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

LISTING 1
Matlab function `CaputoMatrix.m` *that generates* $\hat{\mathbf{D}}_t^\alpha$ *and* $\mathbf{D}_t^\alpha$.

```
% Create the Caputo differentiation matrices
% $\hat{\mathbf D}_t^\alpha$ and $\mathbf D_t^\alpha$.
% N, a, T and dig denote respectively
% $N$, $\alpha$, $T$ and the number of digits for vpa.
function [hatDta,Dta,t]=CaputoMatrix(N,a,T,dig)
digits(dig); % Set the number of digits for vpa
a_vpa=vpa(a); % vpa version of $\alpha$
ceila=double(ceil(a_vpa)); % $\lceil\alpha\rceil$
T_vpa=vpa(T); % vpa version of $T$
t=T_vpa*(1+cos(vpa(pi)*(0:N)/N)')/2; % $t\in[0,T]$
C=sym(zeros(N+1)); % $\mathbf C$
C(1,1)=1;
C(1:2,2)=[-1;2];
for j=3:N+1
    C(1:j,j)=[(-1)^(j-1);4*C(1:j-1,j-1)-2*C(2:j,j-1)-C(2:j,j-2)];
end
D1=t.^((ceila:N)-a_vpa); % $\mathbf D_1$
% $\mathbf d_2$
if double(a_vpa==ceil(a_vpa)) % Test whether $\alpha\in\mathbb Z$
    d2=cumprod([prod(1:a_vpa)/T_vpa^a_vpa,...
        ((a_vpa+1):N)./(1:(N-a_vpa))/T_vpa]).';
else
    d2aux=cumprod([1/gamma(1-a_vpa),((1:N)./((1:N)-a_vpa))/T_vpa]).';
    d2=d2aux(ceila+1:N+1);
end
hatDta=(D1*(d2.*C(ceila+1:N+1,:))); % $\hat{\mathbf D}_t^\alpha$
M=2*cos(vpa(pi)*(0:N)'*(0:N)/N);
M(:,[1 N+1])=M(:,[1 N+1])/2;
M([1 N+1],:)=M([1 N+1],:)/2;
M=M/N;
Dta=double(hatDta*M); % $\mathbf D_t^\alpha$
hatDta=double(hatDta);
t=double(t);
```

LISTING 2
Matlab function `RiemannLiouvilleMatrix.m` *that generates* $\hat{\mathbf{E}}_t^\alpha$ *and* $\mathbf{E}_t^\alpha$.

```
% Create the Riemann-Liouville integration matrices
% $\hat{\mathbf E}_t^\alpha$ and $\mathbf E_t^\alpha$.
% N, a, T and dig denote respectively
% $N$, $\alpha$, $T$ and the number of digits for vpa.
function [hatEta,Eta,t]=RiemannLiouvilleMatrix(N,a,T,dig)
digits(dig); % Set the number of digits for vpa
a_vpa=vpa(a); % vpa version of $\alpha$
T_vpa=vpa(T); % vpa version of $T$
t=T_vpa*(1+cos(vpa(pi)*(0:N)/N)')/2; % $t\in[0,T]$
C=sym(zeros(N+1)); % $\mathbf C$
C(1,1)=1;
C(1:2,2)=[-1;2];
for j=3:N+1
    C(1:j,j)=[(-1)^(j-1);4*C(1:j-1,j-1)-2*C(2:j,j-1)-C(2:j,j-2)];
end
E1=t.^((0:N)+a_vpa); % $\mathbf E_2$
e2=cumprod([1/gamma(1+a_vpa),...
    ((1:N)./((1:N)+a_vpa))/T_vpa]).'; % $\mathbf e_2$
hatEta=E1*(e2.*C); % $\hat{\mathbf E}_t^\alpha$
M=2*cos(vpa(pi)*(0:N)'*(0:N)/N);
M(:,[1 N+1])=M(:,[1 N+1])/2;
M([1 N+1],:)=M([1 N+1],:)/2;
M=M/N;
Eta=double(hatEta*M); % $\mathbf E_t^\alpha$
```

ETNA
Kent State University and
Johann Radon Institute (RICAM)


```
hatEta=double(hatEta);
t=double(t);
```

3.5. Estimation of the number of digits. In general, except for very small values of N, it is not possible to generate $\hat{\mathbf{D}}_t^\alpha$, $\mathbf{D}_t^\alpha$, $\hat{\mathbf{E}}_t^\alpha$, and $\mathbf{E}_t^\alpha$ correctly; a fact that makes the use of vpa compulsory. It is easy to verify this experimentally by just removing the variable `dig` and all the appearances of `vpa` in `CaputoMatrix.m` (see Listing 1) and `RiemannLiouvilleMatrix.m` (see Listing 2). Then, if we take $N = 40$, $\alpha = 0.37$, and $T = 1.2$, the so generated matrices $\hat{\mathbf{D}}_t^\alpha$, $\mathbf{D}_t^\alpha$, $\hat{\mathbf{E}}_t^\alpha$, and $\mathbf{E}_t^\alpha$ are nonsensical and contain entries whose moduli are as large as

$$\|\hat{\mathbf{D}}_t^\alpha\|_{\max} = 2.4053 \times 10^{14}, \qquad \|\mathbf{D}_t^\alpha\|_{\max} = 8.2056 \times 10^{12},$$
$$\|\hat{\mathbf{E}}_t^\alpha\|_{\max} = 2.6064 \times 10^{13}, \quad \text{and} \quad \|\mathbf{E}_t^\alpha\|_{\max} = 8.8991 \times 10^{11},$$

respectively, where $\|\cdot\|_{\max}$ denotes the maximum norm (recall that given a matrix $\mathbf{A} = [a_{ij}]$, $\|\mathbf{A}\|_{\max} = \max_{i,j} |a_{ij}|$ returns the maximum of the moduli of the entries of $\mathbf{A}$).

This behavior is also observed if we work with vpa and a too small number of digits `dig`. For instance, if $N = 100$, $\alpha = 0.37$, and $T = 1.2$, then,

$$\|\hat{\mathbf{D}}_t^\alpha(\texttt{dig} = 30)\|_{\max} = 1.7477 \times 10^{38}, \qquad \|\mathbf{D}_t^\alpha(\texttt{dig} = 30)\|_{\max} = 3.8319 \times 10^{36},$$
$$\|\hat{\mathbf{E}}_t^\alpha(\texttt{dig} = 30)\|_{\max} = 3.3523 \times 10^{37}, \quad \text{and} \quad \|\mathbf{E}_t^\alpha(\texttt{dig} = 30)\|_{\max} = 3.7454 \times 10^{35},$$

which makes it evident that 30 digits are not enough when $N = 100$. However, if we increase the number of digits to 100, then

$$\|\hat{\mathbf{D}}_t^\alpha(\texttt{dig} = 100)\|_{\max} = 46.0508, \qquad \|\mathbf{D}_t^\alpha(\texttt{dig} = 100)\|_{\max} = 26.2840,$$
$$\|\hat{\mathbf{E}}_t^\alpha(\texttt{dig} = 100)\|_{\max} = 1.2029, \quad \text{and} \quad \|\mathbf{E}_t^\alpha(\texttt{dig} = 100)\|_{\max} = 0.19984.$$

Moreover, once the necessary number of digits `dig` has been reached, there is no difference in the final 64-bit precision version of the matrices if we further increase `dig`. For instance, the results using 1000 digits are identical to those with 100 digits. Therefore, given an adequate value of `dig`, we can simply set $\hat{\mathbf{D}}_t^\alpha = \hat{\mathbf{D}}_t^\alpha(\texttt{dig})$, $\mathbf{D}_t^\alpha = \mathbf{D}_t^\alpha(\texttt{dig})$, $\hat{\mathbf{E}}_t^\alpha = \hat{\mathbf{E}}_t^\alpha(\texttt{dig})$, and $\mathbf{E}_t^\alpha = \mathbf{E}_t^\alpha(\texttt{dig})$.

In Figure 3.1, in order to clarify the effect of `dig` on the generation of the matrices, we display on a semilogarithmic scale the values

$$\|\hat{\mathbf{D}}_t^\alpha(\texttt{dig}) - \hat{\mathbf{D}}_t^\alpha(\texttt{dig}+1)\|_{\max}, \qquad \|\mathbf{D}_t^\alpha(\texttt{dig}) - \mathbf{D}_t^\alpha(\texttt{dig}+1)\|_{\max},$$
$$\|\hat{\mathbf{E}}_t^\alpha(\texttt{dig}) - \hat{\mathbf{E}}_t^\alpha(\texttt{dig}+1)\|_{\max}, \quad \text{and} \quad \|\mathbf{E}_t^\alpha(\texttt{dig}) - \mathbf{E}_t^\alpha(\texttt{dig}+1)\|_{\max}$$

with respect to $\texttt{dig} = 2, 3, 4, \ldots$, until these quantities are strictly equal to zero—although it is enough to set them smaller than a given infinitesimal number ε, e.g., $\varepsilon = 2^{-52}$, represented in Matlab by `eps`. As can be seen, the values decay exponentially with respect to `dig`.

To better understand how to choose `dig` in terms of N, we display on the left-hand side of Figure 3.2 the minimum values of `dig` for which

$$\|\hat{\mathbf{D}}_t^\alpha(\texttt{dig}) - \hat{\mathbf{D}}_t^\alpha(\texttt{dig}+1)\|_{\max} = 0 \ \text{(red)},$$
$$\|\mathbf{D}_t^\alpha(\texttt{dig}) - \mathbf{D}_t^\alpha(\texttt{dig}+1)\|_{\max} = 0 \ \text{(blue)},$$
$$\|\hat{\mathbf{E}}_t^\alpha(\texttt{dig}) - \hat{\mathbf{E}}_t^\alpha(\texttt{dig}+1)\|_{\max} = 0 \ \text{(green)}, \ \text{and}$$
$$\|\mathbf{E}_t^\alpha(\texttt{dig}) - \mathbf{E}_t^\alpha(\texttt{dig}+1)\|_{\max} = 0 \ \text{(cyan)}, \quad \text{for } 2 \le N \le 600;$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

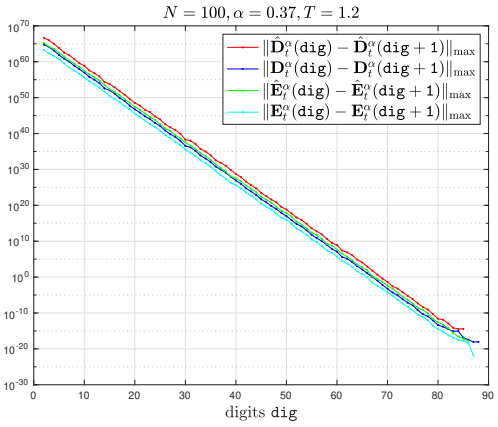

FIG. 3.1. $\|\hat{\mathbf{D}}_t^\alpha(\mathtt{dig}) - \hat{\mathbf{D}}_t^\alpha(\mathtt{dig}+1)\|_{\max}$ *(red),* $\|\mathbf{D}_t^\alpha(\mathtt{dig}) - \mathbf{D}_t^\alpha(\mathtt{dig}+1)\|_{\max}$ *(blue),* $\|\hat{\mathbf{E}}_t^\alpha(\mathtt{dig}) - \hat{\mathbf{E}}_t^\alpha(\mathtt{dig}+1)\|_{\max}$ *(green), and* $\|\mathbf{E}_t^\alpha(\mathtt{dig}) - \mathbf{E}_t^\alpha(\mathtt{dig}+1)\|_{\max}$ *(cyan), for* $\mathtt{dig} = 2, 3, 4, \ldots$ *In all cases, we have taken* $N = 100$*,* $\alpha = 0.37$ *and* $T = 1.2$*.*

these conditions are very exigent and, as mentioned above, can be relaxed by setting, e.g., $< \varepsilon = 2^{-52}$ instead of $= 0$. The numerical experiments reveal that the number of necessary digits grows approximately linearly with N, and a linear regression analysis yields

$$\begin{aligned} &\min \mathtt{dig} \approx 0.7663N + 8.449 \ \text{ for } \hat{\mathbf{D}}_t^\alpha, && \min \mathtt{dig} \approx 0.7704N + 11.57 \ \text{ for } \mathbf{D}_t^\alpha, \\ &\min \mathtt{dig} \approx 0.7671N + 9.59 \ \text{ for } \hat{\mathbf{E}}_t^\alpha, \quad \text{and} && \min \mathtt{dig} \approx 0.7684N + 10.81 \ \text{ for } \mathbf{E}_t^\alpha. \end{aligned}$$

For the sake of comparison, we also display on the left-hand side of Figure 3.2 the decimal logarithm of the maximum of the absolute values of the Nth column of $\mathbf{C}$ in terms of N, i.e., $\log_{10}(\max_{1\le i\le N} |c_{iN}|) \approx 0.7645N - 1.946$, i.e., the fifth regression lines are approximately parallel, which shows that the number of necessary digits is correlated with the number of necessary digits to store the corresponding column of $\mathbf{C}$ in an exact way.

On the other hand, we display on the right-hand side of Figure 3.2 the times needed to generate $\hat{\mathbf{D}}_t^\alpha(\mathtt{dig})$ (red), $\mathbf{D}_t^\alpha(\mathtt{dig})$ (blue), $\hat{\mathbf{E}}_t^\alpha(\mathtt{dig})$ (green), and $\mathbf{E}_t^\alpha(\mathtt{dig})$ (cyan) for the corresponding values of `dig` obtained on the left-hand side of Figure 3.2; note that in the case of $\hat{\mathbf{D}}_t^\alpha$ and $\hat{\mathbf{E}}_t^\alpha$, we have executed `CaputoMatrix.m` and `RiemannLiouvilleMatrix.m` after removing the code corresponding to the generation of $\mathbf{M}$ and of $\mathbf{D}_t^\alpha$ and $\mathbf{E}_t^\alpha$, which diminishes the execution time because the computation of $\mathbf{D}_t^\alpha$ and $\mathbf{E}_t^\alpha$ requires $\hat{\mathbf{D}}_t^\alpha$ and $\hat{\mathbf{E}}_t^\alpha$, respectively, but not the other way around. In all cases, the number of seconds seems to grow as $\mathcal{O}(N^3)$, and a least-square regression analysis yields

$$\begin{aligned} &\text{time} \approx 1.568 \times 10^{-6}N^3 - 0.000454N^2 + 0.09306N - 2.875 && \text{for } \hat{\mathbf{D}}_t^\alpha, \\ &\text{time} \approx 3.59 \times 10^{-6}N^3 - 0.001004N^2 + 0.1744N - 5.329 && \text{for } \mathbf{D}_t^\alpha, \\ &\text{time} \approx 1.56 \times 10^{-6}N^3 - 0.0004728N^2 + 0.0924N - 2.858 && \text{for } \hat{\mathbf{E}}_t^\alpha, \\ &\text{time} \approx 3.585 \times 10^{-6}N^3 - 0.001041N^2 + 0.1789N - 5.557 && \text{for } \mathbf{E}_t^\alpha. \end{aligned}$$

Finally, let us mention that the equivalent results to those in Figure 3.2 are very similar for other values of α and T, but, in case of doubt, it is enough to generate the required matrices with `dig` and `dig` $+ 1$ and test whether they are identical.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

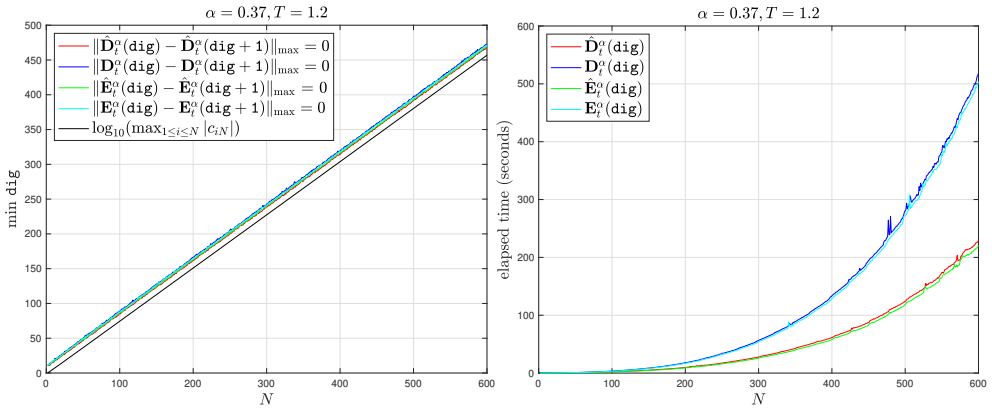

FIG. 3.2. *Left: minimum values of* dig *for which both* $\|\hat{\mathbf{D}}_t^\alpha(\mathtt{dig}) - \hat{\mathbf{D}}_t^\alpha(\mathtt{dig}+1)\|_{\max} = 0$ *and* $\|\mathbf{D}_t^\alpha(\mathtt{dig}) - \mathbf{D}_t^\alpha(\mathtt{dig}+1)\|_{\max} = 0$ *(red), and the minimum values of* dig *for which both* $\|\hat{\mathbf{E}}_t^\alpha(\mathtt{dig}) - \hat{\mathbf{E}}_t^\alpha(\mathtt{dig}+1)\|_{\max} = 0$ *and* $\|\mathbf{E}_t^\alpha(\mathtt{dig}) - \mathbf{E}_t^\alpha(\mathtt{dig}+1)\|_{\max} = 0$ *(blue), together with* $\max_{1\le i\le N}|c_{iN}|$*, for* $2 \le N \le 600$*. Right: elapsed time of the experiments on the left-hand side. In all cases, we have taken* $\alpha = 0.37$ *and* $T = 1.2$*.*

4. Numerical tests. In order to perform the numerical tests, we use the functions `CaputoMatrix.m` and `RiemannLiouvilleMatrix.m` presented in Section 3 to approximate $D_t^\alpha e^{imt}$ and $I_t^\alpha e^{imt}$, respectively, whose exact expressions can be obtained, e.g., by typing in Mathematica [28] the lines

```
f[t_] = Exp[I  m  t];
Integrate[D[f[tau], {tau, n}]/(t - tau)^(1 - n + a),
{tau, 0, t}]/Gamma[n - a]
Integrate[f[tau]/(t - tau)^(1 - a), {tau, 0, t}]/Gamma[a]
```

which, after some minor simplification, yields

$$D_t^\alpha e^{imt} = (im)^\alpha e^{imt}\left(1 - \frac{\Gamma(\lceil\alpha\rceil - \alpha, imt)}{\Gamma(\lceil\alpha\rceil - \alpha)}\right), \quad \alpha \neq \lceil\alpha\rceil, \tag{4.1}$$

$$I_t^\alpha e^{imt} = (im)^{-\alpha} e^{imt}\left(1 - \frac{\Gamma(\alpha, imt)}{\Gamma(\alpha)}\right), \qquad \alpha > 0, \tag{4.2}$$

where $\Gamma(\cdot,\cdot)$, whose corresponding Matlab command is `igamma`, denotes the upper incomplete gamma function:

$$\Gamma(s,x) = \int_x^\infty t^{s-1}e^{-t}dt,$$

and hence, $\Gamma(s) = \Gamma(s,0)$.

4.1. A full numerical test in Matlab. In Listing 3 we provide a Matlab program that numerically approximates $D_t^\alpha e^{imt}$ and $I_t^\alpha e^{imt}$. In the case of $D_t^\alpha e^{imt}$ we compare the results with (4.1) when $\alpha \neq \lceil\alpha\rceil$ and with $(d^\alpha/dt^\alpha)e^{imt} = (im)^\alpha e^{imt}$ when $\alpha = \lceil\alpha\rceil$; in particular, $\alpha = 0$ returns e^{imt} itself. On the other hand, in the case of $I_t^\alpha e^{imt}$ we compare the results with (4.2) when $\alpha > 0$ and with e^{imt} itself when $\alpha = 0$. For the sake of comparison, we have also generated $\mathbf{M}$ as in (2.8) using 64-bit precision and compared $\hat{\mathbf{D}}_t^\alpha \cdot \mathbf{M}$ and $\hat{\mathbf{E}}_t^\alpha \cdot \mathbf{M}$ to $\mathbf{D}_t^\alpha$ and $\mathbf{E}_t^\alpha$, respectively.

The experiment for $\alpha = 1.3 \in (1,2)$, $T = 1.2$, $m = 2$, $N = 100$, and $\mathtt{dig} = 100$ digits takes 8.16 seconds to run, yielding the following errors in the discrete ℓ^∞-norm:

ETNA
Kent State University and
Johann Radon Institute (RICAM)

- $\max\limits_{0\le j\le N} \left|[\mathbf{M}\cdot\mathbf{f}]_j - [\hat{\mathbf{f}}]_j\right| = 3.5376\times 10^{-15}$,
- $\max\limits_{0\le j\le N} \left|[\hat{\mathbf{D}}_t^\alpha\cdot\hat{\mathbf{f}}]_j - D_t^\alpha f(t_j)\right| = \begin{cases} 2.8893\times 10^{-11} & \text{(without Krasny's filter)},\\ 6.8315\times 10^{-14} & \text{(with Krasny's filter)},\end{cases}$
- $\max\limits_{0\le j\le N} \left|[\mathbf{D}_t^\alpha\cdot\mathbf{f}]_j - D_t^\alpha f(t_j)\right| = 3.7006\times 10^{-11}$,
- $\max\limits_{0\le j\le N} \left|[\hat{\mathbf{D}}_t^\alpha\cdot\mathbf{M}\cdot\mathbf{f}]_j - D_t^\alpha f(t_j)\right| = 1.0433\times 10^{-9}$,
- $\max\limits_{0\le j\le N} \left|[\hat{\mathbf{E}}_t^\alpha\cdot\hat{\mathbf{f}}]_j - I_t^\alpha f(t_j)\right| = 3.5108\times 10^{-16}$ (error with and without Krasny's filter),
- $\max\limits_{0\le j\le N} \left|[\mathbf{E}_t^\alpha\cdot\mathbf{f}]_j - I_t^\alpha f(t_j)\right| = 4.5776\times 10^{-16}$,
- $\max\limits_{0\le j\le N} \left|[\hat{\mathbf{E}}_t^\alpha\cdot\mathbf{M}\cdot\mathbf{f}]_j - I_t^\alpha f(t_j)\right| = 4.7429\times 10^{-16}$.

As can be seen, the errors in the approximation of $I_t^\alpha f(t)$ are always of the order of 10^{-16}, and Krasny's filter is unnecessary. However, when approximating $D_t^\alpha f(t)$, $\hat{\mathbf{D}}_t^\alpha\cdot\hat{\mathbf{f}}$ gives a better approximation than $\mathbf{D}_t^\alpha\cdot\mathbf{f}$ because of Krasny's filter. On the other hand, even if $\mathbf{M}\cdot\mathbf{f}$ calculates $\hat{\mathbf{f}}$ with high accuracy, the worst results are obtained with $\hat{\mathbf{D}}_t^\alpha\cdot\mathbf{M}\cdot\mathbf{f}$. Finally, let us mention that the choice of `dig` has a limited impact on the time of execution. For instance, if we take `dig` four times as large, i.e., `dig` $= 400$, then the elapsed time is 9.47 seconds.

LISTING 3
Matlab program `testmatrices.m` *that numerically approximates* $D_t^\alpha e^{imt}$ *and* $I_t^\alpha e^{imt}$.

```
clear
tic
format short
a=0.37; % $\alpha$
N=100; % $N$
dig=100; % Number of digits
T=1.2; % $T$
% $\hat{\mathbf D}^\alpha_t$, $\mathbf D^\alpha_t$ and $t$
[hatDta,Dta,t]=CaputoMatrix(N,a,T,dig);
% $\hat{\mathbf E}^\alpha_t$ and $\mathbf E^\alpha_t$
[hatEta,Eta]=RiemannLiouvilleMatrix(N,a,T,dig);
m=2;
f=exp(1i*m*t); % $f(t)=e^{imt}$
% Exact values of $D^\alpha_t f(t_j)$
if ceil(a)==a % if $\alpha$ is an integer
    Dtaf=(1i*m)^a*exp(1i*m*t);
else
    Dtaf=(1i*m)^a*exp(1i*m*t).*(1-igamma(ceil(a)-a,1i*m*t)/gamma(ceil(a)-a));
end
% Exact values of $I^\alpha_t f(t_j)$
if a == 0 % if $\alpha=0$
    Itaf=exp(1i*m*t);
else
    Itaf=(1i*m)^(-a)*exp(1i*m*t).*(1-igamma(a,1i*m*t)/gamma(a));
end
g=[f;f(N:-1:2)]; % $g_j$
hatg=fft(g)/N; % $\hat g_k$
hatf=hatg(1:N+1); % $\hat f_k$
hatf([1 N+1])=hatf([1 N+1])/2;
M=2*cos(pi*(0:N)'*(0:N)/N); % Generate $M$ using 64-bit precision
M(:,[1 N+1])=M(:,[1 N+1])/2;
M([1 N+1],:)=M([1 N+1],:)/2;
M=M/N;
% Errors in $L^\infty$ norm
norm(hatf-M*f,inf) % error in the approximation of $\hat f$ by $M$
norm(hatDta*hatf-Dtaf,inf) % error without Krasny's filter
```

ETNA
Kent State University and
Johann Radon Institute (RICAM)


```
norm(hatEta*hatf-Itaf,inf) % error without Krasny's filter
hatf(abs(hatf)<eps)=0; % Krasny's filter
norm(hatDta*hatf-Dtaf,inf) % error with Krasny's filter
norm(hatEta*hatf-Itaf,inf) % error with Krasny's filter
norm(Dta*f-Dtaf,inf)
norm(hatDta*M*f-Dtaf,inf)
norm(Eta*f-Itaf,inf)
norm(hatEta*M*f-Itaf,inf)
toc % Elapsed time
```

4.2. Numerical approximation of $D_t^\alpha e^{imt}$ and $I_t^\alpha e^{imt}$ for large natural values of m and different values of α and N. One important application of the matrices that we have constructed is the numerical approximation of the Caputo derivative and the Riemann–Liouville integral of highly oscillatory integrals. In this regard, we consider $D_t^\alpha e^{imt}$ and $I_t^\alpha e^{imt}$, taking $m = 110$, for which a large number of nodes N is required. The numerical experiments reveal that, in general, the approximation of $I_t^\alpha e^{imt}$ poses no problem, and spectrally accurate results are always obtained provided that N is large enough. On the other hand, due to the term $(\mathrm{i}m)^\alpha$ in (4.1), $\max_{t\in[0,T]} |D_t^\alpha e^{imt}|$ grows rapidly with α; therefore, we use the maximum relative error, which enables comparing the results for different values of α in a more consistent way. More precisely, given $f(t) = e^{imt}$, we define

$$\mathrm{err}_{\hat{\mathbf{D}}_t^\alpha} = \begin{cases} \max\limits_{0\le j\le N-1} \left| \dfrac{[\hat{\mathbf{D}}_t^\alpha \cdot \hat{\mathbf{f}}]_j - D_t^\alpha f(t_j)}{D_t^\alpha f(t_j)} \right|, & \alpha \neq \lceil \alpha \rceil, \\ \max\limits_{0\le j\le N} \left| \dfrac{[\hat{\mathbf{D}}_t^\alpha \cdot \hat{\mathbf{f}}]_j - (\mathrm{i}m)^\alpha e^{imt}}{(\mathrm{i}m)^\alpha e^{imt}} \right|, & \alpha = \lceil \alpha \rceil, \end{cases} \tag{4.3}$$

and

$$\mathrm{err}_{\mathbf{D}_t^\alpha} = \begin{cases} \max\limits_{0\le j\le N-1} \left| \dfrac{[\mathbf{D}_t^\alpha \cdot \hat{\mathbf{f}}]_j - D_t^\alpha f(t_j)}{D_t^\alpha f(t_j)} \right|, & \alpha \neq \lceil \alpha \rceil, \\ \max\limits_{0\le j\le N} \left| \dfrac{[\mathbf{D}_t^\alpha \cdot \hat{\mathbf{f}}]_j - (\mathrm{i}m)^\alpha e^{imt}}{(\mathrm{i}m)^\alpha e^{imt}} \right|, & \alpha = \lceil \alpha \rceil, \end{cases} \tag{4.4}$$

where $D_t^\alpha f(t_j)$ is given by (4.1). Note that, when $\alpha \neq \lceil \alpha \rceil$, we have omitted the value $j = N$ corresponding to $t_N = 0$ because $D_t^\alpha f(0) = 0$ in that case.

Since, when $\alpha \in \mathbb{N} \cup \{0\}$, our numerical approximation of D_t^α approximates the standard integer-order derivative of order α, it follows that, for example, $\mathbf{D}_t^\alpha$ should give similar results to those of the standard Chebyshev-differentiation matrix $\mathbf{D}$ generated by `cheb.m` [25]. Therefore, in order to make a fair comparison between $\mathbf{D}_t^\alpha$ and $\mathbf{D}$, we take $T = 2$, so that $t_0 - t_N = 2$ and $\mathbf{D}$ does not need to be scaled. On the left-hand side of Figure 4.1, we display $\mathrm{err}_{\hat{\mathbf{D}}_t^\alpha}$ (red) and $\mathrm{err}_{\mathbf{D}_t^\alpha}$ (blue), for $N = 175$, $T = 2$, and $\alpha \in \{0, 0.005, 0.01, \ldots, 4\}$, taking `dig` $= N$, and, additionally, we display (thick black points), for $\alpha \in \{1, 2, 3, 4\}$,

$$\mathrm{err}_{\mathbf{D}^\alpha} = \max_{0\le j\le N} \left| \frac{[\mathbf{D}^\alpha \cdot \hat{\mathbf{f}}]_j - (\mathrm{i}m)^\alpha e^{imt}}{(\mathrm{i}m)^\alpha e^{imt}} \right|, \tag{4.5}$$

where $\mathbf{D}^\alpha$ denotes the power α of $\mathbf{D}$. The results show that the errors corresponding to $\hat{\mathbf{D}}_t^\alpha$ and $\mathbf{D}_t^\alpha$ are very similar, and the same is valid for the errors corresponding to $\mathbf{D}$ and the natural values of α. Indeed, as with $\mathbf{D}^\alpha$, the results get worse as α increases, but they are remarkably

ETNA
Kent State University and
Johann Radon Institute (RICAM)

good for $\alpha \in [0, 2]$ and, especially, for $\alpha \in [0, 1]$. Note, however, that the graphs of the errors exhibits jumps at the natural values of α because the operator D_t^α is not continuous at these values of α. On the other hand, with regard to the numerical approximation of $I_t^\alpha e^{imt}$, the errors in the discrete ℓ^∞-norm are always of the order of $\mathcal{O}(10^{-15})$ for all the considered values of α.

In order to see the effect of N on the errors, we display on the right-hand side of Figure 4.1 the errors $\mathrm{err}_{\hat{\mathbf{D}}_t^\alpha}$ (red) and $\mathrm{err}_{\mathbf{D}_t^\alpha}$ (blue), for $\alpha = 0.97$, i.e., a value close to 1, and $N \in \{100, 105, \ldots, 1000\}$, taking $\texttt{dig} = N$, which are sufficiently many digits. The results are almost identical for $\mathrm{err}_{\hat{\mathbf{D}}_t^\alpha}$ and $\mathrm{err}_{\mathbf{D}_t^\alpha}$ and reveal that even if these quantities show a weak tendency to grow with N, they always stay below 10^{-10}. Therefore, we can say that the generation of $\hat{\mathbf{D}}_t^\alpha$ and $\mathbf{D}_t^\alpha$ remains stable even when large values of N are considered. With regard to the numerical approximation of $I_t^\alpha e^{imt}$, the errors in the discrete ℓ^∞-norm are again of the order of $\mathcal{O}(10^{-15})$ in the numerical experiments with $N \geq 155$.

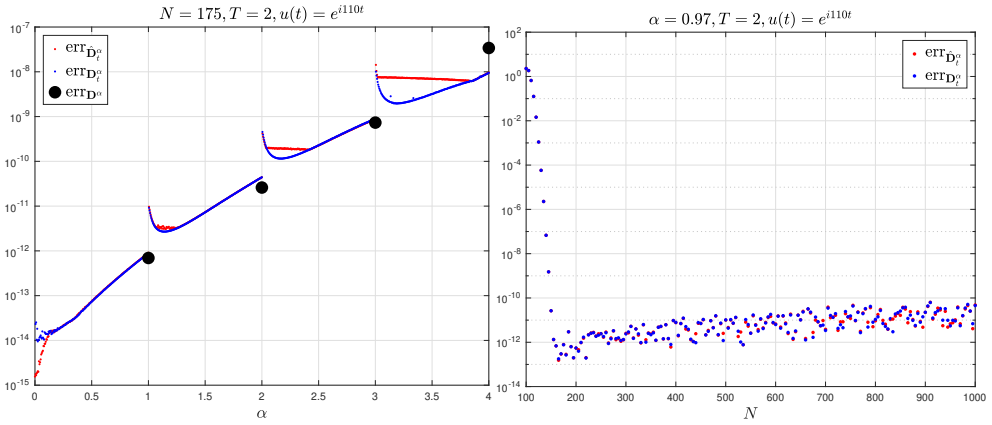

FIG. 4.1. *Left: Errors* $\mathrm{err}_{\hat{\mathbf{D}}_t^\alpha}$ *(red) and* $\mathrm{err}_{\mathbf{D}_t^\alpha}$ *(blue) as defined in* (4.3) *and* (4.4)*, respectively, for* $f(t) = e^{\mathrm{i}110t}$*, i.e.,* $m = 110$*, and* $\alpha \in \{0, 0.005, \ldots, 4\}$*. For the sake of comparison, we also display* $\mathrm{err}_{\mathbf{D}^\alpha}$*, as defined in* (4.5)*, for* $\alpha \in \{1, 2, 3, 4\}$ *(thick black point). Right: Errors* $\mathrm{err}_{\hat{\mathbf{D}}_t^\alpha}$ *(red) and* $\mathrm{err}_{\mathbf{D}_t^\alpha}$ *(blue), for* $f(t) = e^{\mathrm{i}110t}$*, and* $N \in \{100, 105, \ldots, 1000\}$.

5. Numerical simulation of Caputo-type advection-diffusion equations. The constructed matrices allow us, as in [10], to numerically approximate Caputo-type advection-diffusion equations of the form

$$\begin{cases} D_t^\alpha u(t,x) = a_1(x)u_{xx}(t,x) + a_2(x)u_x(t,x) \\ \qquad\qquad + a_3(x)u(t,x) + a_4(t,x), \qquad t \in [0,T], \\ u(0,x) = u_0(x), \end{cases} \tag{5.1}$$

where $a_1(x)$, $a_2(x)$, $a_3(x)$, $a_4(t,x)$ are at least continuous and $x \in \mathbb{R}$ or $x \in [x_a, x_b]$, although other types of spatial domains like semi-infinite ones could be considered, as well as PDEs containing higher-order derivatives with respect to x. As stated in the introduction, the idea is to transform (5.1) into a Sylvester equation of the form $\mathbf{A} \cdot \mathbf{X} + \mathbf{X} \cdot \mathbf{B} = \mathbf{C}$ and then solve it. Moreover, by using the ideas in, e.g., [9], we may consider generalizations of (5.1) to d spatial dimensions, with $d > 1$, and transform the equation into so-called Sylvester tensor equations. Therefore, let us first consider an example in one spatial dimension and then another example in multiple spatial dimensions.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

5.1. An example in one spatial dimension. In one spatial dimension, the procedure is very similar to that explained in [10], so we omit many details that can be consulted in that reference, especially how to incorporate the boundary conditions. Note, however, that, unlike in [10], the nodes $t_j \in [0,T]$ as defined in (2.1) are given in decreasing order, i.e., $t_0 = T > t_1 > \cdots > t_N = 0$, which implies a small modification of the codes in [10]. To illustrate this, we consider the following example:

$$\text{(5.2)} \quad \begin{cases} D_t^\alpha u(t,x) = u_{xx}(t,x) + 2xu_x(t,x) + 2u(t,x) \\ \qquad\qquad + (\mathrm{i}m)^\alpha \left(1 - \dfrac{\Gamma(\lceil\alpha\rceil - \alpha, \mathrm{i}mt)}{\Gamma(\lceil\alpha\rceil - \alpha)}\right) e^{\mathrm{i}mt - x^2}, \quad (t,x) \in [0,2] \times \mathbb{R}, \\ u(0,x) = u_0(x) = e^{-x^2}, \end{cases}$$

where $\alpha \in (0,1)$, $m \in \mathbb{N}$, and whose solution is $u(t,x) = e^{\mathrm{i}mt - x^2}$.

In this case, we discretize the operator D_t^α by means of $\mathbf{D}_t^\alpha$, and the first and second spatial derivatives by means of $\mathbf{D}_x$ and $\mathbf{D}_x^2$. Here, $\mathbf{D}_x \in \mathbb{R}^{N_x \times N_x}$ is the Hermite differentiation matrix generated by the function `herdif`, which requires the functions `herroots` and `poldif` (see [27] for the three Matlab functions) and is based on the Hermite functions

$$\psi_n(x) = \frac{e^{-x^2/2}}{\pi^{1/4}\sqrt{2^n n!}} H_n(x),$$

with $H_n(x)$ being the Hermite polynomials:

$$H_n(x) = \frac{(-1)^n}{2^n} e^{x^2} \frac{d^n}{dx^n} e^{-x^2}.$$

The spatial nodes x_j, with $0 \le j \le N_x - 1$, are the roots of the polynomial $H_n(x)$ with the largest index, multiplied by a scale factor b, which is precisely the third parameter of `herdif`. Therefore, we want to construct a matrix $\mathbf{U} = [u_{jk}] \in \mathbb{C}^{(N_t+1)\times N_x}$ such that $u_{jk} \approx u(t_j, u_k)$; note that we write N_t instead of N to avoid any confusion with N_x.

Following the ideas in [10], let us denote

$$\text{(5.3)} \qquad \mathbf{E} = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \\ \hline 0 & \cdots & 0 \end{bmatrix}, \qquad \mathbf{F} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \\ \hline u_0(x_0) & \cdots & u_0(x_{N_x}) \end{bmatrix}.$$

Note that, due to the fact that the nodes t_j are defined in decreasing order, we store an all-zero vector in the last row of $\mathbf{E}$, and the initial condition $u_0(x)$ is stored in the last row of $\mathbf{F}$, whereas in [10] they were stored in the first rows of $\mathbf{E}$ and $\mathbf{F}$, respectively. Then, we decompose $\mathbf{U}$ as

$$\text{(5.4)} \qquad \mathbf{U} = \mathbf{E} \cdot \mathbf{U}_{\text{inner}} + \mathbf{F},$$

where $\mathbf{U}_{\text{inner}} \in \mathbb{C}^{N_t \times N_x}$ denotes the matrix of $\mathbf{U}$ without its last row but keeping all its columns (in [10] we did not keep the first row of $\mathbf{U}$ to define $\mathbf{U}_{\text{inner}}$):

$$\mathbf{U}_{\text{inner}} = \begin{bmatrix} u_{11} & \cdots & u_{1,N_x} \\ \vdots & \ddots & \vdots \\ u_{N_t,1} & \cdots & u_{N_t,N_x} \end{bmatrix}.$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Moreover, denoting $\mathbf{x} = (x_0, x_1, \ldots, x_{N_x-1})$ and $\operatorname{diag}(\mathbf{x})$ the diagonal matrix whose diagonal entries are precisely those of $\mathbf{x}$, we define

$$\mathbf{G} = \mathbf{D}_x^2 + 2\operatorname{diag}(\mathbf{x})\mathbf{D}_x + 2\mathbf{I} \in \mathbb{R}^{N_x \times N_x},$$

and $\mathbf{H} = [h_{jk}] \in C^{(N_t+1)\times N_x}$ such that

$$h_{jk} = (\mathrm{i}m)^\alpha \left(1 - \frac{\Gamma(\lceil\alpha\rceil - \alpha, \mathrm{i}mt_j)}{\Gamma(\lceil\alpha\rceil - \alpha)}\right) e^{\mathrm{i}mt_j - x_k^2}.$$

Thus, we can discretize (5.2) without its initial data as

$$\mathbf{D}_t^\alpha \cdot \mathbf{U} = \mathbf{U} \cdot \mathbf{G}^T + \mathbf{H},$$

and, in order to incorporate the initial condition, we replace $\mathbf{U}$ by (5.4):

$$\mathbf{D}_t^\alpha \cdot (\mathbf{E} \cdot \mathbf{U}_{\text{inner}} + \mathbf{F}) = (\mathbf{E} \cdot \mathbf{U}_{\text{inner}} + \mathbf{F}) \cdot \mathbf{G}^T + \mathbf{H}.$$

Then, left-multiplying by $\mathbf{E}^T$ and rearranging the terms, we get

$$\mathbf{E}^T \cdot \mathbf{D}_t^\alpha \cdot \mathbf{E} \cdot \mathbf{U}_{\text{inner}} - \mathbf{U}_{\text{inner}} \cdot \mathbf{G}^T = -\mathbf{E}^T \cdot \mathbf{D}_t^\alpha \cdot \mathbf{F} + \mathbf{E}^T \cdot \mathbf{H},$$

where we have used that $\mathbf{E}^T \cdot \mathbf{E} = \mathbf{I}$ and that $\mathbf{E}^T \cdot \mathbf{F} = \mathbf{0}$. Defining

$$\mathbf{A} = \mathbf{E}^T \cdot \mathbf{D}_t^\alpha \cdot \mathbf{E}, \qquad \mathbf{B} = -\mathbf{G}^T, \qquad \mathbf{C} = -\mathbf{E}^T \cdot \mathbf{D}_t^\alpha \cdot \mathbf{F} + \mathbf{E}^T \cdot \mathbf{H},$$

we solve $\mathbf{A} \cdot \mathbf{U}_{\text{inner}} + \mathbf{U}_{\text{inner}} \cdot \mathbf{B} = \mathbf{C}$ by means of the Matlab command `lyap` and apply (5.4) to obtain $\mathbf{U}$. In Listing 4, we provide the Matlab implementation corresponding to $\alpha = 0.97$ (i.e., a value of α close to 1), $m = 330$, $N_x = 16$, $N_t = 400$, `dig` $= 400$, and $T = 2$. Note that, due to the highly oscillating nature of the solution $u(t, x)$ in time, it is obligatory to take a large enough matrix $\mathbf{D}_t^\alpha$. The code takes 158.48 seconds to run, and $\|\mathbf{U}_{exact} - \mathbf{U}\|_{\max} = 6.1766 \times 10^{-13}$, which, in our opinion, is a pretty remarkable result.

LISTING 4
Matlab program `pde2D.m` *that solves* (5.2) *numerically.*

```
clear
tic
a=0.97; % $\alpha$
m=330; % $m$
Nx=16; % $N_x$
Nt=400; % $N_t$
dig=400; % Number of digits
T=2; % $T$
[~,Dta,t]=CaputoMatrix(Nt,a,T,dig);
b=1.4; % Scale factor $b$
[x,DD]=herdif(Nx,2,b);
Dx=DD(:,:,1); % $\mathbf{D_x}$
Dx2=DD(:,:,2); % $\mathbf{D_x^2}$
[tt,xx]=ndgrid(t,x); % Two-dimensional grid
u0=exp(-x.^2).'; % $u_0$
u=exp(1i*m*t-xx.^2); % $u$
E=[eye(Nt);zeros(1,Nt)]; % $\mathbf{E}$
F=[zeros(Nt,Nx);u0]; % $\mathbf{F}$
G=Dx2+2*x.*Dx+2*eye(Nx);
H=((1i*m)^a*(1-igamma(ceil(a)-a,1i*m*t)/gamma(ceil(a)-a))) ...
    .*exp(1i*m*tt-xx.^2); % $\mathbf{H}$
A=E.'*Dta*E; % $\mathbf{A}$
B=-G.'; % $\mathbf{B}$
C=-E.'*Dta*F+E.'*H; % $\mathbf{C}$
Uinner=lyap(A,B,-C); % $\mathbf{U_{inner}}$
U=E*Uinner+F; % $\mathbf{U}$
norm(u(:)-U(:),inf) % Error in max norm
toc % Elapsed time
```

ETNA
Kent State University and
Johann Radon Institute (RICAM)

5.2. An example in multiple spatial dimensions. The ideas in Section 5.1 can be extended to the case of multiple spatial dimensions. To illustrate this, we consider the following Caputo-type advection-diffusion equation, which is a generalization of (5.2):

$$\text{(5.5)}\quad \begin{cases} \mathbf{D}_t^{\alpha} u(t,\mathbf{x}) = \Delta u(t,\mathbf{x}) + 2\mathbf{x}\cdot\nabla u(t,\mathbf{x}) + 2d\, u(t,\mathbf{x}) \\ \qquad\qquad + (\mathrm{i}m)^{\alpha}\left(1 - \dfrac{\Gamma(\lceil\alpha\rceil - \alpha, \mathrm{i}mt)}{\Gamma(\lceil\alpha\rceil - \alpha)}\right) e^{\mathrm{i}mt - \mathbf{x}\cdot\mathbf{x}}, \quad (t,x) \in [0,2]\times\mathbb{R}^d, \\ u(0,\mathbf{x}) = u_0(\mathbf{x}) = e^{-\mathbf{x}\cdot\mathbf{x}}, \end{cases}$$

where $\alpha \in (0,1)$, $m \in \mathbb{N}$, the diffusive term is $\Delta u = \partial^2 u/\partial x_1^2 + \partial^2 u/\partial x_2^2 + \ldots + \partial^2 u/\partial x_d^2$, and the advective term is $2\mathbf{x}\cdot\nabla u = 2x_1\partial u/\partial x_1 + 2x_2\partial u/\partial x_2 + \ldots + 2x_d\partial u/\partial x_d$. Moreover, the solution of (5.5) is $u(t,\mathbf{x}) = e^{\mathrm{i}mt - \mathbf{x}\cdot\mathbf{x}}$.

Then, similarly as in Section 5.1, we discretize the operator D_t^{α} by means of $\mathbf{D}_t^{\alpha}$ and the first and second spatial derivatives by means of $\mathbf{D}_{x_j}$ and $\mathbf{D}_{x_j}^2$, for $1 \le j \le d$, where $\mathbf{D}_{x_j} \in \mathbb{R}^{N_j \times N_j}$ is a Hermite differentiation matrix generated by the Matlab function `herdif`. This gives rise to an equation of the form

$$\text{(5.6)}\qquad \sum_{j=1}^{d+1} \mathbf{A}_j \square_j \mathbf{X} = \mathbf{A}_1\square_1\mathbf{X} + \mathbf{A}_2\square_2\mathbf{X} + \ldots + \mathbf{A}_{d+1}\square_{d+1}\mathbf{X} = \mathbf{B}, \qquad N \in \mathbb{N},$$

where $\mathbf{A}_j \equiv (a_{j,i_1i_2}) \in \mathbb{C}^{n_j\times n_j}$, $\mathbf{B} \equiv (b_{i_1i_2\cdots i_{d+1}}) \in \mathbb{C}^{n_1\times n_2\times\cdots\times n_{d+1}}$, $n_j \in \mathbb{N}$, for all j, and $\square_j$ indicates that the sum is performed along the jth dimension of the array $\mathbf{X} = (x_{i_1i_2\cdots i_{d+1}}) \in \mathbb{C}^{n_1\times n_2\times\cdots\times n_{d+1}}$, i.e.,

$$[\mathbf{A}_j\square_j\mathbf{X}]_{i_1i_2\cdots i_{N+1}} = \sum_{k=1}^{n_j} a_{j,i_jk} x_{i_1i_2\cdots i_{j-1}ki_{j+1}\cdots i_{d+1}}.$$

Some references on the solution of Sylvester tensor equations, like (5.6) for three dimensions, are, for example, [19], where the authors claimed to have solved this equation for the first time, and later on, [12, 13]. References for higher dimensions are more recent, and we can mention, e.g., [29], where a number of iterative algorithms based on Krylov spaces were applied; [8], where a tensor multigrid method and an iterative tensor multigrid method were proposed; [7], where recursive blocked algorithms were used; or [22], where a nested divide-and-conquer scheme was proposed. However, in this section we use the function `sylvesterND` developed in [9] that solves (5.6) by means of an iterative, non-recursive method based on the Bartels–Stewart algorithm [2] that relies on the Schur decomposition of the matrices $\mathbf{A}_j$ and whose implementation is reduced to one single for-loop and hence is independent of the number of dimensions.

Coming back to (5.5), let us denote by $\mathbf{U} \equiv [u_{t_j,i_1,\ldots,i_d}] \in \mathbb{C}^{(N_t+1)\times n_1\times\cdots\times n_d}$ the array such that $u_{t_j,i_1\ldots i_d} \approx u(t_j, x_{1,i_1}, \cdots, x_{d,i_d})$ and by $\mathbf{U}_{\text{inner}} \in \mathbb{C}^{N_t\times n_1\times\cdots\times n_d}$ the array $\mathbf{U}$ without the entries of the form $u_{N_t+1,i_1,\ldots,i_d}$. Then, defining the matrix $\mathbf{E}$ as in (5.3) and the array $\mathbf{F} \equiv [f_{t_j,i_1,\ldots,i_d}] \in \mathbb{C}^{(N_t+1)\times n_1\times\cdots\times n_d}$ consisting of zeros except for the entries of the form $f_{N_t+1,i_1,\ldots,i_d}$ given by $f_{N_t+1,i_1,\ldots,i_d} = u_0(x_{i_1},\ldots,x_{i_d})$, we have the following generalization of (5.4):

$$\text{(5.7)}\qquad \mathbf{U} = \mathbf{E}\square_1\mathbf{U}_{\text{inner}} + \mathbf{F}.$$

Moreover, let $\mathbf{x}_j = (x_{j,0}, x_{j,1}, \ldots, x_{j,n_j-1})$, and let $\operatorname{diag}(\mathbf{x}_j)$ be the diagonal matrix whose diagonal entries are precisely those of $\mathbf{x}_j$, for $1 \le j \le d$. Then, we define

$$\mathbf{G}_j = \mathbf{D}_{x_j}^2 + 2\operatorname{diag}(\mathbf{x}_j)\mathbf{D}_{x_j} + 2\mathbf{I} \in \mathbb{R}^{n_j\times n_j},$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

and $\mathbf{H} \equiv [h_{t_j,i_1,\ldots,i_d}] \in \mathbb{C}^{(N_t+1)\times n_1\times\ldots\times n_d}$ such that

$$h_{t_j,i_1,\ldots,i_d} = (\mathrm{i}m)^\alpha \left(1 - \frac{\Gamma(\lceil\alpha\rceil - \alpha, \mathrm{i}mt_j)}{\Gamma(\lceil\alpha\rceil - \alpha)}\right) e^{\mathrm{i}mt_j - x_{i_1}^2 - \ldots x_{i_d}^2}.$$

Thus, we can discretize (5.5) without its initial data as

$$\mathbf{D}_t^\alpha \square_1 \mathbf{U} = \sum_{j=1}^{d} \mathbf{G}_j \square_{j+1} \mathbf{U} + \mathbf{H},$$

and, in order to incorporate the initial condition, we replace $\mathbf{U}$ by (5.7):

$$\mathbf{D}_t^\alpha \square_1 (\mathbf{E}\square_1 \mathbf{U}_{\text{inner}} + \mathbf{F}) = \sum_{j=1}^{d} \mathbf{G}_j \square_{j+1} (\mathbf{E}\square_1 \mathbf{U}_{\text{inner}} + \mathbf{F}) + \mathbf{H}.$$

Then, left-multiplying by $\mathbf{E}^T$ along the first direction yields

$$\mathbf{E}^T \square_1 (\mathbf{D}_t^\alpha \square_1 (\mathbf{E}\square_1 \mathbf{U}_{\text{inner}} + \mathbf{F})) = \sum_{j=1}^{d} \mathbf{E}^T \square_1 (\mathbf{G}_j \square_{j+1} (\mathbf{E}\square_1 \mathbf{U}_{\text{inner}} + \mathbf{F})) + \mathbf{E}^T \square_1 \mathbf{H},$$

i.e.,

$$(\mathbf{E}^T \cdot \mathbf{D}_t^\alpha \mathbf{E})\square_1 \mathbf{U}_{\text{inner}} - \sum_{j=1}^{d} \mathbf{G}_j \square_{j+1} \mathbf{U}_{\text{inner}} = -(\mathbf{E}^T \cdot \mathbf{D}_t^\alpha)\square_1 \mathbf{F} + \mathbf{E}^T \square_1 \mathbf{H},$$

where we have used that $\mathbf{E}^T \cdot \mathbf{E} = \mathbf{I}$ and that $\mathbf{E}^T \square_1 \mathbf{F} = \mathbf{0}$. Defining

$$\mathbf{A}_1 = \mathbf{E}^T \cdot \mathbf{D}_t^\alpha \cdot \mathbf{E}, \qquad \mathbf{A}_{j+1} = -\mathbf{G}_j, \quad \text{for } 1 \le j \le d, \qquad \text{and}$$
$$\mathbf{B} = -(\mathbf{E}^T \cdot \mathbf{D}_t^\alpha)\square_1 \mathbf{F} + \mathbf{E}^T \square_1 \mathbf{H},$$

we solve

$$\sum_{j=1}^{d+1} \mathbf{A}_j \square_j \mathbf{U}_{\text{inner}} = \mathbf{B}$$

by means of the Matlab function `sylvesterND` from [9] and apply (5.7) to obtain $\mathbf{U}$. Note that we have defined the Matlab function `multND1` that computes $\square_1$, which is required to define $\mathbf{B}$ and to recover $\mathbf{U}$ from $\mathbf{U}_{\text{inner}}$:

```
function B=multND1(A,X)
sizeX=size(X); % size of $\mathbf X$
sizeB=[size(A,1),sizeX(2:end)]; % size of $\mathbf B$
B=reshape(A*reshape(X,sizeX(1),[]),sizeB); % $\mathbf B$
```

This function multiplies `A` by the multidimensional array `X`, which has been transformed into a two-dimensional matrix, and then transforms the product back into a multidimensional array. The idea of transforming $\mathbf{A}$ into a two-dimensional matrix is taken from function `multND` in [9], which is invoked internally by `sylvesterND` to multiply square matrices by multidimensional arrays but unlike `multND1` and `multND` requires permuting the dimensions of `X` in a convenient way.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

In Listing 5, we provide the Matlab implementation corresponding to $\alpha = 0.97$, $m = 29$, $N_x = 16$, $N_t = 60$, `dig` $= 60$, $T = 2$, and $d = 5$, i.e., in 5 spatial dimensions, so the corresponding $\mathbf{U}$ is a six-dimensional array. We have written the code so that it can be executed for any $d \in \mathbb{N}$. Note, for instance, the use of `cell`-structures to generate the $d+1$-dimensional grid,

```
aux=[{t};repmat({x},d,1)];
ttxx=cell(d+1,1);
[ttxx{:}]=ndgrid(aux{:});
```

or to define $\mathbf{F}$,

```
F=zeros([Nt+1,Nx*ones(1,d)]); % $\mathbf F$
index=[{Nt+1};repmat({':'},d,1)];
F(index{:})=u0;
```

The code takes 215.31 seconds to run, and $\|\mathbf{U}_{exact} - \mathbf{U}\|_{\max} = 6.5133 \times 10^{-14}$, which, again, we find to be pretty remarkable.

LISTING 5
Matlab program `pdeND.m` *that solves (5.5) numerically.*

```
clear
tic
d=5; % Number of spatial dimensions
a=0.97; % $\alpha$
m=29; % $m$
Nx=16; % $N_x$
Nt=60; % $N_t$
dig=60; % Number of digits
T=2; % $T$
[~,Dta,t]=CaputoMatrix(Nt,a,T,dig);
b=1.4; % Scale factor $b$
[x,DD]=herdif(Nx,2,b);
Dx=DD(:,:,1); % $\mathbf D_x$
Dx2=DD(:,:,2); % $\mathbf D_x^2$
aux=[{t};repmat({x},d,1)];
ttxx=cell(d+1,1);
[ttxx{:}]=ndgrid(aux{:}); % (d+1)-dimensional grid
xx=cell(d,1);
aux=aux(2:end);
[xx{:}]=ndgrid(aux{:});
u0=1; % Initial data $u_0$
u=exp(1i*m*t); % Exact solution $u$
for j=1:d
    u0=u0.*exp(-xx{j}.^2);
    u=u.*exp(-ttxx{j+1}.^2);
end
E=[eye(Nt);zeros(1,Nt)]; % $\mathbf E$
F=zeros([Nt+1,Nx*ones(1,d)]); % $\mathbf F$
index=[{Nt+1};repmat({':'},d,1)];
F(index{:})=u0;
H=((1i*m)^a*(1-igamma(ceil(a)-a,1i*m*t)/gamma(ceil(a)-a)))...
    .*exp(1i*m*t); % $\mathbf H$
for j=1:d
    H=H.*exp(-ttxx{j+1}.^2);
end
B=-multND1(E.'*Dta,F)+multND1(E.',H); % $\mathbf B$
G=Dx2+2*x.*Dx+2*eye(Nx); % $\mathbf G$
AA=cell(d+1,1); % Store the matrices $\mathbf A_j$
AA{1}=E.'*Dta*E;
for j=1:d
    AA{j+1}=-G;
```

ETNA
Kent State University and
Johann Radon Institute (RICAM)


```
end
Uinner=sylvesterND(AA,B); % $\mathbf U_{inner}$
U=multND1(E, Uinner)+F; % $\mathbf U$
norm(U(:)-u(:),inf) % Error in max norm
toc % Elapsed time
```

REFERENCES

[1] W. M. ABD-ELHAMEED, J. A. T. MACHADO, AND Y. H. YOUSSRI, *Hypergeometric fractional derivatives formula of shifted Chebyshev polynomials: tau algorithm for a type of fractional delay differential equations*, Int. J. Nonlinear Sci. Numer. Simul., 23 (2022), pp. 1253–1268.

[2] R. H. BARTELS AND G. W. STEWART, *Algorithm 432 [C2]: Solution of the matrix equation $AX + XB = C$ [F4]*, Comm. ACM, 15 (1972), pp. 820–826.

[3] J. P. BOYD, *Chebyshev and Fourier Spectral Methods*, 2nd ed., Dover, Mineola, 2001.

[4] M. CAI AND C. LI, *Numerical approaches to fractional integrals and derivatives: a review*, Mathematics, 8 (2020), Paper No. 43, 53 pages.

[5] J. CAO, C. LI, AND Y. CHEN, *High-order approximation to Caputo derivatives and Caputo-type advection-diffusion equations (II)*, Fract. Calc. Appl. Anal., 18 (2015), pp. 735–761.

[6] J. CAYAMA, C. M. CUESTA, AND F. DE LA HOZ, *Numerical approximation of the fractional Laplacian on $\mathbb{R}$ using orthogonal families*, Appl. Numer. Math., 158 (2020), pp. 164–193.

[7] M. CHEN AND D. KRESSNER, *Recursive blocked algorithms for linear systems with Kronecker product structure*, Numer. Algorithms, 84 (2020), pp. 1199–1216.

[8] Y. CHEN AND C. LI, *A tensor multigrid method for solving Sylvester tensor equations*, IEEE Trans. Autom. Sci. Eng., 21 (2024), pp. 4397–4405.

[9] C. M. CUESTA AND F. DE LA HOZ, *A non-recursive Schur-Decomposition algorithm for N-dimensional matrix equation*, Preprint on arXiv, 2024. https://arxiv.org/abs/2412.15840.

[10] F. DE LA HOZ AND P. MUNIAIN, *Numerical approximation of Caputo-type advection-diffusion equations via Sylvester equations*, Preprint on arXiv, 2025. https://arxiv.org/abs/2501.09180

[11] ———, *Caputo-Riemann-Liouville-vpa*. GitHub repository, 2026. https://github.com/fdlhm/Caputo-Riemann-Liouville-vpa.

[12] F. DE LA HOZ AND F. VADILLO, *The solution of two-dimensional advection-diffusion equations via operational matrices*, Appl. Numer. Math., 72 (2013), pp. 172–187.

[13] ———, *A Sylvester-based IMEX method via differentiation matrices for solving nonlinear parabolic equations*, Commun. Comput. Phys., 14 (2013), pp. 1001–1026.

[14] A. K. FARHOOD AND O. H. MOHAMMED, *Shifted Chebyshev operational matrices to solve the fractional time-delay diffusion equation*, Partial Differ. Equ. Appl. Math., 8 (2023), Paper No. 100538, 8 pages.

[15] M. FRIGO AND S. G. JOHNSON, *The design and implementation of FFTW3*, Proc. IEEE, 93 (2005), pp. 216–231.

[16] J. R. GRAEF, L. KONG, AND M. WANG, *A Chebyshev spectral method for solving Riemann-Liouville fractional boundary value problems*, Appl. Math. Comput., 241 (2014), pp. 140–150.

[17] M. S. HASHEMI, M. MIRZAZADEH, M. BAYRAM, AND S. M. EL DIN, *Numerical approximation of the Cauchy non-homogeneous time-fractional diffusion-wave equation with Caputo derivative using shifted Chebyshev polynomials*, Alexandria Eng. J., 81 (2023), pp. 118–129.

[18] R. KRASNY, *A study of singularity formation in a vortex sheet by the point-vortex approximation*, J. Fluid Mech., 167 (1986), pp. 65–93.

[19] B.-W. LI, S. TIAN, Y.-S. SUN, AND Z.-M. HU, *Schur-decomposition for 3D matrix equations and its application in solving radiative discrete ordinates equations discretized by Chebyshev collocation spectral method*, J. Comput. Phys., 229 (2010), pp. 1198–1212.

[20] C. LI, R. WU, AND H. DING, *High-order approximation to Caputo derivatives and Caputo-type advection-diffusion equations (I)*, Commun. Appl. Ind. Math., 6 (2014), Paper No. e-536, 32 pages.

[21] H. LI, J. CAO, AND C. LI, *High-order approximation to Caputo derivatives and Caputo-type advection-diffusion equations (III)*, J. Comput. Appl. Math., 299 (2016), pp. 159–175.

[22] S. MASSEI AND L. ROBOL, *A nested divide-and-conquer method for tensor Sylvester equations with positive definite hierarchically semiseparable coefficients*, IMA J. Numer. Anal., 44 (2024), pp. 3482–3519.

[23] MATHWORKS, *MATLAB, Version R2024b*, 2024.

[24] I. PODLUBNY, *Fractional Differential Equations*, Academic Press, San Diego, 1999.

[25] L. N. TREFETHEN, *Spectral Methods in MATLAB*, SIAM, Philadelphia, 2000.

[26] L. WANG AND Y.-M. CHEN, *Shifted-Chebyshev-polynomial-based numerical algorithm for fractional order polymer visco-elastic rotating beam*, Chaos Solitons Fractals, 132 (2020), Paper No. 109585, 8 pages.

[27] J. A. C. WEIDEMAN AND S. C. REDDY, *A MATLAB differentiation matrix suite*, ACM Trans. Math. Software, 26 (2000), pp. 465–519.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

[28] WOLFRAM RESEARCH, *Mathematica, Version 14.0*.
[29] X.-F. ZHANG AND Q.-W. WANG, *Developing iterative algorithms to solve Sylvester tensor equations*, Appl. Math. Comput., 409 (2021), Paper No. 126403, 14 pages.