Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 402 (2022) 115537
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Approximate Fourier series recursion for problems involving temporal fractional calculus[☆]

Simon Shaw, John R. Whiteman

Department of Mathematics, Brunel University London, Uxbridge, UB8 3PH, England, United Kingdom

Available online 29 August 2022

This article is dedicated to Tinsley Oden, collaborator, colleague and friend over many years.

Abstract

Fractional calculus constitutive models are of wide applicability. They have been used, for example, for: xanthan gum, bread dough, synthetic polymers (e.g. nylon), single collagen fibrils, semi-hard zero-fat cheese, pure ice at $-35°$C, asphalt sealants, epithelial cells, gels, polymers, concrete, asphalt, rock mass, waxy crude oil, breast tissue cells, lung parenchyma, and red blood cell membranes (e.g. Bonfanti et al. (2020)). In this note we are motivated by applications to dispersive media such as lossy dielectrics and viscoelastic polymers, both of which involve constitutive laws with fading memory convolution Volterra integrals, and both of which have been modelled by fractional calculus — which implies a weakly singular Volterra kernel. To step forward in time with such a model the entire hereditary Volterra integral needs to be numerically re-computed at each time step. Done naïvely with standard quadrature this involves $O(N^2)$ operations and $O(N)$ storage to compute over N time steps, and this severely limits the usefulness of such simple methods in large scale computations. Several alternatives addressing these shortcomings of this naïve approach have been offered in the literature, all of which can provide remedies. Here we propose another method, but ours is rather different in that we use a Fourier series proxy over much of the integration range. This allows for a recursive update to the 'history integral', with a much smaller effort expended in standard quadrature near the singularity. The recursive update is basically the same as that used when the Volterra kernel comprises decaying exponentials, a method which is already employed in some commercial codes. Given that this partial implementation is already in use, we intend that our method therefore has a lower 'barrier to entry' for incorporating fractional calculus models into commercial codes: it requires only minor surgery on the weakly singular kernel in order to apply the Fourier series proxy. This study is presented in the spirit of a discussion piece — there are several directions one could take from the basic observations presented here. We have aimed for simplicity and economy of presentation, and included just enough numerical results to give evidence that the method works. In summary, our method mitigates the $O(N)$ storage and $O(N^2)$ storage issues using an easily implemented proxy. It can also be implemented for variable time steps. The code is available from https://github.com/variationalform/fouvol and https://hub.docker.com/u/variationalform.

MSC: 42A16; 26A33; 74S40; 45D05

☆ The work of Shaw was supported in part by the Engineering and Physical Sciences Research Council, United Kingdom (EP/H011072/1). The work of Whiteman on viscoelasticity was supported in part by the Oden Institute (TICAM and ICES), United States at the University of Texas at Austin during his many extended working visits. This support is gratefully acknowledged.

E-mail addresses: simon.shaw@brunel.ac.uk (S. Shaw), john.whiteman@brunel.ac.uk (J.R. Whiteman).

URLs: https://www.brunel.ac.uk/people/simon-shaw (S. Shaw), https://www.brunel.ac.uk/people/john-whiteman (J.R. Whiteman).

https://doi.org/10.1016/j.cma.2022.115537

Keywords: Viscoelasticity; Dispersive media; Hereditary integral; Fractional calculus; Cole–Cole dielectric; Finite element method

1. Introduction

Fractional calculus constitutive models for physical materials involve a quantity at current time being expressed non-locally in terms of a convolution Volterra integral of another quantity over the time history. The kernel in the integral is weakly singular, which has led some authors to call these 'power law' materials e.g. [1].

There are many references we could give but here, to set the wider context, we recall from [2] that among examples of such materials we have the food additive xanthan gum, bread dough, synthetic polymers (e.g. nylon), single collagen fibrils, semi-hard zero-fat cheese, pure ice at −35 °C, asphalt sealants, epithelial cells, gels, polymers, concrete, asphalt, rock mass, waxy crude oil, breast tissue cells, lung parenchyma, and red blood cell membranes.

To fix our ideas and provide a narrower context we are motivated here by the physical description of dispersive media such as lossy dielectrics or viscoelastic polymers. Both of which involve constitutive laws with fading memory convolution Volterra integrals and are described, for example, in [1,3,4].

Typically, in viscoelasticity for example, the constitutive law relating stress at time t, $\sigma(t)$, to strain, ϵ, is given as

$$\sigma(t)=\int_{-\infty}^{t}\varphi(t-s)\epsilon'(s)\,ds \tag{1}$$

with the prime denoting differentiation. Here, the convolution kernel is called a stress *relaxation function* and is usually either a Prony series of decaying exponentials or is weakly-singular:

$$\varphi(t)=\varphi_0+\sum_{q=1}^{Q}\varphi_q e^{-t/\tau_q} \qquad \text{or} \qquad \varphi(t)=\varphi_0 t^{\alpha} \tag{2}$$

for $Q\in\mathbb{N}$, $\varphi_q\geqslant 0$ and (relaxation times) $\tau_q>0$ for each q in the first case, and for $\varphi_0>0$ and $\alpha\in(-1,0)$ in the second.

The 'power law', on the right above, is of course connected to the *fractional calculus* (even though irrational α is also allowed) where, for example, the Riemann–Liouville integral for $\mathrm{Re}(\beta)>0$, and lower limit of zero, is defined by

$$D^{-\beta}u(t)=\int_0^t\frac{(t-s)^{\beta-1}}{\Gamma(\beta)}u(s)\,ds, \qquad \text{or} \qquad D^{-\alpha-1}u(t)=\int_0^t\frac{(t-s)^{\alpha}}{\Gamma(1+\alpha)}u(s)\,ds \tag{3}$$

where $\beta=1+\alpha\in(0,1)$.

On the other hand, in the modelling of Cole–Cole dielectrics Petropoulous in [5] introduced a Caputo fractional derivative, while Li et al. in [6] worked with the Letnikov fractional derivative, defined for $\alpha\in(-1,0)$ by,

$$\frac{\partial^{-\alpha}\boldsymbol{P}}{\partial t^{-\alpha}}=\frac{1}{\Gamma(1+\alpha)}\frac{d}{dt}\int_0^t(t-s)^{\alpha}\boldsymbol{P}(s)\,ds \tag{4}$$

to express polarization in terms of electric field, permittivities and relaxation time as

$$\tau^{-\alpha}\frac{\partial^{-\alpha}\boldsymbol{P}}{\partial t^{-\alpha}}+\boldsymbol{P}(t)=\epsilon_0(\epsilon_s-\epsilon_\infty)\boldsymbol{E}(t). \tag{5}$$

If we have the initial condition $\boldsymbol{P}(0)=\boldsymbol{0}$ then we can integrate by parts in (4) and take the time derivative through to get,

$$\begin{aligned}\frac{\partial^{-\alpha}\boldsymbol{P}}{\partial t^{-\alpha}}&=\frac{1}{\Gamma(1+\alpha)}\frac{d}{dt}\left(\frac{(t-s)^{1+\alpha}}{1+\alpha}\boldsymbol{P}(s)\Big|_t^0+\int_0^t\frac{(t-s)^{1+\alpha}}{1+\alpha}\boldsymbol{P}'(s)\,ds\right),\\ &=\frac{1}{\Gamma(1+\alpha)}\int_0^t(t-s)^{\alpha}\boldsymbol{P}'(s)\,ds\end{aligned} \tag{6}$$

and this is essentially the form used in [5].

In these settings, the stress in a linear viscoelastic body can be thought of as the forcing function in a Volterra first-kind equation for the strain rate, while after using (4) or (6) in (5) the polarization and electric field are related by an (non-fractional) intro-differential equation with some resemblance to a Volterra second-kind problem.

For further details on these physical models we defer to the cited material, here we wanted only to motivate the numerical evaluation of weakly singular Volterra integrals in the context of solving evolution equations. That said, we fix on a Volterra second kind problem and now restrict ourselves to the problem of finding $u: I = [0, T] \to \mathbb{R}$ such that

$$u(t) + (\varphi * u)(t) = f(t) \qquad \text{where} \qquad (\varphi * u)(t) := \int_0^t \varphi(t-s)u(s)\,ds \tag{7}$$

with $T > 0$ and f given, and with power law kernel, $\varphi(t) = \varphi_0 t^\alpha$ for $\alpha \in (-1, 0)$. Therefore, using (3), we see that (7) can also be written as

$$u(t) + \varphi_0 \Gamma(1+\alpha) D^{-\alpha-1} u(t) = f(t) \tag{8}$$

which, because $-\alpha - 1 \in (-1, 0)$, we can interpret as a fractional–integral equation. In fact, we will generalize this slightly as follows. The form of $\varphi(t)$ will in practice be obtained from experiments and the singularity at $t = 0$ may be an inconvenience in data fitting. After all, infinities are not likely to appear on a measuring device. For this reason we include an extra parameter, $\bar{t} \geqslant 0$, and consider the kernel in the form,

$$\varphi(t) = \varphi_0 \cdot (t + \bar{t})^\alpha. \tag{9}$$

The case $\bar{t} = 0$ matches the discussion above, while if $\bar{t} > 0$ we lose the weak singularity and the connection to fractional calculus, but we retain the flexibility of fitting non-infinite data. Ultimately, $\bar{t}$ is just a variable in a computer code.

The examples just given are usually just pieces in a bigger story involving sets of partial differential field equations. The fractional constitutive laws are used to provide connections between the field unknowns resulting in time dependent partial differential equations containing a Volterra operator which, as we can see above, accumulates the history of the solution. After a finite element discretization in space the function u in (the analogue of) (7) will be a vector of degrees of freedom with dimension given by that of the finite element space. Furthermore, when time is discretized into N_t intervals with time step $\Delta t = T/N_t$ the Volterra operator will need to be discretized using, in the most obvious case, a quadrature rule. With $t_n = n\,\Delta t$ and $U_n \approx u(t_n)$, the discrete version of (7) then takes the form,

$$U_n + \Delta t \sum_{m=0}^{n} \varpi_{nm} \varphi(t_n - t_m) U_m = f(t_n)$$

for quadrature weights $\{\varpi_{nm}\}$. Clearly some care is needed when $m = n$ when the power law is used.

The analysis of numerical schemes for problems that take this approach is very advanced for elliptic, parabolic and hyperbolic integrodifferential Volterra problems (see e.g. [7–10] as well as the book [11]) with some, but not all, of these acknowledging the practical difficulties associated with using the power law in place of the Prony series.

For example, if V_{n-1} is an approximation to $(\varphi * u)(t_{n-1})$ with $\varphi(t) = \varphi_1 e^{-t/\tau_1}$ then, obviously, V_n follows from $\int_{t_{n-1}}^{t_n} \varphi(t_n - s)u(s)\,ds + e^{-\Delta t/\tau_1} V_{n-1}$ after approximating the first term, and this is used to obtain $U_n \approx u(t_n)$. This recursion allows us to compute over all N_t time steps with only $O(N_t)$ operations and $O(1)$ storage associated with the history integral, and it also allows alternative formulations in which the Volterra term is replaced by temporally local *internal variables*, each of which evolve according to an ordinary differential equation (see e.g. [12–14]).

The fractional calculus (power law) model does not allow for any such temporal localization and so a naïve quadrature approximation (requiring a sum over n for each $n = 1, 2, \dots, N_t$) needs $O(N_t^2)$ operations to compute over all of the N_t time steps. Moreover, in the Prony series model one need only store a vector of internal variables for each of the Q 'relaxation modes' in the left of (2) and update these with the recursion just outlined. On the other hand, for this naïve discretization of the power law we would have to store a solution vector for every time step, an $O(N_t)$ storage burden, in order to recompute the convolution at every time level in the solution time stepping.

To see the effect of this naïve approach consider a computational grid on $\Omega = (0, 1)^d \subset \mathbb{R}^d$ (for $d = 2, 3$) having $N - 1$ intervals in each coordinate direction. Disregarding the negligible effect of the boundary values, a scalar PDE (e.g. involving $\nabla^2 u$ with piecewise linears) discretized on this grid has a solution vector of length N^d which, in eight-byte (double) precision, requires $8N^d$ bytes of RAM. If $N = 500$, which is realistic – maybe even modest – for hi-fidelity simulations in many cases of interest, then a single solution vector requires 8×500^2 bytes in 2D and

8×500^3 bytes in 3D, or about 2 MB in 2D and 1 GB in 3D. Assuming the atypical luxury of hardware with 64 GB of RAM, and disregarding all other storage demands made by the numerical scheme, we could therefore store a maximum of 32,000 grid vectors in 2D, but only 64 in 3D. This puts a hard limit on the potential for hi-fidelity time stepping of power law models with naïve quadrature discretization: a maximum of 32,000 time steps in 2D (which may be satisfactory) but only 64 in 3D! A more typical situation with 16 GB of RAM reduces the 2D capability to 8000 time steps, with no realistic possibility of 3D time domain simulation capability.

It would seem then that the choice of Prony series for the Volterra kernel should be routine, even automatic, for scientific computing — but this is not the case. We compute to solve physical problems and in so doing we need our model to fit the physics of interest, and cannot expect these physics to fit our preferred model. In particular, and in reference to experimental data in viscoelasticity, it has often been observed (e.g. [2], [1, §1.6.5] and [15, §2.5]) that the power law does a much better job of describing both the short and long-time creep in rigid and high polymers.

For this reason it is important for power law kernels to derive non-naïve approximations to the weakly-singular Volterra integral both to save on the storage requirements and, preferably, also to save on the quadratically increasing execution time when time stepping. In this direction Sloan & Thomée in [9] proposed a *sparse quadrature* rule that increased in order as the integration time variable increased backwards, and yet only matched the accuracy of the lower order rule at the frontier of the time stepping. The quadrature points became sparse in the history of the quadrature and so the operation counts and storage requirements drop significantly. (See the ‘economical’ rules in [11], which is also a good overview source for problems involving Volterra ‘histories’)

This sparse scheme was extended to incorporate weakly singular kernels by Adolfsson et al. and then studied and used in [16,17].

Later Schädle, López-Fernández and Lubich [18] developed the well-known convolution quadrature method, and there is also McLean’s interval clustering method as presented in [19]. These also dramatically reduce storage and arithmetic requirements and can be applied to weakly singular kernels.

These methods are well documented and work well, but they do not seem to have been widely adopted for practical problems of the type we mentioned earlier. Perhaps this is because they require specialist, and in places, non-trivial coding.

The purpose of this note is to point out that the recursion mentioned earlier, based on $V_n =$ approx of $\left(\int_{t_{n-1}}^{t_n} \varphi(t_n - s)u(s)\,ds\right) + e^{-\Delta t/\tau_1} V_{n-1}$, applies also to complex exponentials, and therefore also to Fourier series (see later in (15)). This suggests use of a Fourier series approximation of the weakly singular power law kernel to obtain a recursive history updating algorithm with economical storage requirements, exactly as for the exponential Prony series. Some care needs to be taken with the singularity – where the Fourier series cannot be used – and that is why we call this method an *approximate* Fourier series recursion

With this observation in mind, our idea is then simple: to solve our problem over $[0, T]$, we choose a $T_1 \in (0, T)$ and define an extended time $T_x = T + T_1$. On $[0, T_1] \cup [T, T_x]$ the power law kernel is replaced by Hermite splines which are constructed in such a way that the even periodic extension, called ψ later, of this piecewise $[0, T_x] \to \mathbb{R}$ function is $C^m(\mathbb{R})$ for some chosen $m \in \mathbb{N}$. We then numerically compute the complex exponential Fourier series, $F\psi$, for this $[0, T_x]$ piecewise function ψ (to a chosen number of terms — this is L in (14)) and use this as a proxy for φ on $[T_1, T]$. See Figs. 1 and 2. We never require the segment on $[T, T_x]$ in the solve, while the segment on $[0, T_1]$ must be dealt with in a way that respects the weak singularity.

Herein we choose a simple quadrature using an ‘open’ rectangle rule over the last interval $[t_n - \Delta t, t_n]$. The ‘open-ness’ means we never need to evaluate the kernel at the singularity. So, once the time stepping using naïve quadrature over $t_1, t_2, \ldots$, has advanced so far that $t_n > T_1$, the computation of $\int_0^{t_n} \varphi(t_n - s)u(s)\,ds$ is effected by retaining the naïve quadrature rule over $[T_1, t_n]$, with a one-sided rectangle rule on $[t_{n-1}, t_n]$ so as to avoid needing $\varphi(0)$. We then use the Fourier series proxy in place of the weakly singular kernel and employ the recursive update of the ‘tail’ integral $\int_0^{t_n - T_1} \varphi(t_n - s)u(s)\,ds$ as furnished by the complex exponentials. The detail is given below and culminates in the formula in (15).

The result is an algorithm that has a variational structure for the Fourier series, with a clear path to error estimation, along with reduced storage requirements and increased speed (a decreased operation count). It is important to realize that in this method T_1 is fixed. This implies that the operation count remains $O(N_t^2)$ for this scheme and the storage requirement remains $O(N_t)$ (this was also noted for the sparse method in [16]). However, the gains can be significant and could mean the difference between a practical computation and an impractical one.

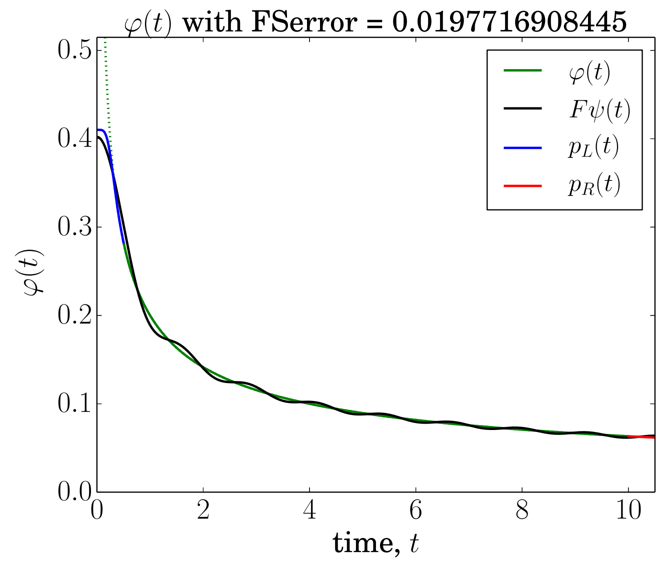


Fig. 1. An example of the proxy for $\varphi(t) = 0.2t^{-0.5}$ for $T_x = T + T_1$, for $T = 10$ and $T_1 = 0.5$. The solid green line shows φ over [1, 10], and the dashed green line shows φ in (0, 1], asymptoting to ∞ at $t = 0$. The blue and red lines show the Hermite spline surgery (for $m = 3$), and the black line shows the 33-term ($L = 16$ in (14)) even Fourier series proxy. The value of FSerror is a crude approximation to the $L_\infty(T_1, T)$ error between φ and the proxy. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

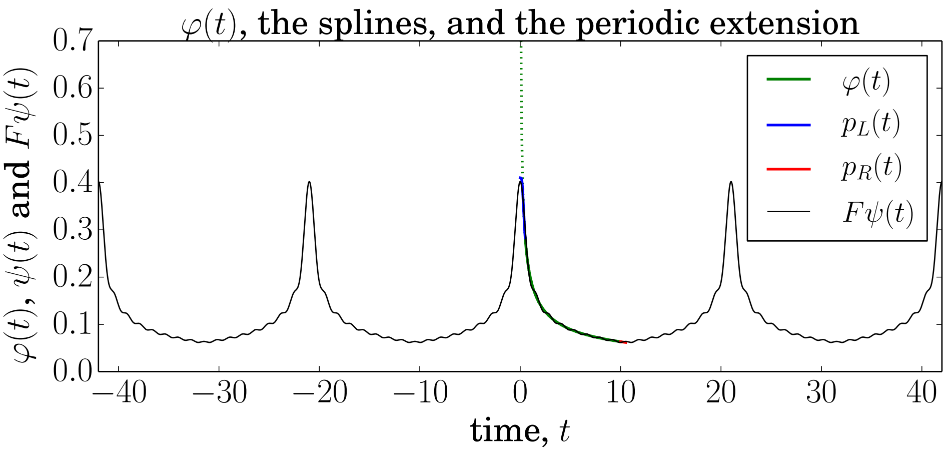


Fig. 2. An illustration of the proxy over a longer time domain. Here we see four periods for the same data as in Fig. 1.

To see this let us return to the examples above. Suppose we set $T_1 = \xi T$ for $\xi \in [0, 1]$ and also require that $\Delta t = T/N_t = T_1/N_1$ so that $N_1 = \xi N_t$. Then, ignoring the bounded storage and operation counts associated with recursing the Fourier series proxy, in the 2D case above we can still store 32,000 solution vectors but these need only be for the N_1 intervals for which we use quadrature. Therefore the number of permitted time steps, with 64 GB of RAM, magnifies to $N_t = 32,000/\xi$. If ξ is 5% then Δt can be made 20 times smaller with $N_t = 640,000$. In the 3D example, the 64 permitted time steps becomes $N_t = 64/\xi$, or 1280. For $\xi = 1\%$ these 2D and 3D figures move from 32,000 to $3,200,000$ and from 64 to 6400. The point is that ξ is fixed, but ours to choose.

For an audience used to high order methods, asymptotically optimal estimates, and rigorous proofs this may seem unsatisfactory and imprecise. In some sense it is, of course, but here we are adopting the view that for practical real life simulations where $h + \Delta t \to 0$ never happens, and in large application codes where software maintenance is very difficult and time consuming, this easy-to-code enhancement can readily furnish fractional calculus constitutive model enhancements capable of delivering engineering accuracy in an easy to implement framework.

We can also note that if experimental data implies that in practice we can expect $\bar{t} > 0$ in (9), then there is no weak singularity and we could allow T_1 to be proportional to Δt, thus significantly ameliorating the already mitigated storage and operation count burdens. This seems very worthy of future investigation.

We further note the related work in [20] in the context of the Havriliak–Negami dielectric model of induced polarization. There the integral over the most recent time step is dealt with using a local approximation based on an asymptotic expansion, as opposed to the 'open quadrature' used here. For the historical integral over times up to $T - \Delta t$ the kernel is approximated by decaying exponentials, exactly as here – and for the same reason – although not with Fourier approximation. Instead a generalized Gaussian quadrature is used to approximate the kernel, which is then approximated by a sum-of-exponentials to a given accuracy. The *Singular Value Decomposition* (SVD) is used to reduce the number of resulting exponentials to a constant M. This method has strong similarities to the one we develop below.

To give full details of our approach along with illustrative numerical results we stay with the scalar second-kind Volterra problem in (8), which is described in just enough detail for our needs in Section 2. The algorithm described above is then developed in detail in Section 3 and followed with an example numerical implementation in Section 4, where we combine the Fourier proxy with the product rectangle rule (see e.g. Linz, [21]), and then with some illustrative numerical results in Section 5. We then finish with a discussion in Section 6. The notation is standard throughout.

2. Volterra equations of the second kind

As already stated, we consider the problem of finding $u : I := [0, T] \to \mathbb{R}$ such that

$$u(t) + \int_0^t \varphi(t-s)u(s)\,ds = f(t) \tag{10}$$

for the kernel $\varphi(t) = \varphi_0(t+\bar{t})^\alpha$ with $\alpha \in (-1, 0)$, $\bar{t} \geqslant 0$ and $\varphi_0 > 0$. In the computations that come later we will take $\bar{t} = 0$ so as to retain the connection to fractional calculus. It is included here for the algorithm development and for incorporation in the code. In order to observe the convergence rates suggested by our numerical schemes we will need an exact solution. For this we take $u(t) = t^{-\alpha}$ and then, with $\bar{t} = 0$,

$$\begin{aligned} f(t) = u(t) + \int_0^t \varphi(t-s)u(s)\,ds &= t^{-\alpha} + \varphi_0 \int_0^t (t-s)^\alpha s^{-\alpha}\,ds, \\ &= t^{-\alpha} + \frac{\varphi_0 t \alpha\pi}{\sin(\alpha\pi)} \end{aligned} \tag{11}$$

because $t^\alpha * t^{-\alpha} = t\alpha\pi/\sin(\alpha\pi)$. This is most easily seen using the Laplace convolution formula: the inverse Laplace transform of the product of the transforms of t^α and $t^{-\alpha}$ is $t\Gamma(1-\alpha)\Gamma(1+\alpha)$ and then $\Gamma(1-\alpha)\Gamma(1+\alpha) = \alpha\Gamma(\alpha)\Gamma(1-\alpha) = \alpha\pi/\sin(\alpha\pi)$.

3. The Fourier series approximation

Given $T_1 \in (0, T)$ and some $T_x > T$ (we take $T_x = T + T_1$) the general idea is to consider the Volterra integral in two pieces, so that for $t \in [0, T]$,

$$\begin{aligned} \int_0^t \varphi(t-s)u(s)\,ds &= \int_0^{\min\{T_1,t\}} \varphi(t-s)u(s)\,ds + \int_{\min\{T_1,t\}}^t \varphi(t-s)u(s)\,ds, \\ &= \int_0^{\min\{T_1,t\}} \varphi(t-s)u(s)\,ds + \int_{\min\{T_1,t\}}^t \psi(t-s)u(s)\,ds, \end{aligned}$$

where $\psi \in C^m_{2T_x-\mathrm{ev-per}}(\mathbb{R})$ (the space of m-times continuously differentiable even $2T_x$-periodic $\mathbb{R} \to \mathbb{R}$ functions) is an even function constructed so that $\psi|_{[T_1,T]} = \varphi|_{[T_1,T]}$. As such, a Fourier series for ψ, denoted $F\psi$ for example, can serve as a proxy for φ on $[T_1, T]$.

The method of construction of ψ is to define a Hermite polynomial on $[0, T_1]$, denoted $p_L : [0, T_1] \to \mathbb{R}$, and of degree $2m+1$, where we choose $m \in \mathbb{N}$, such that $p_L^{(k)}(0) = 0$ for $k = 1, \ldots, m$ with $p_L(0)$ chosen between

$\varphi(T_1)$ and the $t=0$ intercept of the tangent of φ at T_1, and $p^{(k)}(T_1)=\varphi^{(k)}(T_1)$ for $k=0,1,\dots,m$. Further, on the right, we define another Hermite polynomial, $p_R\colon[T,T_x]\to\mathbb{R}$, also of degree $2m+1$ such that $p_R^{(k)}(T)=\varphi^{(k)}(T)$ for $k=0,1,\dots,m$ and $p_R^{(k)}(T_x)=0$ for $k=1,\dots,m$ with $p_R(T_x)$ chosen to lie between $\varphi(T)$ and the $t=T_x$ intercept of the tangent to φ at T. These polynomials are extended into $[T_1,T]$ by φ and the resulting $[0,T_x]\to\mathbb{R}$ function is reflected into $t=0$ to form an even $C[-T_x,T_x]\cap C^m(-T_x,T_x)$ function which is then periodically extended to $\mathbb{R}$. This is ψ.

The next step is to replace ψ in the Volterra integral with its Fourier series, $F\psi$. For this we recall first that

$$\psi(t)=\sum_{k=-\infty}^{\infty}c_k e^{i\pi kt/T_x}\qquad\text{where}\qquad c_k=\frac{1}{2T_x}\int_{-T_x}^{T_x}\psi(t)e^{-i\pi kt/T_x}\,dt,$$

for $-\infty<k<\infty$. Next, because ψ is even and real, we note that $c_k\in\mathbb{R}$ and $c_{-k}=c_k$ for every k and obtain

$$\psi(t)=\sum_{k=0}^{\infty}c_k e^{i\pi kt/T_x}\quad\text{where}\quad c_k=\begin{cases}\dfrac{2}{T_x}\displaystyle\int_0^{T_x}\psi(t)\cos(\pi kt/T_x)\,dt & \text{for } k\neq 0;\\ \dfrac{1}{T_x}\displaystyle\int_0^{T_x}\psi(t)\,dt & \text{for } k=0.\end{cases}$$

4. Implementation: the rectangle-Fourier rule

Recall that we allow $\bar{t}\geqslant 0$ in this for generality in the formulation and for inclusion in the code. Recall also that we set $\Delta t=T/N_t$ for some $N_t\in\mathbb{N}$, so that $t_n=n\Delta t$, and also we require $T_1\in(0,T]$ such that $N_1=T_1/\Delta t\in\mathbb{N}$. We use product quadrature rules, see e.g. Linz [21], whereby φ is integrated exactly near the singularity.

We will assume that $u(0)$ is not available to allow for the possibility that $f(0)$ is undefined. This is not important to the method presented below, and may not even be relevant to the background PDE problems we have in mind where, at least for parabolic and hyperbolic problems, $u(0)$ will stem from an initial condition.

The product-rectangle rule approximation to the Volterra equation problem in (10) is, for $u(t_n)\approx U_n$, and then $n=1,2,\dots$ in turn, find U_n such that,

$$U_n+\sum_{j=1}^{n}U_j\int_{t_{j-1}}^{t_j}\varphi(t_n-s)\,ds=f(t_n).$$

Using (since $\alpha\neq-1$),

$$\begin{aligned}\varphi_0\int_{t_{j-1}}^{t_j}(\bar{t}+t_n-s)^\alpha\,ds&=\frac{\varphi_0}{\alpha+1}\Big((\bar{t}+t_n-t_{j-1})^{\alpha+1}-(\bar{t}+t_n-t_j)^{\alpha+1}\Big),\\&=\frac{\varphi_0}{\alpha+1}\Big(\big(\bar{t}+(n-j+1)\Delta t\big)^{\alpha+1}-\big(\bar{t}+(n-j)\Delta t\big)^{\alpha+1}\Big),\end{aligned}$$

we then get,

$$\begin{aligned}U_n&=\frac{f(t_n)-\sum_{j=1}^{n-1}U_j\int_{t_{j-1}}^{t_j}\varphi(t_n-s)\,ds}{1+\int_{t_{n-1}}^{t_n}\varphi(t_n-s)\,ds},\\&=\frac{f(t_n)-\displaystyle\sum_{j=1}^{n-1}\frac{\varphi_0U_j}{\alpha+1}\Big((\bar{t}+(n-j+1)\Delta t)^{\alpha+1}-(\bar{t}+(n-j)\Delta t)^{\alpha+1}\Big)}{1+\dfrac{\varphi_0}{\alpha+1}\Big((\bar{t}+\Delta t)^{\alpha+1}-\bar{t}^{\alpha+1}\Big)},\\&=d^{-1}f(t_n)-d^{-1}\sum_{j=1}^{n-1}\frac{\varphi_0U_j}{\alpha+1}\Big((\bar{t}+(n-j+1)\Delta t)^{\alpha+1}-(\bar{t}+(n-j)\Delta t)^{\alpha+1}\Big),\end{aligned}\tag{12}$$

where

$$d:=1+\int_{t_{n-1}}^{t_n}\varphi(t_n-s)\,ds=1+\frac{\varphi_0}{\alpha+1}\Big((\bar{t}+\Delta t)^{\alpha+1}-\bar{t}^{\alpha+1}\Big).$$

This is the basic product-rectangle rule approximation and will be applied up to $t = T_1$ in the scheme we develop here. It can also be applied all the way to $t = T$, in which case we will simply have a naïve quadrature rule as discussed earlier in Section 1.

We now use the proxy and introduce the Fourier series approximation for when $t > T_1$. The infinite Fourier sum is replaced by a sum over the integers in $[0, L]$, for $L \in \mathbb{N}$ of our choosing. This results in,

$$\begin{aligned}
&\left(1+\int_{t_{n-1}}^{t_n} \varphi(t_n - s)\,ds\right) U_n \\
&\quad = f(t_n) - \sum_{j=1}^{n-1} U_j \int_{t_{j-1}}^{t_j} \varphi(t_n - s)\,ds, \qquad (13)\\
&\quad = f(t_n) - \sum_{j=n-N_1+1}^{n-1} U_j \int_{t_{j-1}}^{t_j} \varphi(t_n - s)\,ds - \sum_{j=1}^{n-N_1} U_j \int_{t_{j-1}}^{t_j} \varphi(t_n - s)\,ds,\\
&\quad = f(t_n) - \sum_{j=n-N_1+1}^{n-1} U_j \int_{t_{j-1}}^{t_j} \varphi(t_n - s)\,ds - \sum_{j=1}^{n-N_1} U_j \int_{t_{j-1}}^{t_j} \psi(t_n - s)\,ds,\\
&\quad \approx f(t_n) - \sum_{j=n-N_1+1}^{n-1} U_j \int_{t_{j-1}}^{t_j} \varphi(t_n - s)\,ds - \sum_{k=0}^{L} c_k \sum_{j=1}^{n-N_1} U_j \int_{t_{j-1}}^{t_j} e^{i\pi k(t_n-s)/T_x}\,ds. \qquad (14)
\end{aligned}$$

Define

$$\mathcal{H}_k(n) := c_k \sum_{j=1}^{n-N_1} U_j \int_{t_{j-1}}^{t_j} e^{i\pi k(t_n-s)/T_x}\,ds$$

so that

$$\mathcal{H}_k(n-1) := c_k \sum_{j=1}^{(n-1)-N_1} U_j \int_{t_{j-1}}^{t_j} e^{i\pi k(t_{n-1}-s)/T_x}\,ds.$$

It then follows that

$$\begin{aligned}
\mathcal{H}_k(n) &= c_k U_{n-N_1} \int_{t_{n-N_1-1}}^{t_{n-N_1}} e^{i\pi k(t_n-s)/T_x}\,ds + c_k \sum_{j=1}^{(n-1)-N_1} U_j \int_{t_{j-1}}^{t_j} e^{i\pi k(t_n-t_{n-1}+t_{n-1}-s)/T_x}\,ds,\\
&= c_k U_{n-N_1} \int_{t_{n-N_1-1}}^{t_{n-N_1}} e^{i\pi k(t_n-s)/T_x}\,ds + e^{i\pi k\Delta t} c_k \sum_{j=1}^{(n-1)-N_1} U_j \int_{t_{j-1}}^{t_j} e^{i\pi k(t_{n-1}-s)/T_x}\,ds,\\
&= c_k U_{n-N_1} \int_{t_{n-N_1-1}}^{t_{n-N_1}} e^{i\pi k(t_n-s)/T_x}\,ds + e^{i\pi k\Delta t}\,\mathcal{H}_k(n-1). \qquad (15)
\end{aligned}$$

Returning to (14), and accepting the approximation resulting from selecting $L < \infty$, then gives,

$$U_n = d^{-1} f(t_n) - d^{-1} \sum_{j=n-N_1+1}^{n-1} U_j \int_{t_{j-1}}^{t_j} \varphi(t_n - s)\,ds - d^{-1} \sum_{k=0}^{L} \mathcal{H}_k(n) \qquad (16)$$

The time marching scheme is therefore:

- Use (12) to determine U_n for $n = 1, 2, \dots, N_1$.
- Initialize each $\mathcal{H}_k(N_1) = 0$.
- For $n = N_1 + 1, N_1 + 2, \dots, N$:
  - Use (15) to determine each $\mathcal{H}_k(n)$.
  - Use (16) to determine U_n for $n = N_1 + 1, N_1 + 2, \dots, N_t$.

$N_t \downarrow \;\vert\; L \rightarrow$	0
32	1.841(−02)
64	9.369(−03)
128	4.741(−03)
256	2.389(−03)
512	1.201(−03)
1024	6.030(−04)
2048	3.023(−04)
4096	1.514(−04)
8192	7.582(−05)
16384	3.794(−05)
32768	1.898(−05)
65536	9.497(−06)

$N_t \downarrow \;\vert\; L \rightarrow$	0
32	1.000(+00)
64	0.000(+00)
128	0.000(+00)
256	1.000(+00)
512	0.000(+00)
1024	1.000(+00)
2048	2.000(+00)
4096	9.000(+00)
8192	3.400(+01)
16384	1.590(+02)
32768	6.390(+02)
65536	2.555(+03)

Fig. 3. Tabulated and graphical errors (left) and timings (right) for the product-Rectangle method with $\alpha = -0.5$ and $T = 10$.

In code the $2L+1$ quantities $\mathcal{H}_k$ can be updated for each n, overwriting the previous values according to (15). The operation counts and storage associated with these terms are then bounded independently of N. The storage and operation count burden, as discussed earlier, is carried by the middle term on the right of (16).

Note that in the formulae above there is no deep requirement placed on the time step Δt and so this scheme could be easily adapted to a variable time step algorithm.

5. Numerical experiments

In this section we give a few illustrative numerical results. There are many possible parameter combinations as we can vary φ_0, α and $\bar{t}$ in the kernel, along with T, T_1, N_t, L and m in the algorithm. To catalogue the results across this entire parameter range would require too much time and space and so we will report on just a few cases to show how the scheme works.

In particular, we have chosen the values $\varphi_0 = 0.2$, $\alpha = -0.5$ and $\bar{t} = 0$ in the kernel, and taken $T = 10$. We give results for the cases $T_1 = 0.5$, $T_1 = 0.1$ and $T_1 = 0.05$ (i.e. $\xi = 5\%$, 1% and 0.5% in the earlier discussion), and explore the errors and timings over ranges $N_t \in \{2^n : n = 5, 6, \ldots, 16\}$ and $L \in \{2^{2n+1} : n = 1, 2, \ldots, 5\}$.

To give a reference point we show in Fig. 3 the results for the pure naïve rectangle quadrature rule (i.e. (12)), and then give comparison results for the proxy, with $T_1 = 0.5$, in Figs. 4–7 for the cases $m = 1, 5, 10, 15$. The wall-clock timings are in seconds and the error is $|u(T) - U_{N_t}|$. Similar results are given in Figs. 8 and 9, for $T_1 = 0.1$, and in Figs. 10 and 11 for $T_1 = 0.05$, but just for $m = 1, 5$ in each case.

The results for the proxy calculations show both tabulated errors and timings and also their graphical representation. These make it easier to see where the convergence stalls for the lower order Fourier series approximations, and also how the set-up time overhead for larger L at small N_t becomes negligible for larger N_t. These figures also show an error/time comparison where the error and time values for each N_t are plotted for each L and also for the reference naïve rectangle quadrature rule. Note that the reference line is not useful at larger errors due to the negligible run time not fitting into the logarithmic axis.

The code is pure python and these results were obtained with python 2.7.17 on an `x86` Intel(R) Core(TM) i7-2640M CPU 2.80 GHz, using a linux mint 19.3 installation with kernel version `#82~18.04.1-Ubuntu SMP Fri Apr 16 15:10:02 UTC 2021`, and kernel release `5.4.0-73-generic`. The code can be obtained from

https://github.com/variationalform/fouvol

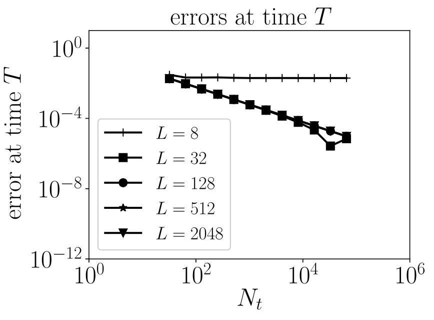
with `git clone git@github.com:variationalform/fouvol.git`. Or, from

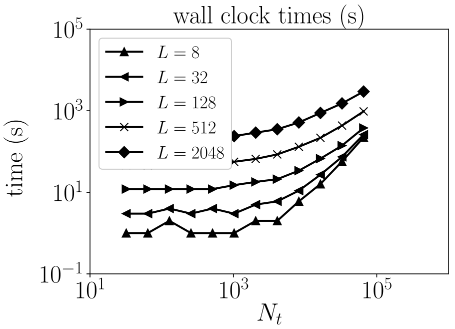
https://hub.docker.com/u/variationalform

with `docker pull variationalform/puretime:fouvol`.

$N_t \downarrow$ | $L \rightarrow$	8	32	128	512	2048
32	3.024(−02)	1.842(−02)	1.841(−02)	1.841(−02)	1.841(−02)
64	2.128(−02)	9.378(−03)	9.369(−03)	9.369(−03)	9.370(−03)
128	2.128(−02)	4.750(−03)	4.741(−03)	4.741(−03)	4.760(−03)
256	2.161(−02)	2.380(−03)	2.390(−03)	2.389(−03)	2.387(−03)
512	2.043(−02)	1.192(−03)	1.202(−03)	1.201(−03)	1.199(−03)
1024	1.983(−02)	5.938(−04)	6.032(−04)	6.030(−04)	6.007(−04)
2048	1.988(−02)	2.887(−04)	3.025(−04)	3.023(−04)	3.025(−04)
4096	1.991(−02)	1.354(−04)	1.516(−04)	1.514(−04)	1.515(−04)
8192	1.983(−02)	5.982(−05)	7.602(−05)	7.582(−05)	7.585(−05)
16384	1.979(−02)	2.194(−05)	3.815(−05)	3.795(−05)	3.797(−05)
32768	1.980(−02)	2.673(−06)	1.919(−05)	1.899(−05)	1.898(−05)
65536	1.980(−02)	6.972(−06)	9.699(−06)	9.499(−06)	9.497(−06)

$N_t \downarrow$ | $L \rightarrow$	8	32	128	512	2048
32	1.000(+00)	3.000(+00)	1.200(+01)	4.600(+01)	2.070(+02)
64	1.000(+00)	3.000(+00)	1.200(+01)	4.500(+01)	2.030(+02)
128	2.000(+00)	4.000(+00)	1.200(+01)	5.100(+01)	2.170(+02)
256	1.000(+00)	3.000(+00)	1.200(+01)	4.800(+01)	2.110(+02)
512	1.000(+00)	4.000(+00)	1.200(+01)	5.000(+01)	2.310(+02)
1024	1.000(+00)	3.000(+00)	1.500(+01)	5.600(+01)	2.390(+02)
2048	2.000(+00)	5.000(+00)	1.800(+01)	6.600(+01)	2.910(+02)
4096	2.000(+00)	6.000(+00)	2.100(+01)	8.500(+01)	3.520(+02)
8192	6.000(+00)	1.100(+01)	3.400(+01)	1.310(+02)	5.160(+02)
16384	1.600(+01)	2.700(+01)	6.700(+01)	2.170(+02)	8.840(+02)
32768	5.700(+01)	7.400(+01)	1.400(+02)	4.330(+02)	1.514(+03)
65536	2.230(+02)	2.600(+02)	3.790(+02)	9.650(+02)	2.958(+03)





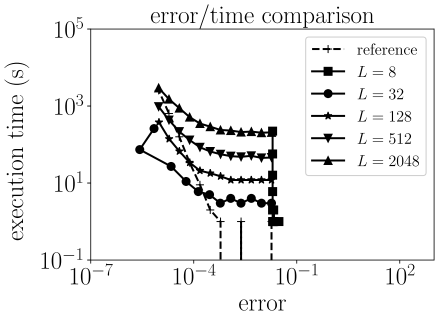


Fig. 4. Tabulated and graphical errors and timings for the product-Rectangle-Fourier method with $T = 10$, $\alpha = -0.5$, $T_1 = 0.5$ and a Hermite smoothness of $m = 1$.

As an implementation detail the value of T_1 may get altered somewhat because we are using constant Δt in this code. Specifically, given T, T_1 and N_t we define $\Delta t = T/N_t$ and then set $N_1 = \lceil T_1/\Delta t \rceil$. The value of T_1 is then adjusted so that $T_1 = N_1\,\Delta t$, and then we set the extended time as $T_x = T + T_1$.

We now move on to a discussion of these results.

$N_t \downarrow \mid L \rightarrow$	8	32	128	512	2048
32	3.151(−02)	1.801(−02)	1.841(−02)	1.841(−02)	1.841(−02)
64	2.255(−02)	8.972(−03)	9.369(−03)	9.369(−03)	9.370(−03)
128	2.286(−02)	4.516(−03)	4.741(−03)	4.741(−03)	4.760(−03)
256	2.334(−02)	2.264(−03)	2.390(−03)	2.389(−03)	2.387(−03)
512	2.216(−02)	1.076(−03)	1.202(−03)	1.201(−03)	1.199(−03)
1024	2.157(−02)	4.781(−04)	6.033(−04)	6.030(−04)	6.007(−04)
2048	2.164(−02)	1.875(−04)	3.027(−04)	3.023(−04)	3.025(−04)
4096	2.167(−02)	4.138(−05)	1.518(−04)	1.514(−04)	1.515(−04)
8192	2.160(−02)	3.423(−05)	7.619(−05)	7.582(−05)	7.585(−05)
16384	2.156(−02)	7.211(−05)	3.831(−05)	3.794(−05)	3.797(−05)
32768	2.157(−02)	9.049(−05)	1.935(−05)	1.898(−05)	1.898(−05)
65536	2.157(−02)	9.969(−05)	9.866(−06)	9.497(−06)	9.497(−06)

$N_t \downarrow \mid L \rightarrow$	8	32	128	512	2048
32	1.000(+00)	3.000(+00)	1.200(+01)	4.700(+01)	2.350(+02)
64	1.000(+00)	4.000(+00)	1.200(+01)	4.900(+01)	2.150(+02)
128	1.000(+00)	3.000(+00)	1.100(+01)	4.800(+01)	2.150(+02)
256	2.000(+00)	4.000(+00)	1.200(+01)	5.100(+01)	2.240(+02)
512	1.000(+00)	3.000(+00)	1.300(+01)	5.000(+01)	2.270(+02)
1024	1.000(+00)	4.000(+00)	1.400(+01)	6.500(+01)	2.550(+02)
2048	2.000(+00)	5.000(+00)	1.600(+01)	6.700(+01)	2.900(+02)
4096	2.000(+00)	6.000(+00)	2.100(+01)	8.500(+01)	3.930(+02)
8192	6.000(+00)	1.100(+01)	3.400(+01)	1.430(+02)	5.200(+02)
16384	1.600(+01)	2.500(+01)	6.300(+01)	2.170(+02)	8.360(+02)
32768	6.200(+01)	8.100(+01)	1.390(+02)	4.350(+02)	1.518(+03)
65536	2.200(+02)	2.630(+02)	3.980(+02)	9.070(+02)	3.044(+03)

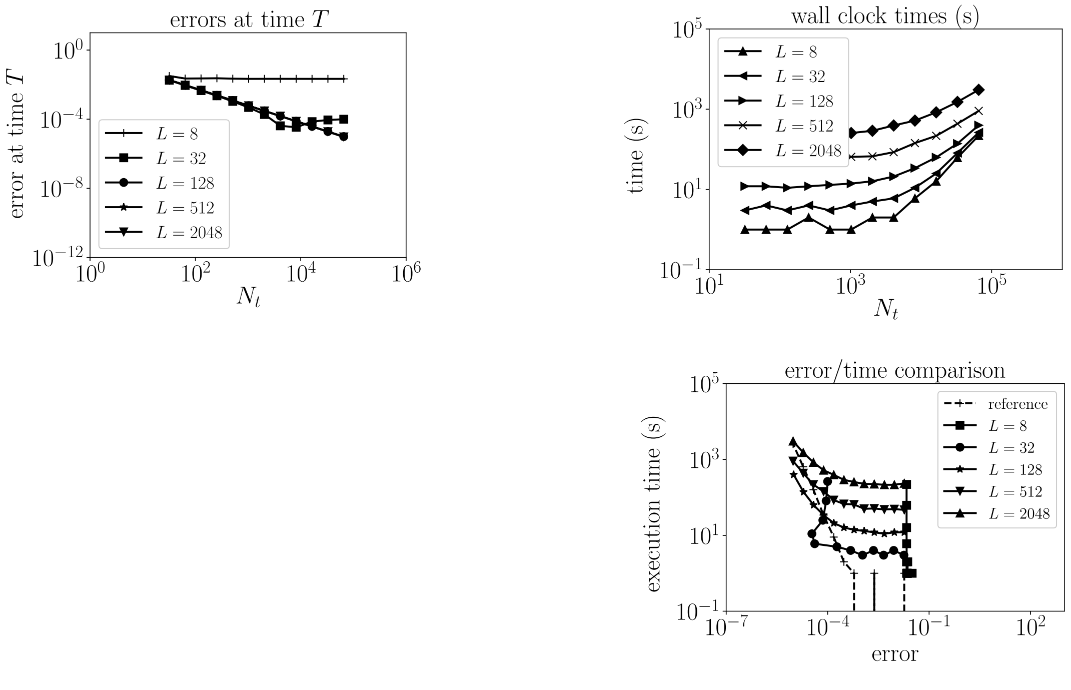


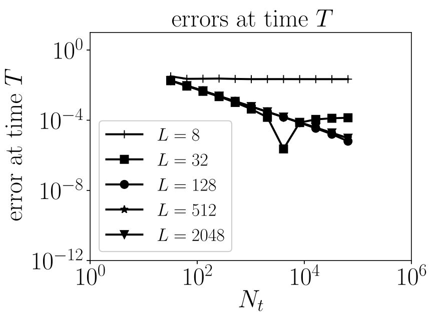
Fig. 5. Tabulated and graphical errors and timings for the product-Rectangle-Fourier method with $T = 10$, $\alpha = -0.5$, $T_1 = 0.5$ and a Hermite smoothness of $m = 5$.

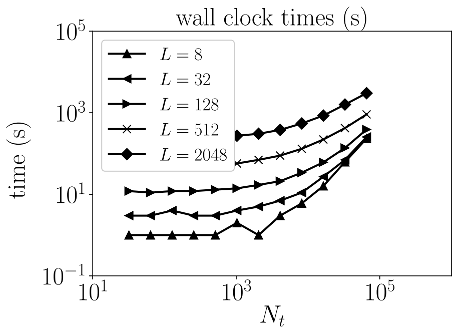
6. Discussion and conclusions

This short note has demonstrated that to within and beyond engineering accuracy a weakly singular Volterra convolution kernel can be replaced over much of the temporal domain $[0, T]$ by a Fourier series proxy. This proxy has storage and operation count requirements that are bounded independently of N_t. The algorithm can be used to

$N_t \downarrow$ | $L \rightarrow$	8	32	128	512	2048
32	3.180(−02)	1.787(−02)	1.841(−02)	1.841(−02)	1.841(−02)
64	2.285(−02)	8.833(−03)	9.369(−03)	9.369(−03)	9.370(−03)
128	2.323(−02)	4.433(−03)	4.740(−03)	4.741(−03)	4.772(−03)
256	2.374(−02)	2.218(−03)	2.387(−03)	2.389(−03)	2.387(−03)
512	2.256(−02)	1.030(−03)	1.199(−03)	1.201(−03)	1.199(−03)
1024	2.197(−02)	4.322(−04)	6.002(−04)	6.030(−04)	6.007(−04)
2048	2.204(−02)	1.462(−04)	2.993(−04)	3.023(−04)	3.025(−04)
4096	2.208(−02)	2.315(−06)	1.483(−04)	1.514(−04)	1.515(−04)
8192	2.200(−02)	7.330(−05)	7.270(−05)	7.582(−05)	7.585(−05)
16384	2.197(−02)	1.112(−04)	3.482(−05)	3.794(−05)	3.797(−05)
32768	2.197(−02)	1.293(−04)	1.585(−05)	1.898(−05)	1.898(−05)
65536	2.197(−02)	1.383(−04)	6.362(−06)	9.497(−06)	9.497(−06)

$N_t \downarrow$ | $L \rightarrow$	8	32	128	512	2048
32	1.000(+00)	3.000(+00)	1.200(+01)	4.800(+01)	2.340(+02)
64	1.000(+00)	3.000(+00)	1.100(+01)	5.100(+01)	2.290(+02)
128	1.000(+00)	4.000(+00)	1.200(+01)	5.500(+01)	2.340(+02)
256	1.000(+00)	3.000(+00)	1.200(+01)	5.000(+01)	2.370(+02)
512	1.000(+00)	3.000(+00)	1.300(+01)	5.200(+01)	2.390(+02)
1024	2.000(+00)	4.000(+00)	1.400(+01)	5.700(+01)	2.730(+02)
2048	1.000(+00)	5.000(+00)	1.700(+01)	7.000(+01)	3.050(+02)
4096	3.000(+00)	7.000(+00)	2.100(+01)	8.900(+01)	3.800(+02)
8192	6.000(+00)	1.100(+01)	3.400(+01)	1.310(+02)	5.460(+02)
16384	1.600(+01)	2.700(+01)	6.100(+01)	2.210(+02)	8.530(+02)
32768	6.200(+01)	7.000(+01)	1.370(+02)	4.210(+02)	1.591(+03)
65536	2.330(+02)	2.550(+02)	3.890(+02)	9.110(+02)	3.020(+03)





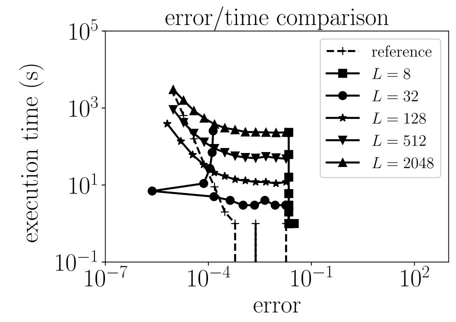


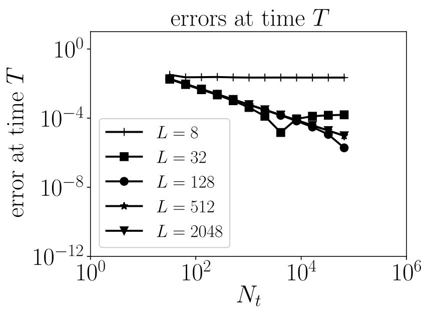
Fig. 6. Tabulated and graphical errors and timings for the product-Rectangle-Fourier method with $T = 10$, $\alpha = -0.5$, $T_1 = 0.5$ and a Hermite smoothness of $m = 10$.

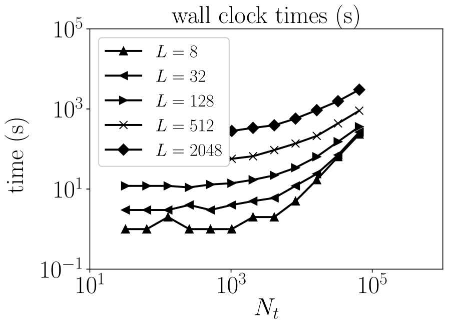
mitigate the $O(N_t)$ storage and $O(N_t^2)$ operation count requirement of a naïve quadrature approximation and yet retain the advantage of allowing variable time stepping.

In the setting described earlier, with $N_1 = \xi N_t$, the operation count becomes $O(N_1^2)$ to reach T_1, and thereafter can be estimated as $O\big((N_1 + L)(N_t - N_1)\big)$ to compute from T_1 to T. The total is therefore reduced (assuming

$N_t \downarrow \mid L \rightarrow$	8	32	128	512	2048
32	3.191(−02)	1.781(−02)	1.841(−02)	1.841(−02)	1.841(−02)
64	2.296(−02)	8.775(−03)	9.370(−03)	9.369(−03)	9.370(−03)
128	2.336(−02)	4.398(−03)	4.740(−03)	4.741(−03)	4.760(−03)
256	2.389(−02)	2.199(−03)	2.383(−03)	2.389(−03)	2.387(−03)
512	2.271(−02)	1.011(−03)	1.195(−03)	1.201(−03)	1.199(−03)
1024	2.212(−02)	4.123(−04)	5.962(−04)	6.030(−04)	6.007(−04)
2048	2.219(−02)	1.282(−04)	2.950(−04)	3.023(−04)	3.025(−04)
4096	2.223(−02)	1.471(−05)	1.439(−04)	1.514(−04)	1.515(−04)
8192	2.216(−02)	9.032(−05)	6.826(−05)	7.582(−05)	7.585(−05)
16384	2.212(−02)	1.282(−04)	3.039(−05)	3.794(−05)	3.797(−05)
32768	2.212(−02)	1.462(−04)	1.140(−05)	1.898(−05)	1.898(−05)
65536	2.213(−02)	1.552(−04)	1.903(−06)	9.497(−06)	9.497(−06)

$N_t \downarrow \mid L \rightarrow$	8	32	128	512	2048
32	1.000(+00)	3.000(+00)	1.200(+01)	5.700(+01)	2.500(+02)
64	1.000(+00)	3.000(+00)	1.200(+01)	4.800(+01)	2.530(+02)
128	2.000(+00)	3.000(+00)	1.200(+01)	4.800(+01)	2.690(+02)
256	1.000(+00)	4.000(+00)	1.100(+01)	5.000(+01)	2.460(+02)
512	1.000(+00)	3.000(+00)	1.300(+01)	5.800(+01)	2.580(+02)
1024	1.000(+00)	4.000(+00)	1.400(+01)	5.700(+01)	2.820(+02)
2048	2.000(+00)	5.000(+00)	1.700(+01)	6.600(+01)	3.370(+02)
4096	2.000(+00)	6.000(+00)	2.200(+01)	9.400(+01)	3.910(+02)
8192	5.000(+00)	1.200(+01)	3.400(+01)	1.370(+02)	5.740(+02)
16384	1.700(+01)	2.400(+01)	6.400(+01)	2.120(+02)	9.330(+02)
32768	6.400(+01)	7.100(+01)	1.520(+02)	4.350(+02)	1.564(+03)
65536	2.340(+02)	2.640(+02)	3.610(+02)	9.040(+02)	3.027(+03)





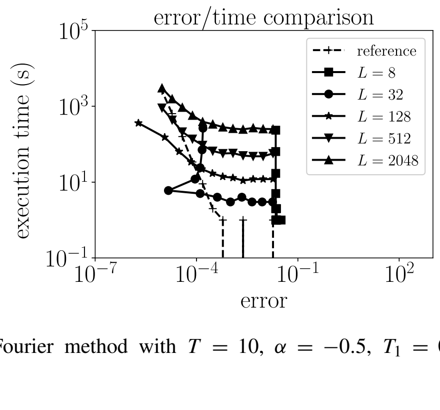


Fig. 7. Tabulated and graphical errors and timings for the product-Rectangle-Fourier method with $T = 10$, $\alpha = -0.5$, $T_1 = 0.5$ and a Hermite smoothness of $m = 15$.

$\xi \ll 1$ and $L \leqslant \xi N_t$) from $O(N_t^2)$ to $O\big((\xi N_t + L)(1-\xi)N_t\big) = O\big(\xi N_t^2\big)$, while the storage requirement is reduced from $O(N_t)$ to $O(\xi N_t)$. These are asymptotic estimates, of course, and the assumption that $L \leqslant \xi N_t$ as $N_t \to \infty$ is more than justified by the numerical results presented in Section 5.

Indeed, those results show that comparable accuracy can be obtained with the proxy with a much lower storage demand, but that there is a break-even point prior to which nothing is gained. To see this, recall that the exact

$N_t \downarrow \mid L \rightarrow$	8	32	128	512	2048
32	5.422(−02)	1.788(−02)	1.841(−02)	1.841(−02)	1.841(−02)
64	5.918(−02)	1.570(−02)	9.374(−03)	9.369(−03)	9.377(−03)
128	5.453(−02)	1.109(−02)	4.745(−03)	4.741(−03)	4.748(−03)
256	5.399(−02)	1.432(−02)	2.368(−03)	2.390(−03)	2.386(−03)
512	5.279(−02)	1.314(−02)	1.180(−03)	1.201(−03)	1.198(−03)
1024	5.235(−02)	1.425(−02)	5.328(−04)	6.031(−04)	5.968(−04)
2048	5.208(−02)	1.484(−02)	1.967(−04)	3.023(−04)	3.045(−04)
4096	5.192(−02)	1.514(−02)	2.576(−05)	1.513(−04)	1.478(−04)
8192	5.184(−02)	1.507(−02)	4.987(−05)	7.572(−05)	7.114(−05)
16384	5.180(−02)	1.503(−02)	8.774(−05)	3.785(−05)	3.270(−05)
32768	5.179(−02)	1.501(−02)	1.067(−04)	1.889(−05)	1.346(−05)
65536	5.178(−02)	1.500(−02)	1.162(−04)	9.399(−06)	3.818(−06)

$N_t \downarrow \mid L \rightarrow$	8	32	128	512	2048
32	1.000(+00)	4.000(+00)	1.100(+01)	4.900(+01)	1.960(+02)
64	1.000(+00)	3.000(+00)	1.100(+01)	4.500(+01)	2.040(+02)
128	2.000(+00)	4.000(+00)	1.300(+01)	4.700(+01)	2.100(+02)
256	1.000(+00)	3.000(+00)	1.200(+01)	4.900(+01)	2.070(+02)
512	1.000(+00)	4.000(+00)	1.300(+01)	5.200(+01)	2.150(+02)
1024	1.000(+00)	4.000(+00)	1.400(+01)	5.600(+01)	2.340(+02)
2048	2.000(+00)	4.000(+00)	1.700(+01)	6.600(+01)	2.830(+02)
4096	2.000(+00)	6.000(+00)	2.400(+01)	9.300(+01)	3.740(+02)
8192	3.000(+00)	9.000(+00)	3.200(+01)	1.260(+02)	5.240(+02)
16384	6.000(+00)	1.600(+01)	5.800(+01)	2.190(+02)	8.900(+02)
32768	1.600(+01)	3.500(+01)	1.020(+02)	3.900(+02)	1.573(+03)
65536	5.600(+01)	8.700(+01)	2.170(+02)	7.680(+02)	2.907(+03)

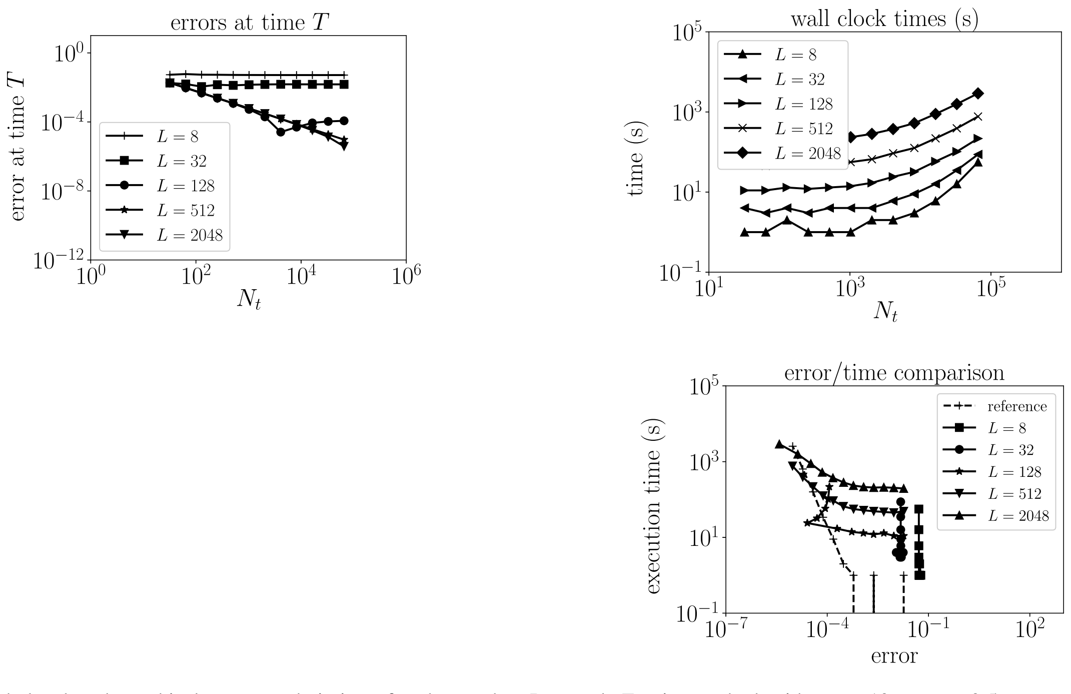
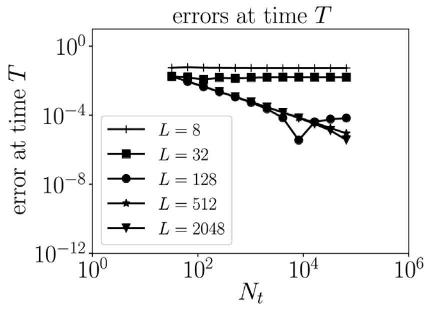


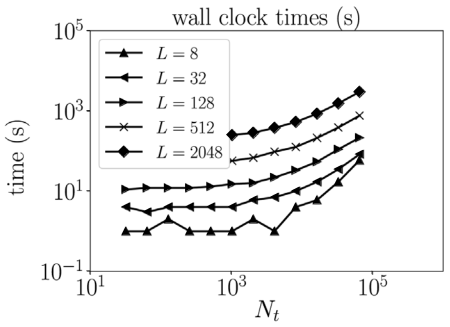
Fig. 8. Tabulated and graphical errors and timings for the product-Rectangle-Fourier method with $T = 10$, $\alpha = -0.5$, $T_1 = 0.1$ and a Hermite smoothness of $m = 1$.

solution for our numerical results is $u(t) = t^{-\alpha}$, for which $\|u\|_{L_\infty(0,T)} = \sqrt{10} \approx 3.16$. Then, for example, from Figs. 3 and 4 we see that a relative error of around 0.1% (i.e. $0.004741/3.16 \approx 0.1\%$) in the naïve rule (Fig. 3) requires $N_t = 128$ and negligible time while to reach a comparable relative error of $0.00475/3.16 \approx 0.1\%$ for the $m = 1$ proxy (Fig. 4) we still need $N_t = 128$, with $L = 32$, but with 4 seconds. The extra time is associated with the Fourier coefficient set up and may also be spent recursing the proxy as in (15). The longer execution time is

$N_t \downarrow$ | $L \rightarrow$	8	32	128	512	2048
32	5.663(−02)	1.797(−02)	1.842(−02)	1.841(−02)	1.841(−02)
64	6.165(−02)	1.637(−02)	9.161(−03)	9.369(−03)	9.377(−03)
128	5.700(−02)	1.176(−02)	4.532(−03)	4.741(−03)	4.748(−03)
256	5.632(−02)	1.533(−02)	2.350(−03)	2.390(−03)	2.386(−03)
512	5.512(−02)	1.414(−02)	1.162(−03)	1.202(−03)	1.198(−03)
1024	5.463(−02)	1.533(−02)	5.601(−04)	6.030(−04)	5.968(−04)
2048	5.432(−02)	1.596(−02)	2.381(−04)	3.020(−04)	3.045(−04)
4096	5.415(−02)	1.628(−02)	7.194(−05)	1.511(−04)	1.478(−04)
8192	5.407(−02)	1.621(−02)	3.682(−06)	7.545(−05)	7.113(−05)
16384	5.403(−02)	1.617(−02)	4.156(−05)	3.757(−05)	3.270(−05)
32768	5.401(−02)	1.615(−02)	6.052(−05)	1.861(−05)	1.345(−05)
65536	5.400(−02)	1.614(−02)	7.001(−05)	9.126(−06)	3.816(−06)

$N_t \downarrow$ | $L \rightarrow$	8	32	128	512	2048
32	1.000(+00)	4.000(+00)	1.100(+01)	4.800(+01)	2.210(+02)
64	1.000(+00)	3.000(+00)	1.200(+01)	4.800(+01)	2.130(+02)
128	2.000(+00)	4.000(+00)	1.200(+01)	5.300(+01)	2.080(+02)
256	1.000(+00)	4.000(+00)	1.200(+01)	5.600(+01)	2.070(+02)
512	1.000(+00)	4.000(+00)	1.300(+01)	5.000(+01)	2.180(+02)
1024	1.000(+00)	4.000(+00)	1.500(+01)	5.700(+01)	2.490(+02)
2048	2.000(+00)	6.000(+00)	1.600(+01)	6.800(+01)	2.860(+02)
4096	1.000(+00)	7.000(+00)	2.200(+01)	9.600(+01)	3.780(+02)
8192	4.000(+00)	1.000(+01)	3.300(+01)	1.260(+02)	5.320(+02)
16384	6.000(+00)	1.700(+01)	5.400(+01)	2.120(+02)	8.540(+02)
32768	1.700(+01)	3.500(+01)	1.100(+02)	3.870(+02)	1.545(+03)
65536	5.900(+01)	8.300(+01)	2.150(+02)	7.650(+02)	2.990(+03)





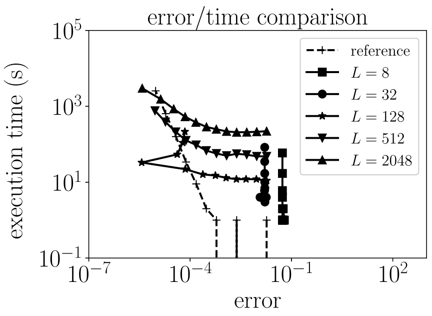


Fig. 9. Tabulated and graphical errors and timings for the product-Rectangle-Fourier method with $T = 10$, $\alpha = -0.5$, $T_1 = 0.1$ and a Hermite smoothness of $m = 5$.

however offset by the drop in storage: $N_t = 128$ values for the naïve rule compared to $N_1 + L + 1 = 7 + 32 + 1 = 40$ for the proxy. On the other hand, for errors around 10^{-5} we see an order of magnitude difference in speed, and the storage saving reduces from 65536 to $N_1 + L + 1 = 3277 + 32 + 1 = 3294$ for $m = 1$.

For $T_1 = 0.5$ the convergence pattern does not seem to be sensitive to m, as shown by comparing Fig. 4 to Figs. 5–7, although there does seem to be some boundary near $L = 32$ that separates out the regions where convergence

$N_t \downarrow \mid L \rightarrow$	8	32	128	512	2048
32	5.422(−02)	1.788(−02)	1.841(−02)	1.841(−02)	1.841(−02)
64	5.918(−02)	1.570(−02)	9.374(−03)	9.369(−03)	9.377(−03)
128	5.595(−02)	2.403(−02)	4.472(−03)	4.741(−03)	4.725(−03)
256	5.357(−02)	2.169(−02)	2.121(−03)	2.390(−03)	2.374(−03)
512	5.051(−02)	2.441(−02)	1.526(−03)	1.200(−03)	1.201(−03)
1024	4.990(−02)	2.381(−02)	9.277(−04)	6.014(−04)	6.030(−04)
2048	4.885(−02)	2.442(−02)	1.058(−03)	3.022(−04)	3.023(−04)
4096	4.827(−02)	2.471(−02)	1.186(−03)	1.528(−04)	1.374(−04)
8192	4.796(−02)	2.485(−02)	1.267(−03)	7.790(−05)	7.583(−05)
16384	4.792(−02)	2.481(−02)	1.229(−03)	4.003(−05)	3.796(−05)
32768	4.790(−02)	2.479(−02)	1.211(−03)	2.107(−05)	1.900(−05)
65536	4.789(−02)	2.478(−02)	1.201(−03)	1.158(−05)	9.509(−06)

$N_t \downarrow \mid L \rightarrow$	8	32	128	512	2048
32	1.000(+00)	4.000(+00)	1.200(+01)	4.500(+01)	1.970(+02)
64	1.000(+00)	3.000(+00)	1.200(+01)	4.700(+01)	1.920(+02)
128	2.000(+00)	3.000(+00)	1.100(+01)	5.100(+01)	2.070(+02)
256	1.000(+00)	3.000(+00)	1.200(+01)	4.800(+01)	2.300(+02)
512	1.000(+00)	4.000(+00)	1.300(+01)	4.800(+01)	2.080(+02)
1024	1.000(+00)	4.000(+00)	1.400(+01)	5.800(+01)	2.340(+02)
2048	2.000(+00)	4.000(+00)	1.700(+01)	6.600(+01)	2.680(+02)
4096	1.000(+00)	6.000(+00)	2.200(+01)	9.100(+01)	3.710(+02)
8192	3.000(+00)	9.000(+00)	3.200(+01)	1.280(+02)	5.250(+02)
16384	6.000(+00)	1.500(+01)	5.500(+01)	2.160(+02)	9.040(+02)
32768	1.200(+01)	3.300(+01)	9.700(+01)	3.840(+02)	1.511(+03)
65536	3.500(+01)	7.300(+01)	2.000(+02)	7.170(+02)	2.909(+03)

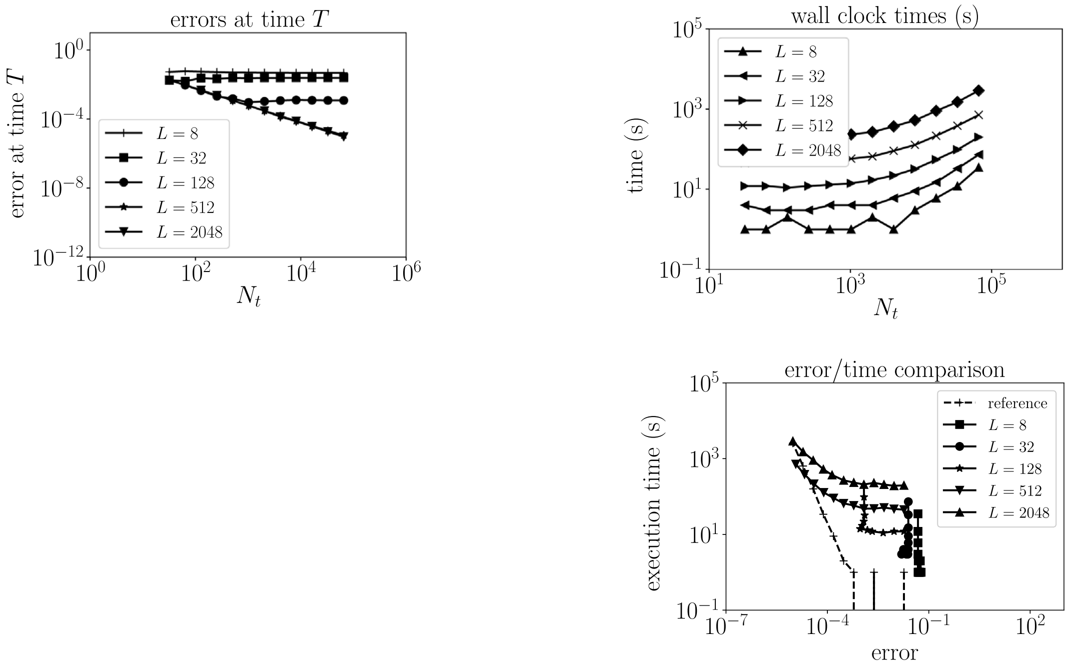
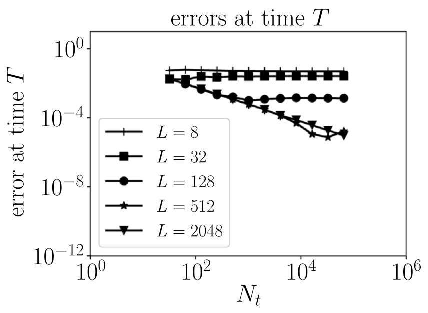


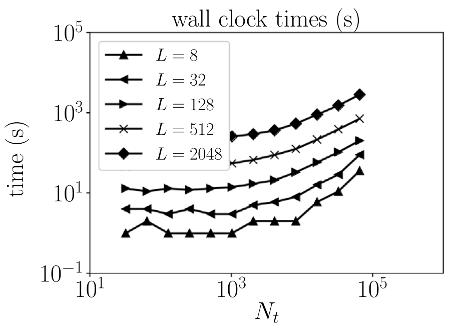
Fig. 10. Tabulated and graphical errors and timings for the product-Rectangle-Fourier method with $T = 10$, $\alpha = -0.5$, $T_1 = 0.05$ and a Hermite smoothness of $m = 1$.

is stalled by the lack of Fourier series accuracy. Then, as we push the method and reduce to $T_1 = 0.1$ we infer from Figs. 8 and 9, that this boundary moves up to around $L = 128$. Further, for $T_1 = 0.05$, in Figs. 10 and 11, we can just see the stalling initiating for $L = 512$ but only for $m = 5$. It is not clear why this deterioration should happen for larger m – it could be due, for example, to instabilities in computing the Hermite splines, or inaccuracy in numerical quadrature – and we leave the observation here as an open question for further investigation.

$N_t \downarrow$ | $L \rightarrow$	8	32	128	512	2048
32	5.663(−02)	1.797(−02)	1.842(−02)	1.841(−02)	1.841(−02)
64	6.165(−02)	1.637(−02)	9.161(−03)	9.369(−03)	9.377(−03)
128	5.801(−02)	2.530(−02)	4.518(−03)	4.745(−03)	4.725(−03)
256	5.562(−02)	2.296(−02)	2.167(−03)	2.394(−03)	2.374(−03)
512	5.236(−02)	2.574(−02)	1.624(−03)	1.210(−03)	1.201(−03)
1024	5.175(−02)	2.514(−02)	1.026(−03)	6.119(−04)	6.030(−04)
2048	5.064(−02)	2.576(−02)	1.198(−03)	3.004(−04)	3.023(−04)
4096	5.003(−02)	2.604(−02)	1.351(−03)	1.348(−04)	1.374(−04)
8192	4.970(−02)	2.618(−02)	1.446(−03)	4.931(−05)	7.582(−05)
16384	4.966(−02)	2.614(−02)	1.409(−03)	1.144(−05)	3.795(−05)
32768	4.965(−02)	2.612(−02)	1.390(−03)	7.522(−06)	1.899(−05)
65536	4.964(−02)	2.611(−02)	1.380(−03)	1.701(−05)	9.498(−06)

$N_t \downarrow$ | $L \rightarrow$	8	32	128	512	2048
32	1.000(+00)	4.000(+00)	1.300(+01)	4.500(+01)	2.160(+02)
64	2.000(+00)	4.000(+00)	1.100(+01)	4.700(+01)	1.970(+02)
128	1.000(+00)	3.000(+00)	1.300(+01)	5.300(+01)	2.020(+02)
256	1.000(+00)	4.000(+00)	1.200(+01)	5.300(+01)	2.100(+02)
512	1.000(+00)	3.000(+00)	1.300(+01)	5.300(+01)	2.180(+02)
1024	1.000(+00)	3.000(+00)	1.400(+01)	5.500(+01)	2.580(+02)
2048	2.000(+00)	5.000(+00)	1.700(+01)	6.700(+01)	2.940(+02)
4096	2.000(+00)	6.000(+00)	2.100(+01)	8.900(+01)	3.670(+02)
8192	2.000(+00)	8.000(+00)	3.300(+01)	1.260(+02)	5.370(+02)
16384	6.000(+00)	1.600(+01)	5.800(+01)	2.150(+02)	9.020(+02)
32768	1.100(+01)	2.900(+01)	1.050(+02)	3.890(+02)	1.528(+03)
65536	3.600(+01)	9.000(+01)	2.020(+02)	7.130(+02)	2.837(+03)





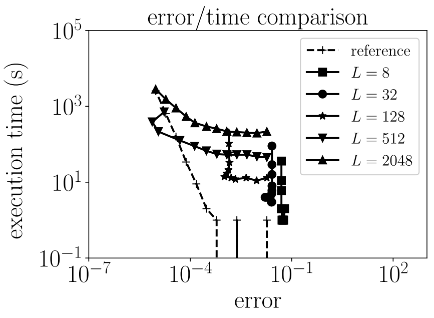


Fig. 11. Tabulated and graphical errors and timings for the product-Rectangle-Fourier method with $T = 10$, $\alpha = -0.5$, $T_1 = 0.05$ and a Hermite smoothness of $m = 5$.

Furthermore, and possibly related to this, our results in Figs. 4–7 indicate that convergence begins to stall for $L = 32$ for all of $m = 1, 5, 10, 15$. We expected that as the proxy increased in smoothness so its convergence to its subject would increase resulting in us needing smaller values for L. This seems not to be happening and is left here as another open question worthy of follow-up investigation.

The error/time comparison plots have been included for interest. These indicate that there would in all likelihood be a break-even point for error and time. On the other hand, the timings shown here are probably not that relevant to the PDE applications that have motivated this study. This code was written in pure python, whereas much faster code implementations would be chosen for high fidelity PDE solvers.

Also, in practice we should take account of the time overhead in pre-computing the Fourier series and Hermite splines for the proxy, but – again – in the context of a large scale PDE solver, this is likely to be a negligible one-off cost (at least for linear problems with piecewise constant material data). It is also likely that in this context it is the storage need that will obviate operation count concerns. This is because the CPU cycles associated with the Volterra history summation could be easily parallelized, and the accompanying arithmetic demand may not be out of line with the linear algebra arithmetic associated with the PDE approximation at each time step (particularly for implicit solves).

We close with some speculative suggestions and directions for further development.

1. As mentioned above, the usefulness of the proxy rests on the update formula (15), and that can also be used with variable time steps. In that case though we would need to replace N_1 with a margin, $T - T_1$ as in [16,17]. Variable time steps make adaptive time stepping possible.
2. If the storage problem is the key difficulty in working with large scale fractional calculus models then it would be useful to explore the effect of using low precision data types for the history. Single and even half precision data types are currently enjoying a re-introduction into modern computing tools due to their usefulness in data science applications.
3. The $O(N_1^2)$ and $O(N_1)$ burdens required in our method to deal with the rectangle rule integration near the weak singularity could be further mitigated by using, say, the product trapezoidal rule up to $t_n - \Delta t$, and the product rectangle rule on the latest time interval. To keep the errors comparable the trapezoidal rule could have a larger time step of $\Delta' t$ to retain consistency with the first order rectangle rule error. This is essentially the idea put forward in [9], where the choice $\Delta' t = T/N_1' \sim \sqrt{\Delta t}$ would be made, reducing N_1 to $N_1' = \sqrt{T_1 N_1}$. If, as earlier, $N_1 = \xi N_t$ and $T_1 = \xi T$ then $N_1' = \xi\sqrt{T N_t}$ and so with – for example – $N_t = 1000$ and $T = 10$ we get with $\xi = 5\%$ that $N_1' = 5$, whereas $N_1 = 50$. The N_1 burden for our approach would therefore reduce dramatically. The higher order Simpson's and Boole's Newton–Cotes rules could also be considered, and the modifications required to the algorithm earlier, and the code, are trivial.
4. On the other hand, as we discussed earlier, in [20] a local expansion was used to approximate this local integral. Such a method could also be used here to remove any need for numerical quadrature.
5. We also note that these weakly singular kernels will be obtained from fitting constitutive material data, and that if the data favour $\bar{t} > 0$ in (10), then there is no *a priori* need for the $[0, T_1]$ margin employed here. For example, we can shift φ to the right by $\bar{t}$ and use $T_1 = \bar{t}$ and $T_x = T + 2T_1$ in the calculations outlined earlier to get the proxy. Once we have the proxy we can shift it left by $\bar{t}$ and then the Fourier series applies over the entire interval $[0, T]$, thus removing the entire storage and operation count burden associated with the naïve quadrature. The scheme is then particularly easy and advantageous to implement and can use the same routines as used for Prony series type recursion (e.g. [22,23]). We have not tried that here because there is no exact solution available and so no useful numerical data could be presented.
6. Lastly, we note that an error analysis of this method would comprise three independent contributions: a Taylor series type of analysis related to the naïve quadrature, a projection analysis related to the Fourier series, and, another quadrature-error analysis related to determining the Fourier coefficients. We could expect such an analysis to contribute to solving the open questions posed earlier.

On the last point, to estimate the Fourier coefficients we used the `quad` routine available from `scipy` in python with `from scipy.integrate import quad`. We are aware that this is yet another area where the algorithm and results presented here could be improved, but as this study is presented in the spirit of a discussion piece we have not been able yet to drill down to the fine details. We have aimed for simplicity and economy of presentation, and included just enough numerical results to give evidence that the method works. There is certainly scope for improved computation of the Fourier coefficients, and this again may be of relevance to the open questions identified earlier.

In summary, our method can be used to mitigate the $O(N_t)$ storage and $O(N_t^2)$ storage issues using an easily implemented proxy. It is, at least as presented here, not a panacea though. These burdens are reduced, not eliminated, and there are performance issues to investigate. Nevertheless, the method shows enough promise to justify further investigations.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The manuscript contains details of how to obtain the source codes used to generate the results.

References

[1] J.M. Golden, G.A.C. Graham, Boundary Value Problems in Linear Viscoelasticity, Springer-Verlag, 1988.

[2] A. Bonfanti, J.L. Kaplan, G. Charras, A. Kabla, Fractional viscoelastic models for power-law materials, Soft Matter 16 (2020) 6002–6020.

[3] Mauro Fabrizio, Angelo Morro, Mathematical Problems in Linear Viscoelasticity, in: Studies in applied mathematics, vol. 12, SIAM, Philadelphia, 1992.

[4] Mauro Fabrizio, Angelo Morro, Electromagnetism of Continuous Media: Mathematical Modelling and Applications, Oxford University Press, 2003.

[5] Peter G. Petropoulos, On the time-domain response of Cole-Cole dielectrics, IEEE Trans. Antennas and Propagation 53 (2005) 3741–3746.

[6] Jichun Li, Yunqing Huang, Yanping Lin, Developing finite element methods for Maxwell's equations in a cole-cole dispersive medium, SIAM J. Sci. Comput. 33 (2011) 3153–3174, http://dx.doi.org/10.1137/110827624.

[7] Simon Shaw, J.R. Whiteman, Numerical solution of linear quasistatic hereditary viscoelasticity problems, SIAM J. Numer. Anal 38 (1) (2000) 80–97.

[8] Y. Lin, V. Thomée, L.B. Wahlbin, Ritz-Volterra projections to finite-element spaces and applications to integrodifferential and related equations, SIAM J. Numer. Anal. 28 (1991) 1047–1070.

[9] I.H. Sloan, V. Thomée, Time discretization of an integro-differential equation of parabolic type, SIAM J. Numer. Anal. 23 (1986) 1052–1061.

[10] E.G. Yanik, G. Fairweather, Finite element methods for parabolic and hyperbolic partial integro-differential equations, Nonlinear Anal. 12 (1988) 785–809.

[11] Chen Chuanmiao, Shih Tsimin, Finite Element Methods for Integrodifferential Equations, in: Series on Applied Mathematics, vol. 9, World Scientific Publishing Co. Pte. Ltd., 1998.

[12] A.R. Johnson, Modeling viscoelastic materials using internal variables, Shock Vib. Dig. 31 (1999) 91–100.

[13] Béatrice Rivière, Simon Shaw, Mary F. Wheeler, J.R. Whiteman, Discontinuous Galerkin finite element methods for linear elasticity and quasistatic linear viscoelasticity, Numer. Math. 95 (2003) 347–376.

[14] Béatrice Rivière, Simon Shaw, J.R. Whiteman, Discontinuous Galerkin finite element methods for dynamic linear solid viscoelasticity problems, Numer. Methods Partial Differential Equations 23 (2007) 1149–1166, See also report 05/7 at www.brunel.ac.uk/bicom.

[15] W.N. Findley, J.S. Lai, K. Onaran, Creep and Relaxation of Nonlinear Viscoelastic Materials (with an Introduction to Linear Viscoelasticity), Dover Publications Inc., New York, 1989.

[16] K. Adolfsson, M. Enelund, S. Larsson, Adaptive discretization of fractional order viscoelasticity using sparse time history, Comput. Methods Appl. Mech. Engrg. 193 (2004) 4567–4590.

[17] K. Adolfsson, M. Enelund, S. Larsson, Space-time discretization of an integro-differential equation modeling quasi-static fractional-order viscoelasticity, J. Vib. Control 14(9–10) (2008) 1631–1649.

[18] Achim Schädle, María López-Fernández, Christian Lubich, Fast and oblivious convolution quadrature, SIAM J. Sci. Comput. 28 (2006) 421–438.

[19] William McLean, Fast summation by interval clustering for an evolution equation with memory, SIAM J. Sci. Comput. 34 (2012) A3039–A3056.

[20] Matthew F. Causley, Peter G. Petropoulos, Shidong Jiang, Incorporating the Havriliak–Negami dielectric model in the FD-TD method, J. Comput. Phys. 230 (2011) 3884–3899, http://dx.doi.org/10.1016/j.jcp.2011.02.012.

[21] Peter Linz, Analytical and Numerical Methods for Volterra Equations, SIAM, Philadelphia, 1985.

[22] Simon Shaw, M.K. Warby, J.R. Whiteman, C. Dawson, M.F. Wheeler, Numerical techniques for the treatment of quasistatic viscoelastic stress problems in linear isotropic solids, Comput. Methods Appl. Mech. Engrg. 118 (1994) 211–237.

[23] A.R. Johnson, A. Tessler, M. Dambach, Dynamics of thick viscoelastic beams, Trans. ASME, J. Eng. Mater. Technol. 119 (1997) 273–278.