Numerical Algorithms
https://doi.org/10.1007/s11075-024-01884-y

ORIGINAL PAPER

Numerical solution of FDE-IVPs by using fractional HBVMs: the `fhbvm` code

Luigi Brugnano[1] · **Gianmarco Gurioli**[1] · **Felice Iavernaro**[2]

Received: 7 March 2024 / Accepted: 8 July 2024

Abstract
In this paper we describe the efficient numerical implementation of *Fractional HBVMs*, a class of methods recently introduced for solving systems of fractional differential equations. The reported arguments are implemented in the Matlab© code `fhbvm`, which is made available on the web. An extensive experimentation of the code is reported, to give evidence of its effectiveness.

Keywords Fractional differential equations · Fractional integrals · Caputo derivative · Jacobi polynomials · Fractional hamiltonian boundary value methods · FHBVMs

Mathematics Subject Classification (2010) 34A08 · 65R20 · 65-04

1 Introduction

Fractional differential equations have become a common description tool across a variety of applications (see, e.g., the classical references [23, 36] for an introduction). For this reason, their numerical solution has been the subject of many researches (see, e.g. [1, 24, 25, 34, 35, 37, 38]), with the development of corresponding software (see, e.g., [21, 27, 28]). In this context, the present contribution is addressed for solving *initial value problems for fractional differential equations (FDE-IVPs)* in the form

$$y^{(\alpha)}(t) = f(y(t)), \qquad t \in [0, T], \qquad y(0) = y_0 \in \mathbb{R}^m, \tag{1}$$

✉ Luigi Brugnano
luigi.brugnano@unifi.it

Gianmarco Gurioli
gianmarco.gurioli@unifi.it

Felice Iavernaro
felice.iavernaro@uniba.it

[1] Dipartimento di Matematica e Informatica "U. Dini", Università di Firenze, Florence, Italy

[2] Dipartimento di Matematica, Università di Bari Aldo Moro, Bari, Italy

Published online: 16 July 2024

where, for the sake of brevity, we have omitted the argument t for f. Here, for $\alpha \in (0,1)$, $y^{(\alpha)}(t) \equiv D^\alpha y(t)$ is the Caputo fractional derivative:[1]

$$D^\alpha g(t) = \frac{1}{\Gamma(1-\alpha)} \int_0^t (t-x)^{-\alpha} \left[\frac{\mathrm{d}}{\mathrm{d}x} g(x) \right] \mathrm{d}x, \tag{2}$$

The Riemann-Liouville integral associated to (2) is given by:

$$I^\alpha g(t) = \frac{1}{\Gamma(\alpha)} \int_0^t (t-x)^{\alpha-1} g(x) \mathrm{d}x. \tag{3}$$

Consequently, the solution of (1) can be formally written as:

$$y(t) = y_0 + I^\alpha f(y(t)) \equiv y_0 + \frac{1}{\Gamma(\alpha)} \int_0^t (t-x)^{\alpha-1} f(y(x)) \mathrm{d}x, \qquad t \in [0,T]. \tag{4}$$

The numerical method we shall consider, relies on *Fractional HBVMs (FHBVMs)*, a class of methods recently introduced in [8], as an extension of Hamiltonian Boundary Value Methods (HBVMs), special low-rank Runge-Kutta methods originally devised for Hamiltonian problems (see, e.g., [10, 11]), and later extended along several directions (see, e.g., [2, 4, 6, 8, 9, 12]), including the numerical solution of FDEs. A main feature of HBVMs is the fact that they can gain spectrally accuracy, when approximating ODE-IVPs [3, 19, 20], and such a feature has been recently extended to the FDE case [8].

With this premise, the structure of the paper is as follows: in Section 2 we recall the main facts about the numerical solution of FDE-IVPs proposed in [8]; in Section 3 we provide full implementation details for the Matlab© code `fhbvm` used in the numerical tests; in Section 4 we report an extensive experimentation of the code, providing some comparisons with another existing one; at last, a few conclusions are given in Section 5.

2 Fractional HBVMs

To begin with, in order to obtain a piecewise approximation to the solution of the problem, we consider a partition of the integration interval in the form:

$$t_n = t_{n-1} + h_n, \qquad n = 1, \dots, N, \tag{5}$$

where $h_n > 0$, $n = 1, \dots, N$, and such that

$$t_0 = 0, \qquad t_N = T \equiv \sum_{n=1}^{N} h_n. \tag{6}$$

[1] As is usual, Γ denotes the Euler gamma function, such that, for $x > 0$, $x\Gamma(x) = \Gamma(x+1)$.

Further, by setting

$$y_n(ch_n) := y(t_{n-1} + ch_n), \qquad c \in [0,1], \qquad n = 1, \ldots, N, \tag{7}$$

the restriction of the solution of (1) on the interval $[t_{n-1}, t_n]$, and taking into account (4) and (5–6), one obtains, for $t \equiv t_{n-1} + ch_n$, $c \in [0,1]$,

$$\begin{aligned}
y(t) \equiv y_n(ch_n) &= y_0 + \frac{1}{\Gamma(\alpha)} \int_0^{t_{n-1}+ch_n} (t_{n-1} + ch_n - x)^{\alpha-1} f(y(x))\mathrm{d}x \\
&= y_0 + \frac{1}{\Gamma(\alpha)} \sum_{\nu=1}^{n-1} \int_{t_{\nu-1}}^{t_\nu} (t_{n-1} + ch_n - x)^{\alpha-1} f(y(x))\mathrm{d}x \\
&\quad + \frac{1}{\Gamma(\alpha)} \int_{t_{n-1}}^{t_{n-1}+ch_n} (t_{n-1} + ch_n - x)^{\alpha-1} f(y(x))\mathrm{d}x \\
&= y_0 + \frac{1}{\Gamma(\alpha)} \sum_{\nu=1}^{n-1} \int_0^{h_\nu} (t_{n-1} - t_{\nu-1} + ch_n - x)^{\alpha-1} f(y_\nu(x))\mathrm{d}x \\
&\quad + \frac{1}{\Gamma(\alpha)} \int_0^{ch_n} (ch_n - x)^{\alpha-1} f(y_n(x))\mathrm{d}x \\
&= y_0 + \frac{1}{\Gamma(\alpha)} \sum_{\nu=1}^{n-1} h_\nu^\alpha \int_0^1 \left(\frac{t_{n-1} - t_{\nu-1}}{h_\nu} + c\frac{h_n}{h_\nu} - \tau \right)^{\alpha-1} f(y_\nu(\tau h_\nu))\mathrm{d}\tau \\
&\quad + \frac{h_n^\alpha}{\Gamma(\alpha)} \int_0^c (c-\tau)^{\alpha-1} f(y_n(\tau h_n))\mathrm{d}\tau \\
&= y_0 + \frac{1}{\Gamma(\alpha)} \sum_{\nu=1}^{n-1} h_\nu^\alpha \int_0^1 \left(\sum_{i=\nu}^{n-1} \frac{h_i}{h_\nu} + c\frac{h_n}{h_\nu} - \tau \right)^{\alpha-1} f(y_\nu(\tau h_\nu))\mathrm{d}\tau \\
&\quad + \frac{h_n^\alpha}{\Gamma(\alpha)} \int_0^c (c-\tau)^{\alpha-1} f(y_n(\tau h_n))\mathrm{d}\tau, \qquad c \in [0,1].
\end{aligned} \tag{8}$$

To make manageable the handling of the ratios $h_i/h_\nu, i = \nu, \ldots, n, \nu = 1, \ldots, n-1$, we shall hereafter consider the following choices for the mesh (5–6):

– **Graded mesh.** In order to cope with possible singularities in the derivative of the vector field at the origin, we consider the *graded mesh*

$$h_n = rh_{n-1} \equiv r^{n-1}h_1, \qquad n = 1 \ldots, N, \tag{9}$$

where $r > 1$ and $h_1 > 0$ satisfy, by virtue of (5–6),

$$h_1 \frac{r^N - 1}{r - 1} = T. \tag{10}$$

As a result, one obtains that (8) becomes:

$$\begin{aligned} y_n(ch_n) \;=\; & y_0 \;+ \\ & \frac{h_1^\alpha}{\Gamma(\alpha)} \sum_{\nu=1}^{n-1} r^{\alpha(\nu-1)} \int_0^1 \left(\frac{r^{n-\nu}-1}{r-1} + cr^{n-\nu} - \tau \right)^{\alpha-1} f(y_\nu(\tau r^{\nu-1} h_1))\mathrm{d}\tau \\ & + \frac{h_1^\alpha r^{\alpha(n-1)}}{\Gamma(\alpha)} \int_0^c (c-\tau)^{\alpha-1} f(y_n(\tau r^{n-1} h_1))\mathrm{d}\tau \\ =:\; & \phi_{n-1}^\alpha(c; h_1, r, y) \;+\; \frac{h_1^\alpha r^{\alpha(n-1)}}{\Gamma(\alpha)} \int_0^c (c-\tau)^{\alpha-1} f(y_n(\tau r^{n-1} h_1))\mathrm{d}\tau, \\ & c \in [0,1]. \end{aligned} \tag{11}$$

– **Uniform mesh.** This case is equivalent to allowing $r = 1$ in (9). Consequently, from (5–6) we derive

$$h_n \equiv h_1 := \frac{T}{N}, \quad n = 1, \ldots, N, \quad \Rightarrow \quad t_n = nh_1, \quad n = 0, \ldots, N. \tag{12}$$

As a result, (8) reads:

$$\begin{aligned} y_n(ch_n) \equiv\; & y_n(ch_1) \\ = \; & y_0 + \frac{h_1^\alpha}{\Gamma(\alpha)} \sum_{\nu=1}^{n-1} \int_0^1 (n-\nu+c-\tau)^{\alpha-1} f(y_\nu(\tau h_1))\mathrm{d}\tau \\ & + \frac{h_1^\alpha}{\Gamma(\alpha)} \int_0^c (c-\tau)^{\alpha-1} f(y_n(\tau h_1))\mathrm{d}\tau \\ \equiv \; & \phi_{n-1}^\alpha(c; h_1, 1, y) \;+\; \frac{h_1^\alpha}{\Gamma(\alpha)} \int_0^c (c-\tau)^{\alpha-1} f(y_n(\tau h_1))\mathrm{d}\tau, \\ & c \in [0,1]. \end{aligned} \tag{13}$$

Remark 1 As is clear, in order to obtain an accurate approximation of the solution, it is important to establish which kind of mesh (graded or uniform) is appropriate. Besides this, also a proper choice of the parameters h_1, r, and N in (9–10) is crucial. Both aspects will be studied in Section 3.1.

2.1 Quasi-polynomial approximation

We now discuss a piecewise quasi-polynomial approximation to the solution of (1),

$$\sigma(t) \approx y(t), \qquad t \in [0, T],$$

such that

$$\sigma_n(ch_n) \;:=\; \sigma(t_{n-1} + ch_n), \qquad c \in [0,1], \qquad n = 1, \ldots, N, \tag{14}$$

is the approximation to $y_n(ch_n)$, defined in (7). According to [8], such approximation will be derived through the following steps:

1. expansion of the vector field, in each sub-interval $[t_{n-1}, t_n]$, $n = 1, \ldots, N$, (recall (5–6)) along a suitable orthonormal polynomial basis;
2. truncation of the infinite expansion, in order to obtain a local polynomial approximation.

Let us first consider the expansion of the vector field along the orthonormal polynomial basis, w.r.t. the weight function

$$\omega(x) = \alpha(1-x)^{\alpha-1}, \qquad s.t. \qquad \int_0^1 \omega(x)\,\mathrm{d}x = 1, \tag{15}$$

resulting into a scaled and shifted family of Jacobi polynomials:[2]

$$P_j(x) := \sqrt{\frac{2j+\alpha}{\alpha}}\,\overline{P}_j^{(\alpha-1,0)}(2x-1), \qquad x \in [0,1], \qquad j = 0, 1, \ldots,$$

such that

$$\int_0^1 \omega(x) P_i(x) P_j(x)\mathrm{d}x = \delta_{ij}, \qquad i, j = 0, 1, \ldots. \tag{16}$$

In so doing, for $n = 1, \ldots, N$, one obtains:

$$f(y_n(ch_n)) = \sum_{j\geq 0} P_j(c)\gamma_j(y_n), \qquad c \in [0,1], \tag{17}$$

with (see (15))

$$\gamma_j(y_n) = \int_0^1 \omega(\tau) P_j(\tau) f(y_n(\tau h_n))\mathrm{d}\tau, \qquad j = 0, 1, \ldots. \tag{18}$$

Consequently, the FDE (1), can be rewritten as:

$$y_n^{(\alpha)}(ch_n) = \sum_{j\geq 0} P_j(c)\gamma_j(y_n), \quad c \in [0,1], \quad n = 1, \ldots, N, \qquad y_1(0) = y_0. \tag{19}$$

The approximation is derived by truncating the infinite series in (17) to a finite sum with s terms, thus replacing (19) with a series of local problems, whose vector field is a polynomial of degree s:

$$\sigma_n^{(\alpha)}(ch_n) = \sum_{j=0}^{s-1} P_j(c)\gamma_j(\sigma_n), \quad c \in [0,1], \quad n = 1, \ldots, N, \qquad \sigma_1(0) = y_0, \tag{20}$$

[2] Here, $\overline{P}_j^{(a,b)}(x)$ denotes the j-th Jacobi polynomial with parameters a and b, in $[-1, 1]$.

with $\gamma_j(\sigma_n)$ defined similarly as in (18):

$$\gamma_j(\sigma_n) = \int_0^1 \omega(\tau) P_j(\tau) f(\sigma_n(\tau h_n)) \mathrm{d}\tau, \qquad j = 0, \ldots, s-1. \tag{21}$$

As a consequence, (11) will be approximated as:

$$\begin{aligned}
\sigma_n(ch_n) \;=\; & \\
& y_0 + \frac{h_1^\alpha}{\Gamma(\alpha)} \sum_{\nu=1}^{n-1} r^{\alpha(\nu-1)} \int_0^1 \left(\frac{r^{n-\nu}-1}{r-1} + c r^{n-\nu} - \tau \right)^{\alpha-1} \sum_{j=0}^{s-1} P_j(\tau) \gamma_j(\sigma_\nu) \mathrm{d}\tau \\
& \quad + \frac{h_1^\alpha r^{\alpha(n-1)}}{\Gamma(\alpha)} \int_0^c (c-\tau)^{\alpha-1} \sum_{j=0}^{s-1} P_j(\tau) \gamma_j(\sigma_n) \mathrm{d}\tau \;=\; y_0 + \\
& h_1^\alpha \sum_{\nu=1}^{n-1} r^{\alpha(\nu-1)} \sum_{j=0}^{s-1} \left[\frac{1}{\Gamma(\alpha)} \int_0^1 \left(\frac{r^{n-\nu}-1}{r-1} + c r^{n-\nu} - \tau \right)^{\alpha-1} P_j(\tau) \mathrm{d}\tau \right] \gamma_j(\sigma_\nu) \\
& \quad + h_1^\alpha r^{\alpha(n-1)} \sum_{j=0}^{s-1} \left[\frac{1}{\Gamma(\alpha)} \int_0^c (c-\tau)^{\alpha-1} P_j(\tau) \mathrm{d}\tau \right] \gamma_j(\sigma_n) \\
=:\; & y_0 + h_1^\alpha \sum_{\nu=1}^{n-1} r^{\alpha(\nu-1)} \sum_{j=0}^{s-1} J_j^\alpha \left(\frac{r^{n-\nu}-1}{r-1} + c r^{n-\nu} \right) \gamma_j(\sigma_\nu) \\
& \quad + h_1^\alpha r^{\alpha(n-1)} \sum_{j=0}^{s-1} I^\alpha P_j(c)\, \gamma_j(\sigma_n) \\
=:\; & \phi_{n-1}^{\alpha,s}(c; h_1, r, \sigma) \;+\; h_1^\alpha r^{\alpha(n-1)} \sum_{j=0}^{s-1} I^\alpha P_j(c)\, \gamma_j(\sigma_n), \qquad c \in [0,1],
\end{aligned} \tag{22}$$

where (see (3))

$$I^\alpha P_j(c) = \frac{1}{\Gamma(\alpha)} \int_0^c (c-\tau)^{\alpha-1} P_j(\tau) \mathrm{d}\tau, \qquad j = 0, \ldots, s-1, \tag{23}$$

is the Riemann-Liouville integral of $P_j(c)$, and, for $x \geq 1$:

$$J_j^\alpha(x) = \frac{1}{\Gamma(\alpha)} \int_0^1 (x-\tau)^{\alpha-1} P_j(\tau) \mathrm{d}\tau, \qquad j = 0, \ldots, s-1. \tag{24}$$

The efficient numerical evaluation of the integrals (23) and (24) will be explained in detail in Section 3.2.

Remark 2 We observe that, for $c = x = 1$, by virtue of (15–16) one has:

$$I^\alpha P_j(1) = J_j^\alpha(1) = \frac{\delta_{j0}}{\Gamma(\alpha+1)}, \qquad j = 0, \ldots, s-1. \tag{25}$$

Similarly, when using a uniform mesh, by means of similar steps as above, (13) turns out to be approximated by:

$$\begin{aligned}\sigma_n(ch_n) \;=\; & \\ & y_0 \;+\; h_1^\alpha \sum_{\nu=1}^{n-1}\sum_{j=0}^{s-1} J_j^\alpha\,(n-\nu+c)\,\gamma_j(\sigma_\nu) \;+\; h_1^\alpha \sum_{j=0}^{s-1} I^\alpha P_j(c)\,\gamma_j(\sigma_n) \\ \equiv\; & \phi_{n-1}^{\alpha,s}(c;h_1,1,\sigma) \;+\; h_1^\alpha \sum_{j=0}^{s-1} I^\alpha P_j(c)\,\gamma_j(\sigma_n), \qquad c\in[0,1]. \end{aligned} \tag{26}$$

In both cases, the approximation at t_n is obtained, by setting $c = 1$ and taking into account (25), as:

$$\begin{aligned}\sigma_n(h_n) &= \phi_{n-1}^{\alpha,s}(1;h_1,r,\sigma) \;+\; \frac{h_1^\alpha r^{\alpha(n-1)}}{\Gamma(\alpha+1)}\gamma_0(\sigma_n) \\ &\equiv \phi_{n-1}^{\alpha,s}(1;h_1,r,\sigma) \;+\; \frac{h_n^\alpha}{\Gamma(\alpha+1)}\gamma_0(\sigma_n), \end{aligned} \tag{27}$$

which formally holds also in the case of a uniform mesh (see (12)) by setting $r = 1$.

2.2 The fully discrete method

Quoting Dahlquist and Björk [22], *"as is well known, even many relatively simple integrals cannot be expressed in finite terms of elementary functions, and thus must be evaluated by numerical methods"*. In our framework, this obvious statement means that, in order to obtain a numerical method, at step n the Fourier coefficients $\gamma_j(\sigma_n)$ in (21) need to be approximated by means of a suitable quadrature formula. Fortunately enough, this can be done up to machine precision by using a Gauss-Jacobi formula of order $2k$ based at the zeros of $P_k(c)$, $c_1,\ldots,c_k$, with corresponding weights (see (15))

$$b_i = \int_0^1 \omega(c)\ell_i(c)\mathrm{d}c, \qquad \ell_i(c) = \prod_{j\neq i}\frac{c-c_j}{c_i-c_j}, \qquad i=1,\ldots,k,$$

by choosing a value of k, $k \geq s$, large enough. In other words,

$$\gamma_j^n \;:=\; \sum_{i=1}^{k} b_i\,P_j(c_i)\,f(\sigma_n(c_i h_n)) \;\doteq\; \gamma_j(\sigma_n), \qquad j=0,\ldots,s-1, \tag{28}$$

where $\doteq$ means *equal within machine precision*. Because of this, and for sake of brevity, we shall continue using σ_n to denote the fully discrete approximation (compare

with (22), or with (26), in case $r = 1$):[3]

$$\sigma_n(ch_n) = \phi_{n-1}^{\alpha,s}(c; h_1, r, \sigma) + h_n^\alpha \sum_{j=0}^{s-1} I^\alpha P_j(c)\gamma_j^n, \qquad c \in [0,1]. \tag{29}$$

As is clear, the coefficientes γ_j^ν, $j = 0, \ldots, s-1$, $\nu = 1, \ldots, n-1$, needed to evaluate $\phi_{n-1}^{\alpha,s}(c; h_1, r, \sigma)$, have been already computed at the previous time-steps.

We observe that, from (28), in order to compute the (discrete) Fourier coefficients, it is enough evaluating (29) only at the quadrature abscissae $c_1, \ldots, c_k$. In so doing, by combining (28) and (29), one obtains a discrete problem in the form:

$$\begin{aligned} \gamma_j^n = \sum_{i=1}^{k} b_i P_j(c_i) f\left(\phi_{n-1}^{\alpha,s}(c_i; h_1, r, \sigma) + h_n^\alpha \sum_{\ell=0}^{s-1} I^\alpha P_\ell(c_i)\gamma_\ell^n\right), \\ j = 0, \ldots, s-1. \end{aligned} \tag{30}$$

Once it has been solved, according to (27), the approximation of $y(t_n)$ is given by:

$$\bar{y}_n := \sigma_n(h_n) = \phi_{n-1}^{\alpha,s}(1; h_1, r, \sigma) + \frac{h_n^\alpha}{\Gamma(\alpha+1)}\gamma_0^n. \tag{31}$$

It is worth noticing that the discrete problem (30) can be cast in vector form, by introducing the (block) vectors

$$\gamma^n = \begin{pmatrix} \gamma_0^n \\ \vdots \\ \gamma_{s-1}^n \end{pmatrix} \in \mathbb{R}^{sm}, \qquad \phi_{n-1}^{\alpha,s} = \begin{pmatrix} \phi_{n-1}^{\alpha,s}(c_1; h_1, r, \sigma) \\ \vdots \\ \phi_{n-1}^{\alpha,s}(c_k; h_1, r, \sigma) \end{pmatrix} \in \mathbb{R}^{km},$$

and the matrices

$$\mathcal{P}_s = \begin{pmatrix} P_0(c_1) & \ldots & P_{s-1}(c_1) \\ \vdots & & \vdots \\ P_0(c_k) & \ldots & P_{s-1}(c_k) \end{pmatrix}, \quad \mathcal{I}_s^\alpha = \begin{pmatrix} I^\alpha P_0(c_1) & \ldots & I^\alpha P_{s-1}(c_1) \\ \vdots & & \vdots \\ I^\alpha P_0(c_k) & \ldots & I^\alpha P_{s-1}(c_k) \end{pmatrix} \in \mathbb{R}^{k\times s},$$

$$\Omega = \begin{pmatrix} b_1 & & \\ & \ddots & \\ & & b_k \end{pmatrix} \in \mathbb{R}^{k\times k},$$

as:

$$\gamma^n = \mathcal{P}_s^\top \Omega \otimes I_m f\left(\phi_{n-1}^{\alpha,s} + h_n^\alpha \mathcal{I}_s^\alpha \otimes I_m \gamma^n\right), \tag{32}$$

[3] Hereafter, we shall use h_n in place of $h_1 r^{n-1}$.

with the obvious notation for the function f, evaluated in a vector of (block) dimension k, of denoting the (block) vector of dimension k containing the function f evaluated in each (block) entry. As observed in [8], the (block) vector

$$Y := \phi_{n-1}^{\alpha,s} + h_n^{\alpha} \mathcal{I}_s^{\alpha} \otimes I_m \gamma^n \in \mathbb{R}^{km}, \tag{33}$$

appearing at the r.h.s. in (32) as argument of f, satisfies the equation

$$Y = \phi_{n-1}^{\alpha,s} + h_n^{\alpha} \mathcal{I}_s^{\alpha} \mathcal{P}_s^{\top} \Omega \otimes I_m f(Y), \tag{34}$$

obtained combining (32) and (33). Consequently, it can be regarded as the stage vector of a Runge-Kutta type method with Butcher tableau

$$\begin{array}{c|c} \boldsymbol{c} & \mathcal{I}_s^{\alpha} \mathcal{P}_s^{\top} \Omega \\ \hline & \boldsymbol{b}^{\top} \end{array}, \qquad \boldsymbol{b} = \left(b_1, \ldots, b_k\right)^{\top}, \qquad \boldsymbol{c} = \left(c_1, \ldots, c_k\right)^{\top}. \tag{35}$$

Remark 3 Though the two formulations (32) and (34) are equivalent each other, nevertheless, the former has (block) dimension s *independently* of the considered value of k ($k \geq s$), which is the (block) dimension of the latter. Consequently, in the practical implementation of the method, the discrete problem (32) is the one to be preferred, since it is independent of the number of stages.

When $\alpha = 1$, the Runge-Kutta method (35) reduces to a HBVM(k, s) method [3, 5, 9–12, 19, 20]. Consequently, we give the following definition [8].

Definition 1 The method defined by (35) (i.e., (31–32)) is called *fractional HBVM with parameters* (k, s). In short, FHBVM(k, s).

The efficient numerical solution of the discrete problem (32) will be considered in Section 3.4.

3 Implementation issues

In this section we report all the implementation details used in the Matlab© code `fhbvm`, which will be used in the numerical tests reported in Section 4.

3.1 Graded or uniform mesh?

The first relevant problem to face is the choice between a graded or a uniform mesh and, in the former case, also choosing appropriate values for the parameters r, N, and h_1 in (9–10). We start considering a proper choice of this latter parameter, i.e., h_1 which, in turn, will allow us to choose the type of mesh, too. For this purpose, the user is required to provide, in input, a convenient integer value $M > 1$, such that, if a uniform mesh is appropriate, then the stepsize is given by:

$$h = \frac{T}{M}. \tag{36}$$

It is to be noted that, since the code is using a method with spectral accuracy in time, the value of M should be as small as possible. That said, we set $h_1^0 = h$ and apply the code on the interval $[0, h_1^0]$ both in a single step, and by using a graded mesh of 2 sub-intervals defined by a value of $r := \hat{r} \equiv 3$. As a result, the two sub-intervals, obtained by solving (10) for h_1, with

$$r = 3, \qquad N = 2, \qquad T = h_1^0,$$

are[4]

$$[0, h_1^0/4], \qquad [h_1^0/4, h_1^0].$$

In so doing, we obtain two approximations to $y(h_1^0)$, say y_1 and y_2. According to the analysis in [8], if[5]

$$\|(y_1 - y_2)./(1 + |y_2|)\|_\infty \leq tol, \tag{37}$$

with *tol* a suitably small tolerance,[6] then this means the stepsize $h_1 := h_1^0$ is appropriate. Moreover, in such a case, a uniform mesh with $N \equiv M$ turns out to be appropriate, too.

Conversely, the procedure is repeated on the sub-interval $[0, h_1^1] \equiv [0, h_1^0/4]$, and so forth, until a convenient initial stepsize h_1 is obtained. This process can be repeated up to a suitable maximum number of times that, considering that at each iteration the current guess of h_1 is divided by 4, allows for a substantial reduction of the initial value (36). Clearly, in such a case, a graded mesh is needed. At the end of this procedure, we have then chosen the initial stepsize h_1 which, if the procedure ends at the ℓ-th iteration, for a convenient $\ell > 1$, is given by:

$$h_1 = 4^{1-\ell} h \equiv 4^{1-\ell} \frac{T}{M}. \tag{38}$$

We need now to appropriately choose the parameters r and N in (9–10). To simplify this choice, we require that the last stepsize in the mesh be equal to (36). Consequently, we would ideally satisfy the following requirements:

$$h_1 \frac{r^N - 1}{r - 1} = T, \qquad h_1 r^{N-1} = \frac{T}{M},$$

which, by virtue of (38), become

$$4^{1-\ell} \frac{r^N - 1}{r - 1} = M, \qquad 4^{1-\ell} r^{N-1} = 1.$$

[4] It is to be noticed that the division by 4 is done without introducing round-off errors.

[5] Here, ./ means the componentwise division between two vectors, and $|y_2|$ denotes the vector with the absolute values of the entries of y_2.

[6] In view of the spectral accuracy of the method, this tolerance is only slightly larger than the machine epsilon.

Combining the two equations then gives:

$$r = \frac{M - 4^{1-\ell}}{M-1} > 1, \qquad N = 1 + \log_r(4^{\ell-1}). \tag{39}$$

As is clear, this value of N is not an integer, in general, so that we shall choose, instead:

$$N = \lceil 1 + \log_r(4^{\ell-1}) \rceil. \tag{40}$$

Remark 4 From (40), and considering that $Nh_1 < T$ (conversely, a uniform mesh could have been used) and, by virtue of (38), one has:

$$2 \le N < 4^{\ell-1} M. \tag{41}$$

As is clear, using the value of N in (40) in place of that in (39) implies that we need to recompute r, so that the requirement (now h_1, N, and T are given)

$$h_1 \frac{r^N - 1}{r-1} = T \tag{42}$$

is again fulfilled. Equation (42) can be rewritten as

$$r = [1 + (r-1)\beta]^{1/N} =: \psi(r), \qquad \beta := \frac{T}{h_1} > 1, \tag{43}$$

thus inducing the iterative procedure

$$r_0 > 1 \quad (given), \qquad r_{i+1} = \psi(r_i), \qquad i = 0, 1, \ldots. \tag{44}$$

As a convenient choice for r_0, one can use the guess for r defined in (39). The following result can be proved.

Theorem 1 *There exists a unique $\bar{r} > 1$ satisfying $\bar{r} = \psi(\bar{r})$, and the iteration (44) globally converges to this value over the interval $(1, +\infty)$.*

Proof A direct computation shows that:

$$\psi(1) = 1, \qquad \psi'(r) > 0, \ \text{for } r > 1, \tag{45}$$

$$\psi'(1) = \frac{\beta}{N} > 1, \quad \lim_{r \to +\infty} \psi'(r) = 0, \qquad \psi''(r) < 0, \ \text{for } r > 1. \tag{46}$$

From (45) we deduce that the (positive) mapping $\psi(r)$ is strictly increasing and admits $(1, +\infty)$ as invariant set, since $r > 1$ implies $\psi(r) > \psi(1) = 1$.

From (46) we additionally deduce that $\psi(r)$ is concave and $\psi(r) > r$ for $r \in (1, 1+\varepsilon)$, with $\varepsilon > 0$ sufficiently small. These properties and the fact that $\psi'(r) \to 0$, for $r \to +\infty$, imply that:

– the equation $r = \psi(r)$ admits a unique solution $\bar{r} > 1$ and $\psi'(\bar{r}) < 1$;

– $\psi\big((1, \bar{r})\big) \subset (1, \bar{r})$ and $\psi\big((\bar{r}, +\infty)\big) \subset (\bar{r}, +\infty)$ (since ψ is increasing);

– $\psi(r) > r$ for $1 < r < \bar{r}$ and $\psi(r) < r$ for $r > \bar{r}$.

From the two latter properties we conclude that, for any $r_0 > 1$, the sequence generated by (44) converges monotonically to $\bar{r}$. □

Consequently, we have derived the parameters h_1 in (38), N in (40), and r satisfying (42) of the graded mesh (9–10), the latter one obtained by a few iterations of (44).

Remark 5 In the actual implementation of the code, we allow the use of a uniform mesh also when $\ell = 2$ steps of the previous procedure are required, provided that M has a moderate value (say, $M \leq 5$). Consequently, for the final mesh, $h_1 = h/4$, $r = 1$, and $N = 4M$.

3.2 Approximating the fractional integrals

The practical implementation of the method requires the evaluation of the following integrals (recall (30) and the definitions (23–24)):

$$I^\alpha P_j(c_i), \qquad i = 1, \ldots, k, \quad j = 0, \ldots, s-1, \tag{47}$$

and

$$J_j^\alpha \left(\frac{r^\nu - 1}{r - 1} + c_i r^\nu \right), \qquad i = 1, \ldots, k, \quad j = 0, \ldots, s-1, \quad \nu = 1 \ldots, N-1, \tag{48}$$

in case a graded mesh is used, or

$$J_j^\alpha(\nu + c_i), \qquad i = 1, \ldots, k, \quad j = 0, \ldots, s-1, \quad \nu = 1 \ldots, N-1, \tag{49}$$

in case a uniform mesh is used.

It must be emphasized that all such integrals ((47) and (48), or (47) and (49)), can be pre-computed once for all, for later use. For their computation, in the first version of the software, we adapted an algorithm based on [5] which, however, required a quadruple precision, for the considered values of k and s. In this respect, the use of the standard `vpa` of Matlab©, which is based on a symbolic computation, turned out to be too slow. For this reason, hereafter we describe two new algorithms for computing the above integrals, which result to be quite satisfactory, when using the standard double precision IEEE. Needless to say that, since `vpa` is no more required, they turn out to be much faster than the previous ones.

Let us describe, at first, the computation of (47). One has, by considering that $k \geq s$ and $c_i \in (0, 1)$:

$$
\begin{aligned}
I^{\alpha} P_j(c_i) &= \frac{1}{\Gamma(\alpha)} \int_0^{c_i} (c_i - \tau)^{\alpha-1} P_j(\tau) \mathrm{d}\tau \\
&= \frac{c_i^{\alpha-1}}{\Gamma(\alpha)} \int_0^{c_i} \left(1 - \frac{\tau}{c_i}\right)^{\alpha-1} P_j(\tau) \mathrm{d}\tau \; = \; \frac{c_i^{\alpha}}{\Gamma(\alpha)} \int_0^{1} (1 - \xi)^{\alpha-1} \, P_j(\xi c_i) \mathrm{d}\xi \\
&= \frac{c_i^{\alpha}}{\Gamma(\alpha+1)} \int_0^{1} \alpha \, (1 - \xi)^{\alpha-1} \, P_j(\xi c_i) \mathrm{d}\xi \; \equiv \; \frac{c_i^{\alpha}}{\Gamma(\alpha+1)} \sum_{\ell=1}^{k} b_\ell P_j(c_i c_\ell), \\
&\qquad\qquad j = 0, \ldots, s-1,
\end{aligned}
$$

where the last equality holds because the Jacobi quadrature formula has order $2k$. Clearly, for each $i = 1, \ldots, k$, all the above integrals can be computed in vector form in "one shot", by using the usual three-term recurrence to compute the Jacobi polynomials.

Concerning the integrals (48-49), let us now consider the evaluation of a generic $J_j^{\alpha}(x)$, $j = 0, \ldots, s-1$, for $x > 1$. In this respect, there is numerical evidence that, for $x \geq 1.1$, a high-order Gauss-Legendre formula is able to approximate the required integral to full machine precision. Since we will use a value of $s = 20$, we consider, for this purpose, a Gauss-Legendre formula of order 60, which turns out to be fully accurate. Instead, for $x \in (1, 1.1)$, one has:

$$
\begin{aligned}
J_j^{\alpha}(x) &= \frac{1}{\Gamma(\alpha)} \int_0^{1} (x - \tau)^{\alpha-1} P_j(\tau) \mathrm{d}\tau \\
&= \frac{1}{\Gamma(\alpha)} \left[\int_0^{x} (x - \tau)^{\alpha-1} P_j(\tau) \mathrm{d}\tau \; - \; \int_1^{x} (x - \tau)^{\alpha-1} P_j(\tau) \mathrm{d}\tau \right] \\
&= \frac{x^{\alpha-1}}{\Gamma(\alpha)} \int_0^{x} \left(1 - \frac{\tau}{x}\right)^{\alpha-1} P_j(\tau) \mathrm{d}\tau \\
&\qquad - \frac{(x-1)^{\alpha-1}}{\Gamma(\alpha)} \int_0^{x-1} \left(1 - \frac{c}{x-1}\right)^{\alpha-1} P_j(1 + c) \mathrm{d}c \\
&= \frac{x^{\alpha}}{\Gamma(\alpha)} \int_0^{1} (1 - \xi)^{\alpha-1} P_j(\xi x) \mathrm{d}\xi - \frac{(x-1)^{\alpha}}{\Gamma(\alpha)} \int_0^{1} (1 - \xi)^{\alpha-1} P_j(1 + \xi(x-1)) \mathrm{d}\xi \\
&= \frac{x^{\alpha}}{\Gamma(\alpha+1)} \int_0^{1} \alpha (1 - \xi)^{\alpha-1} P_j(\xi x) \mathrm{d}\xi \\
&\qquad - \frac{(x-1)^{\alpha}}{\Gamma(\alpha+1)} \int_0^{1} \alpha (1 - \xi)^{\alpha-1} P_j(1 + \xi(x-1)) \mathrm{d}\xi \\
&\equiv \frac{x^{\alpha}}{\Gamma(\alpha+1)} \sum_{\ell=1}^{k} b_\ell P_j(c_\ell x) \; - \; \frac{(x-1)^{\alpha}}{\Gamma(\alpha+1)} \sum_{\ell=1}^{k} b_\ell P_j(1 + c_\ell (x-1)), \\
&\qquad\qquad j = 0, \ldots, s-1,
\end{aligned}
$$

again, due to the fact that the quadrature is exact for polynomials of degree at most $s-1$. Also in this case, for each fixed $x > 1$, all the integrals can be computed in "one shot" by using the three-term recurrence of the Jacobi polynomials.

We observe that the previous expression is exact also for $x = 1$, since $J_j^\alpha(1) = \delta_{j0}/\Gamma(\alpha+1)$.

3.3 Error estimation

An estimate of the global error can be derived by computing the solution on a doubled mesh. In other words, if (see (31))

$$\bar{y}_n \approx y(t_n), \qquad n = 0, \ldots, N,$$

with t_n as in (5–6), are the obtained approximations, then

$$e_n := y(t_n) - \bar{y}_n \approx \hat{y}_{2n} - \bar{y}_n, \qquad n = 0, \ldots, N,$$

where $\hat{y}_n \approx y(\hat{t}_n)$, $n = 0, \ldots, 2N$, is the solution computed on a doubled mesh.

When a uniform mesh (12) is used, the doubled mesh is simply given by:

$$\hat{t}_n = n\frac{h_1}{2} \equiv n\frac{T}{2N}, \qquad n = 0, \ldots, 2N.$$

Conversely, when a graded mesh (9–10) is considered, the doubled mesh is given by:

$$\hat{t}_n = \hat{t}_{n-1} + \hat{h}_n, \qquad \hat{h}_n = \hat{r}\hat{h}_{n-1} \equiv \hat{r}^{n-1}\hat{h}_1, \qquad n = 1, \ldots, 2N,$$

with $\hat{t}_0 = 0$, and

$$\hat{r} = \sqrt{r}, \qquad \hat{h}_1 = h_1\frac{\hat{r}-1}{r-1}.$$

The choice of $\hat{h}_1$ is done for having

$$\hat{h}_1\frac{\hat{r}^{2N}-1}{\hat{r}-1} = \hat{h}_1\frac{r^N-1}{\hat{r}-1} \equiv h_1\frac{r^N-1}{r-1} = T,$$

according to (10).

3.4 The nonlinear iteration

At last, we describe the efficient numerical solution of the discrete problem (32), which has to be solved at the n-th integration step. As is clear, the very formulation of the problem induces a straightforward fixed-point iteration:

$$\gamma^{n,\ell} = \mathcal{P}_s^\top \Omega \otimes I_m f\left(\phi_{n-1}^{\alpha,s} + h_n^\alpha \mathcal{I}_s^\alpha \otimes I_m \gamma^{n,\ell-1}\right), \qquad n = 1, 2, \ldots, \tag{50}$$

which can be conveniently started from $\gamma^{n,0} = \mathbf{0}$. The following straightforward result holds true.

Theorem 2 *Assume f be Lipchitz with constant L in in the interval $[t_{n-1}, t_n]$. Then, the iteration (50) is convergent for all timesteps h_n such that*

$$h_n^\alpha L \|\mathcal{P}_s^\top \Omega\| \|\mathcal{I}_s^\alpha\| < 1.$$

Proof See [8, Theorem 2]. □

Nevertheless, as is easily seen, also the simple equation

$$y^{(\alpha)} = -\lambda y, \qquad t \in [0, T], \qquad y(0) = 1, \qquad T, \lambda \gg 1,$$

whose solution is almost everywhere close to 0, after an initial transient, suffers from stepsize limitations, if the fixed-point iteration (50) is used, since it has to be everywhere proportional to $\lambda^{-1/\alpha}$.

In order to overcome this drawback, a Newton-type iteration is therefore needed. Hereafter, we consider the so-called *blended iteration* which has been at first studied in a series of papers [7, 14, 16, 17]. It has been implemented in the Fortran codes `BIM` [15], for ODE-IVPs, and `BIMD` [18], for ODE-IVPs and linearly implicit DAEs, and in the Matlab code `hbvm` [10, 13], for solving Hamiltonian problems. We here consider its adaption for solving (32). By neglecting, for sake of brevity, the time-step index n, we then want to solve the equation:

$$G(\gamma) := \gamma - \mathcal{P}_s^\top \Omega \otimes I_m f\left(\phi^{\alpha,s} + h^\alpha \mathcal{I}_s^\alpha \otimes I_m \gamma\right) = \mathbf{0}. \tag{51}$$

By setting f_0' the Jacobian of f evaluated at the first entry of $\phi^{\alpha,s}$, $I = I_s \otimes I_m$, and

$$X_s^\alpha := \mathcal{P}_s^\top \Omega \mathcal{I}_s^\alpha, \tag{52}$$

the application of the simplified Newton method then reads:

$$\begin{aligned} \text{solve}: &\ \left(I - h^\alpha X_s^\alpha \otimes f_0'\right) \Delta\gamma^\ell = -G(\gamma^\ell) \equiv \boldsymbol{\eta}^\ell, \\ \text{set}: &\ \gamma^{\ell+1} = \gamma^\ell + \Delta\gamma^\ell, \qquad \ell = 0, 1, \ldots. \end{aligned} \tag{53}$$

Even though this iteration has the advantage of using a coefficient matrix which is constant at each time-step, nevertheless, its dimension may be large, when either s or m are large. To study a different iteration, able to get rid of this problem, let us decouple the linear system into the various eigenspaces of f_0', thus studying the simpler problem

$$\begin{aligned} \text{solve}: &\ \left(I_s - h^\alpha \mu X_s^\alpha\right) \Delta\gamma^\ell = -g(\gamma^\ell) \equiv \eta^\ell, \\ \text{set}: &\ \gamma^{\ell+1} = \gamma^\ell + \Delta\gamma^\ell, \qquad \ell = 0, 1, \ldots, \end{aligned}$$

with all involved vectors of dimension s, and $\mu \in \sigma(f_0')$ a generic eigenvalue of f_0', and an obvious meaning of $g(\gamma^\ell)$. By setting $q = h^\alpha \mu$, the iteration then reads:

$$\begin{aligned} \text{solve}: \ & \left(I_s - q X_s^\alpha\right) \Delta\gamma^\ell = \eta^\ell, \\ \text{set}: \ & \gamma^{\ell+1} = \gamma^\ell + \Delta\gamma^\ell, \qquad \ell = 0, 1, \ldots. \end{aligned} \tag{54}$$

Hereafter, we consider the iterative solution of the linear system in (54). A linear analysis of convergence (in the case the r.h.s. is constant) is then made, as at first suggested in [31–33], and later refined in [14, 17]. Consequently, skipping the iteration index ℓ, let us consider the linear system to be solved:

$$\left(I_s - q X_s^\alpha\right) \Delta\gamma = \eta,$$

and its equivalent formulation, derived considering that matrix (52) is nonsingular, and with $\xi > 0$ a parameter to be later specified,

$$\xi \left((X_s^\alpha)^{-1} - q I_s\right) \Delta\gamma = \xi (X_s^\alpha)^{-1} \eta =: \eta_1.$$

Further, we consider the *blending* of the previous two equivalent formulations with weights $\theta(q)$ and $I_s - \theta(q)$, where, by setting $O \in \mathbb{R}^{s\times s}$ the zero matrix,

$$\theta(q) := I_s (1 - \xi q)^{-1} \begin{cases} \approx I_s, & q \approx 0, \\ \to O, & q \to \infty. \end{cases}$$

In so doing, one obtains the linear system

$$M(q) \Delta\gamma = \eta_1 + \theta(q)(\eta - \eta_1) =: \eta(q), \tag{55}$$

with the coefficient matrix,

$$M(q) = \xi \left((X_s^\alpha)^{-1} - q I_s\right) + \theta(q) \left[\left(I_s - q X_s^\alpha\right) - \xi \left((X_s^\alpha)^{-1} - q I_s\right)\right],$$

such that [14]:

$$M(q) \approx \begin{cases} I_s, & q \approx 0, \\ -\xi q I_s, & |q| \gg 1. \end{cases}$$

This naturally induces the splitting matrix

$$N(q) := I_s (1 - \xi q) \equiv \theta(q)^{-1}, \tag{56}$$

defining the *blended iteration*

$$\Delta\gamma_i = [I_s - \theta(q) M(q)] \Delta\gamma_{i-1} + \theta(q)\eta(q), \qquad i = 1, 2, \ldots. \tag{57}$$

This latter iteration converges iff the spectral radius of the iteration matrix,

$$\rho\left(I_s - \theta(q)M(q)\right) =: \rho(q) < 1. \tag{58}$$

The iteration is said to be A-convergent if (58) holds true for all $q \in \mathbb{C}^-$, the left-half complex plane, and L-convergent if, in addition, $\rho(q) \to 0$, as $q \to \infty$. Since [14]

$$\theta(0)M(0) = I_s, \qquad \theta(q)M(q) \to I_s, \quad q \to \infty,$$

the blended iteration is L-convergent iff it is A-convergent. For this purpose, we shall look for a suitable choice of the positive parameter $\xi > 0$. Considering that $\theta(q)$ is well defined for all $q \in \mathbb{C}^-$, the following statement easily follows from the maximum modulus theorem.

Theorem 3 *The blended iteration is L-convergent iff the* maximum amplification factor,

$$\rho^* := \max_{x>0} \rho(\mathrm{i}x) \leq 1.$$

The following result also holds true.

Theorem 4 *The eigenvalues of the iteration matrix $I_s - \theta(q)M(q)$ are given by:*

$$\frac{q(\lambda - \xi)^2}{\lambda(1 - q\xi)^2}, \qquad \lambda \in \sigma(X_s^\alpha).$$

Consequently, the maximum amplification factor is given by:

$$\rho^* = \max_{\lambda \in \sigma(X_s^\alpha)} \frac{|\lambda - \xi|^2}{2\xi|\lambda|}. \tag{59}$$

Proof See [14, Theorem 2 and (25)], by considering that

$$\max_{q \in \mathbb{C}^-} \frac{q(\lambda - \xi)^2}{\lambda(1 - q\xi)^2} \equiv \max_{x>0} \frac{x|\lambda - \xi|^2}{|\lambda|(1 + x^2\xi^2)}$$

is obtained at $x = \xi^{-1}$, so that (59) follows. □

Slightly generalizing the arguments in [14], we then consider the following choice of the parameter ξ,

$$\xi = \mathrm{argmin}_{\mu \in \sigma(X_s^\alpha)} \max_{\lambda \in \sigma(X_s^\alpha)} \frac{|\lambda - |\mu||^2}{2|\mu||\lambda|}, \tag{60}$$

which is computed once forall, and always provides, in our experiments, an L-convergent iteration. In particular, the code `fhbvm` uses, at the moment, $k = 22$ and $s = 20$: the corresponding maximum amplification factor (59) is depicted in Fig. 1, w.r.t. the order α of the fractional derivative, thus confirming this.

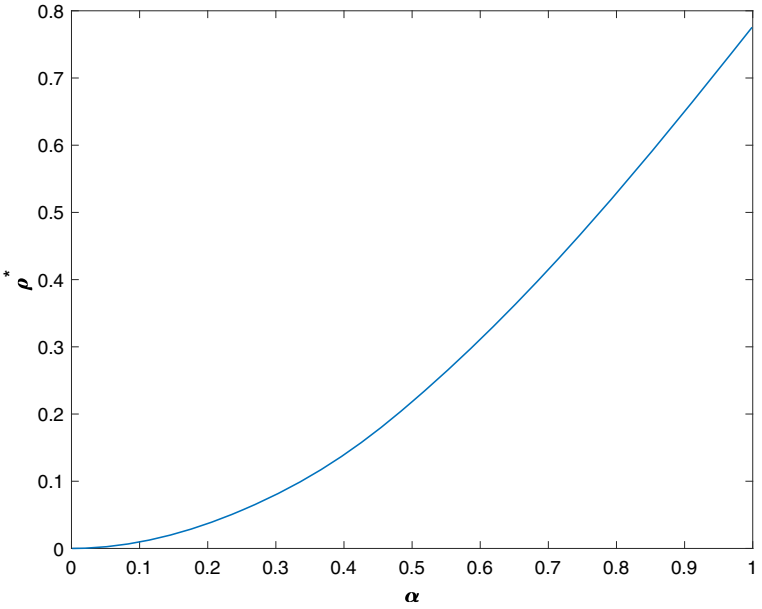

Fig. 1 Maximum amplification factor (59–60), $k = 22$ and $s = 20$

Coming back to the original problem (53), starting from the initial guess $\Delta\gamma = \mathbf{0}$, and updating the r.h.s. as soon as a new approximation to γ is available, one has that the iteration (55–57) simplifies to:

$$\begin{aligned} \boldsymbol{\eta}^{\ell} &= -G(\gamma^{\ell}) \\ \boldsymbol{\eta}_1^{\ell} &= \xi (X_s^{\alpha})^{-1} \otimes I_m \, \boldsymbol{\eta}^{\ell} \\ \gamma^{\ell+1} &= \gamma^{\ell} + I_s \otimes \Theta \left[\boldsymbol{\eta}_1^{\ell} + I_s \otimes \Theta \left(\boldsymbol{\eta}^{\ell} - \boldsymbol{\eta}_1^{\ell} \right) \right], \qquad \ell = 0, 1, \ldots, \end{aligned} \tag{61}$$

with ξ chosen according to (60), and

$$\Theta = \left(I_m - h^{\alpha} \xi \, f_0' \right)^{-1}.$$

Consequently, only the factorization of one matrix, having the same dimension m of the problem, is needed. Moreover, the initial guess $\gamma^0 = \mathbf{0}$ can be conveniently considered in (61).

Remark 6 It is worth mentioning that, due to the properties of the Kronecker product, the iteration (61) can be compactly cast in matrix form, thus avoiding an explicit use of the Kronecker product. This implementation has been considered in the code `fhbvm` used in the numerical tests.

Actually, according to Theorem 2, in the code `fhbvm` we automatically switch between the fixed-point iteration (50) or the blended iteration (61), depending on the

fact that

$$h^{\alpha}\|f_0'\|\|\mathcal{P}_s^{\top}\Omega\|\|\mathcal{I}_s^{\alpha}\| \leq tol,$$

with $tol < 1$ a suitable tolerance.

4 Numerical Tests

In this section we report a few numerical tests using the Matlab© code `fhbvm`: the code implements a FHBVM(22,20) method using all the strategies discussed in the previous section. The calling sequence of the code is:

```
[t,y,stats,err] = fhbvm( fun, y0, T, M )
```

In more details,

In input:

- `fun` is the identifier (or the function handling) of the function evaluating the r.h.s. of the equation (also in vector mode), its Jacobian, and the order α of the fractional derivative (see `help fhbvm` for more details);
- `y0` is the initial condition;
- `T` is the final integration time;
- `M` is the parameter in (36) (it should be as small as possible);

In output:

- `t,y` contain the computed mesh and solution;
- `stats` (optional) is a vector containing the following time statistics:
  1. the pre-processing time for computing the parameters h_1, r, and N (see Section (3.1)) and the fractional itegrals (47), and (48) or (49);
  2. the time for solving the problem;
  3. the pre-processing time for computing the fractional itegrals (47), and (48) or (49) for the error estimation;
  4. the time for solving the problem on the doubled mesh, for the error estimation;
- `err` (optional), if specified, contains the estimate of the absolute error. This estimate, obtained on a doubled mesh, is relatively costly: for this reason, when the parameter is not specified, the solution on the doubled mesh is not computed.

For the first two problems, we shall also make a comparison with the Matlab© code `flmm2` [27],[7] in order to emphasize the potentialities of the new code. All numerical tests have been done on a M2-Silicon based computer with 16GB of shared memory, using Matlab© R2023b.

The comparisons will be done by using a so called *Work Precision Diagram (WPD)*, where the execution time (in `sec`) is plotted against accuracy. The accuracy, in turn, is

[7] In particular, the BDF2 method is selected (`method=3`), with the parameters `tol=1e-15` and `itmax=1000`.

measured through the *mixed error significant computed digits* (mescd) [39], defined, by using the same notation seen in (37), as [8]

$$\texttt{mescd} := -\log_{10} \max_{i=0,\ldots,N} \|(y(t_i) - \bar{y}_i)./(1 + |y(t_i)|)\|_\infty,$$

being t_i, $i = 0, \ldots, N$, the computational mesh of the considered solver, and $y(t_i)$ and $\bar{y}_i$ the corresponding values of the solution and of its approximation.

4.1 Example 1

The first problem [28] is given by:

$$y^{(\alpha)} = -|y|^{1.5} + \frac{8!}{\Gamma(9-\alpha)} t^{8-\alpha} - 3\frac{\Gamma(5+\alpha/2)}{\Gamma(5-\alpha/2)} t^{4-\alpha/2} + \left(\frac{3}{2} t^{\alpha/2} - t^4\right)^3 + \frac{9}{4}\Gamma(\alpha+1), \qquad t \in [0,1], \qquad y(0) = 0, \tag{62}$$

whose solution is

$$y(t) = t^8 - 3\, t^{4+\alpha/2} + \frac{9}{4} t^\alpha.$$

We consider the value $\alpha = 0.3$, and use the codes with the following parameters, to derive the corresponding WPD:

- flmm2: $h = 10^{-1} 2^{-\nu}$, $\nu = 1, \ldots, 20$;
- fhbvm: $M = 2, 3, 4, 5$.

Figure 2 contains the obtained results: as one may see, flmm2 reaches less than 12 mescd, since by continuing reducing the stepsize, at the 15-th mesh doubling the error starts increasing. On the other hand, the execution time essentially doubles at each new mesh doubling. Conversely, fhbvm can achieve full machine accuracy by employing a uniform mesh with stepsize $h_1 = 1/M$, with M very small, thus using very few mesh points. As a result, fhbvm requires very short execution times.

4.2 Example 2

We now consider the following linear problem:

$$y^{(0.5)} = \begin{pmatrix} -50 & 0 \\ -49 & -1 \end{pmatrix} y, \qquad t \in [0, 20],$$
$$y(0) = (2,\ 3)^\top, \tag{63}$$

[8] This definition corresponds to set atol=rtol in the definition used in [39].

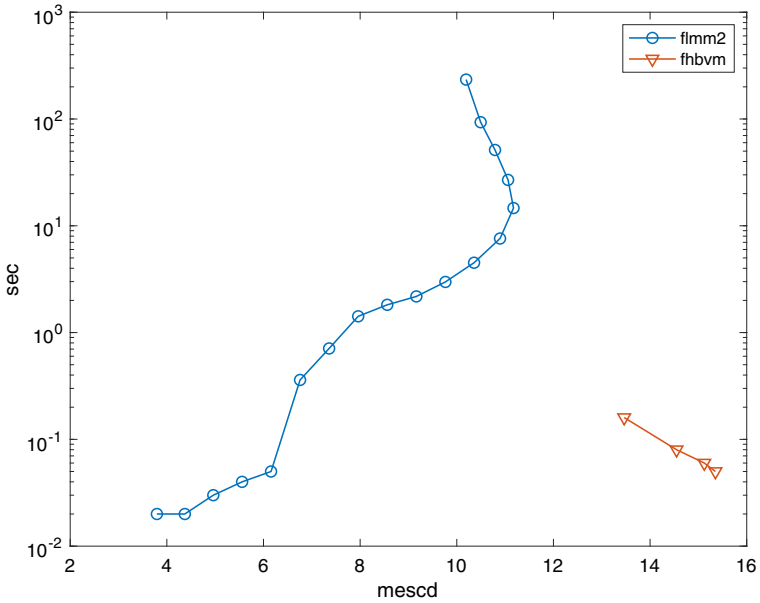

Fig. 2 Work-precision diagram for problem (62), $\alpha = 0.3$

having solution

$$y(t) = \begin{pmatrix} 2\,E_{0.5}\left(-50 \cdot t^{0.5}\right) \\ 2\,E_{0.5}\left(-50 \cdot t^{0.5}\right) + E_{0.5}\left(-t^{0.5}\right) \end{pmatrix},$$

with $E_{0.5}$ the Mittag-Leffler function.[9] We use the codes with the following parameters, to derive the corresponding WPD:

- `flmm2`: $h = 10^{-1}2^{-\nu}$, $\nu = 1, \ldots, 20$;
- `fhbvm`: $M = 5, \ldots, 10$.

Figure 3 contains the obtained results, from which one deduces that `flmm2` achieves about 5 `mescd` (with a run time of about 85 `sec`), whereas `fhbvm` has approximately 13 `mescd`, with an execution time of about 1 `sec`. Further, in Fig. 4, we plot the true and estimated (absolute) errors for `fhbvm` in the case $M = 10$ (corresponding to a computational mesh made up of 251 mesh-points, with an initial stepsize $h_1 \approx 7.3 \cdot 10^{-12}$, and a final stepsize $h_{250} \approx 2$): as one may see from the figure, there is a substantial agreement between the two errors.

[9] We have used the Matlab© function `ml` [26] for its evaluation.

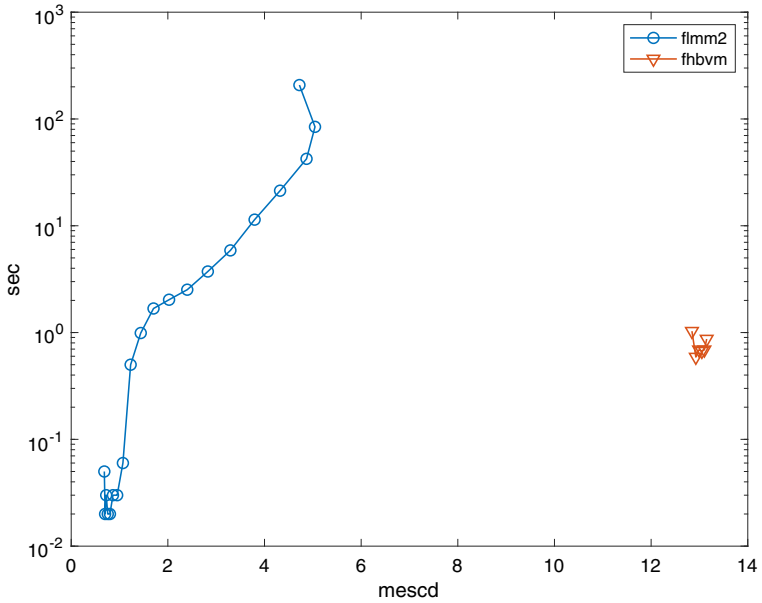

Fig. 3 Work-precision diagram for problem (63)

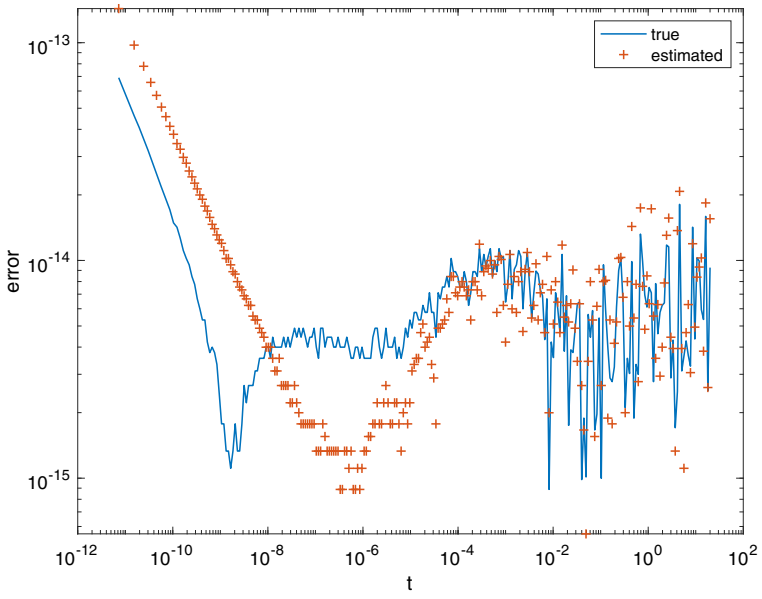

Fig. 4 True and estimated absolute errors for `fhbvm` solving problem (63), $M = 10$

4.3 Example 3

We now consider the following nonlinear problem [8]:

$$\begin{aligned} y_1^{(1/3)}(t) &= \frac{t}{10}\left(y_1^3 - \left(\sqrt{y_2}+1\right)^3\right) + \frac{\Gamma(5/3)}{\Gamma(4/3)}t^{1/3}, \\ y_2^{(1/3)}(t) &= \frac{1}{3}\left(y_2^3 - (y_1-1)^6\right) + \Gamma(7/3)t, \qquad t \in [0,1], \\ y(0) &= (1,\ 0)^\top, \end{aligned} \tag{64}$$

having solution

$$y(t) = \begin{pmatrix} t^{2/3}+1 \\ t^{4/3} \end{pmatrix}.$$

This problem is relatively simple and, in fact, both `flmm2` and `fhbvm` solve it accurately. We use it to show the estimated error by using `fhbvm` with parameter $M = 2$, which produces a graded mesh with 41 mesh-points, with $h_1 \approx 1.8 \cdot 10^{-12}$ and $h_{40} \approx 0.49$. The absolute errors (true and estimated) for each component are depicted in Fig. 5, showing a perfect agreement for both of them.

In this case, the evaluation of the solution requires ≈ 0.04 `sec`, and the error estimation requires ≈ 0.11 `sec`.

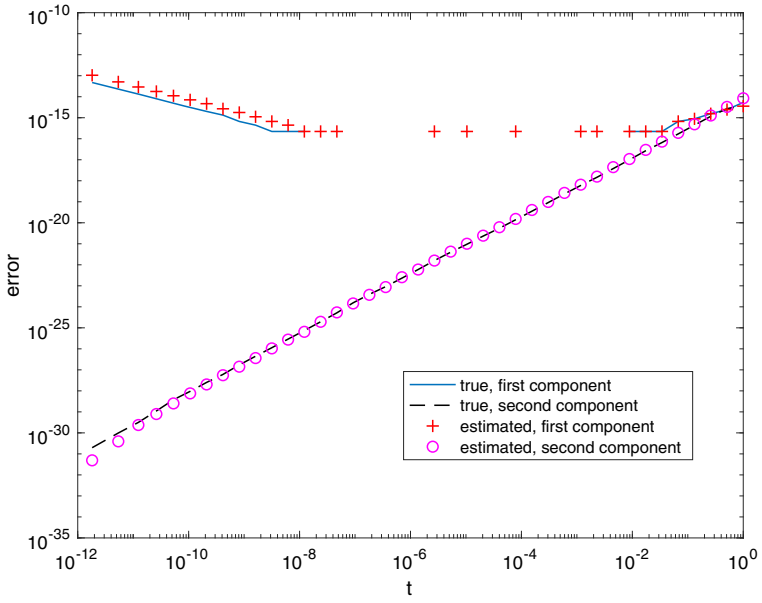

Fig. 5 True and estimated absolute errors for `fhbvm` solving problem (64), $M = 2$

4.4 Example 4

At last, we consider the following fractional Brusselator model:

$$\begin{aligned} y_1^{(0.7)} &= 1 - 4y_1 + y_1^2 y_2, \\ y_2^{(0.7)} &= 3y_1 - y_1^2 y_2, \qquad t \in [0, 5], \\ y(0) &= (1.2,\ 2.8)^\top, \end{aligned} \tag{65}$$

By solving this problem using `fhbvm` with parameter $M = 5$, a graded mesh of 46 points is produced, with $h_1 \approx 6.1 \cdot 10^{-5}$ and $h_{45} \approx 0.98$. The maximum estimated error in the computed solution is less than $3.5 \cdot 10^{-13}$, whereas the phase-plot of the solution is depicted in Fig. 6.

In this case, the evaluation of the solution requires ≈ 0.04 `sec`, and the error estimation requires ≈ 0.14 `sec`.

5 Conclusions

In this paper we have described in full details the implementation of the Matlab© code `fhbvm`, able to solving systems of FDE-IVPs. The code is based on a FHBVM(22,20) method, as described in [8]. We have also provided comparisons with another existing Matlab© code, thus confirming its potentialities. In fact, due to the spectral accuracy in time of the FHBVM(22,20) method, the generated computational mesh, which can

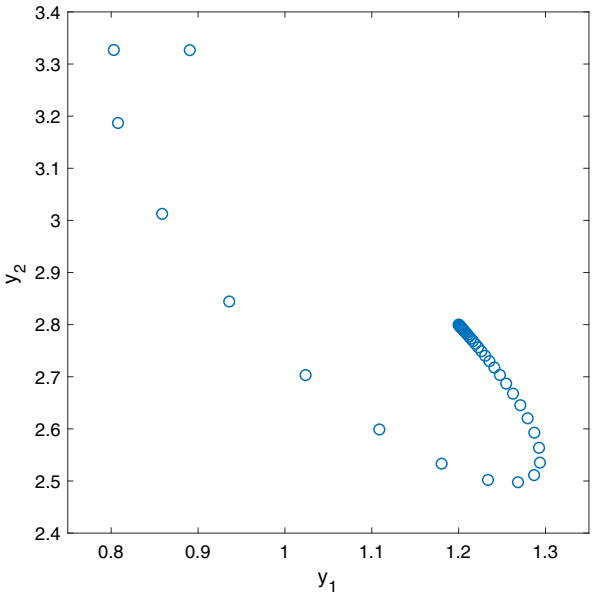

Fig. 6 Phase-plot of the computed solution by using `fhbvm` solving problem (65), $M = 5$

be either a uniform or a graded one, depending on the problem at hand, requires relatively few mesh points. This, in turn, allows to reduce the execution time due to the evaluation of the memory term required at each step.

We plan to further develop the code `fhbvm`, in order to provide approximations at prescribed mesh points, as well as to allow selecting different FHBVM(k, s), $k \geq s$, methods [8]. At last, we plan to extend the code to cope with values of the fractional derivative, α, greater than 1.

Acknowledgements The authors are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

Author Contributions All authors contributed equally to this work.

Funding Open access funding provided by Universitá degli Studi di Firenze within the CRUI-CARE Agreement. No grants were received for conducting this study.

Data Availability All data reported in the manuscript have been obtained by the Matlab© code `fhbvm`, Rel. 2024-03-06, available at the url [40].

Declarations

Conflicts of Interest/Competing Interests The authors declare no conflict of interests, nor competing interests.

Ethical Approval Not applicable.

References

1. Aceto, L., Magherini, C., Novati, P.: Fractional convolution quadrature based on generalized Adams methods. Calcolo **51**, 441–463 (2014). https://doi.org/10.1007/s10092-013-0094-4
2. Amodio, P., Brugnano, L., Iavernaro, F.: Spectrally accurate solutions of nonlinear fractional initial value problems. AIP Conf. Proc. **2116**, 140005 (2019). https://doi.org/10.1063/1.5114132
3. Amodio, P., Brugnano, L., Iavernaro, F.: Analysis of spectral hamiltonian boundary value methods (SHBVMs) for the numerical solution of ODE problems. Numer. Algorithms **83**, 1489–1508 (2020). https://doi.org/10.1007/s11075-019-00733-7
4. Amodio, P., Brugnano, L., Iavernaro, F.: Arbitrarily high-order energy-conserving methods for poisson problems. Numer. Algoritms **91**, 861–894 (2022). https://doi.org/10.1007/s11075-022-01285-z
5. Amodio, P., Brugnano, L., Iavernaro, F.: A note on a stable algorithm for computing the fractional integrals of orthogonal polynomials. Appl. Math. Lett. **134**, 108338 (2022). https://doi.org/10.1016/j.aml.2022.108338
6. Amodio, P., Brugnano, L., Iavernaro, F.: (Spectral) Chebyshev collocation methods for solving differential equations. Numer. Algoritms **93**, 1613–1638 (2023). https://doi.org/10.1007/s11075-022-01482-w

7. Brugnano, L.: Blended block BVMs (B_3VMs): a family of economical implicit methods for ODEs. J. Comput. Appl. Math. **116**, 41–62 (2000). https://doi.org/10.1016/S0377-0427(99)00280-0
8. Brugnano, L., Burrage, K., Burrage, P., Iavernaro, F.: A spectrally accurate step-by-step method for the numerical solution of fractional differential equations. J. Sci. Comput. **99**, 48 (2024). https://doi.org/10.1007/s10915-024-02517-1
9. Brugnano, L., Frasca-Caccia, G., Iavernaro, F., Vespri, V.: A new framework for polynomial approximation to differential equations. Adv. Comput. Math. **48**, 76 (2022). https://doi.org/10.1007/s10444-022-09992-w
10. Brugnano, L., Iavernaro, F.: Line Integral Methods for Conservative Problems. Chapman et Hall/CRC, Boca Raton, FL, USA (2016)
11. Brugnano, L., Iavernaro, F.: Line integral solution of differential problems. Axioms **7**(2), 36 (2018). https://doi.org/10.3390/axioms7020036
12. Brugnano, L., Iavernaro, F.: A general framework for solving differential equations. Ann. Univ. Ferrara Sez. VII Sci. Mat. **68**,243–258 (2022). https://doi.org/10.1007/s11565-022-00409-6
13. Brugnano, L., Iavernaro, F., Trigiante, D.: A note on the efficient implementation of hamiltonian BVMs. J. Comput. Appl. Math. **236**, 375–383 (2011). https://doi.org/10.1016/j.cam.2011.07.022
14. Brugnano, L., Magherini, C.: Blended implementation of block implicit methods for ODEs. Appl. Numer. Math. **42**, 29–45 (2002). https://doi.org/10.1016/S0168-9274(01)00140-4
15. Brugnano, L., Magherini, C.: The BiM code for the numerical solution of ODEs. J. Comput. Appl. Math. **164–165**, 145–158 (2004). https://doi.org/10.1016/j.cam.2003.09.004
16. Brugnano, L., Magherini, C.: Blended implicit methods for solving ODE and DAE problems, and their extension for second order problems. J. Comput. Appl. Math. **205**, 777–790 (2007). https://doi.org/10.1016/j.cam.2006.02.057
17. Brugnano, L., Magherini, C.: Recent advances in linear analysis of convergence for splittings for solving ODE problems. Appl. Numer. Math. **59**, 542–557 (2009). https://doi.org/10.1016/j.apnum.2008.03.008
18. Brugnano, L., Magherini, C., Mugnai, F.: Blended implicit methods for the numerical solution of DAE problems. J. Comput. Appl. Math. **189**, 34–50 (2006). https://doi.org/10.1016/j.cam.2005.05.005
19. Brugnano, L., Montijano, J.I., Iavernaro, F., Randéz, L.: Spectrally accurate space-time solution of hamiltonian PDEs. Numer. Algorithms **81**, 1183–1202 (2019). https://doi.org/10.1007/s11075-018-0586-z
20. Brugnano, L., Montijano, J.I., Randéz, L.: On the effectiveness of spectral methods for the numerical solution of multi-frequency highly-oscillatory hamiltonian problems. Numer. Algorithms **81**, 345–376 (2019). https://doi.org/10.1007/s11075-018-0552-9
21. Cardone, A., Conte, D., Paternoster, B.: A Matlab code for fractional differential equations based on two-step spline collocation methods. In: Fractional Differential Equations, Modeling, Discretization, and Numerical Solvers, Cardone, A., et al. (eds.) Springer INDAM Series, vol. 50, pp. 121–146 (2023). https://doi.org/10.1007/978-981-19-7716-9_8
22. Dahlquist, G., Björk, Å.: Numerical Methods in Scientific Computing. SIAM, Philadelphia, PA, USA (2008)
23. Diethelm, K.: The analysis of fractional differential equations. An application-oriented exposition using differential operators of Caputo type. Lecture Notes in Math, 2004. Springer-Verlag, Berlin, (2010)
24. Diethelm, K., Ford, N.J., Freed, A.D.: A predictor-corrector approach for the numerical solution of fractional differential equations. Nonlinear Dyn. **29**, 3–22 (2002). https://doi.org/10.1023/A:1016592219341
25. Diethelm, K., Ford, N.J., Freed, A.D.: Detailed error analysis for a fractional adams method. Numer. Algorithms **36**, 31–52 (2004). https://doi.org/10.1023/B:NUMA.0000027736.85078.be
26. Garrappa, R.: Numerical evaluation of two and three parameter mittag-leffler functions. SIAM J. Numer. Anal. **53** No. 3, 1350–1369 (2015). https://doi.org/10.1137/140971191
27. Garrappa, R.: Trapezoidal methods for fractional differential equations: theoretical and computational aspects. Math. Comput. Simul. **110**, 96–112 (2015). https://doi.org/10.1016/j.matcom.2013.09.012
28. Garrappa, R.: Numerical solution of fractional differential equations: a survey and a software tutorial. Math. **6**(2), 16 (2018). https://doi.org/10.3390/math6020016
29. Gu, Z.: Spectral collocation method for nonlinear riemann-liouville fractional terminal value problems. J. Compt. Appl. math. **398**, 113640 (2021). https://doi.org/10.1016/j.cam.2021.113640
30. Gu, Z., Kong, Y.: Spectral collocation method for caputo fractional terminal value problems. Numer. Algorithms **88**, 93–111 (2021). https://doi.org/10.1007/s11075-020-01031-3

31. van der Houwen, P.J., de Swart, J.J.B.: Triangularly implicit iteration methods for ODE-IVP solvers. SIAM J. Sci. Comput. **18**, 41–55 (1997). https://doi.org/10.1137/S1064827595287456
32. van der Houwen, P.J., de Swart, J.J.B.: Parallel linear system solvers for runge-kutta methods. Adv. Comput. Math. **7**(1–2), 157–181 (1997). https://doi.org/10.1023/A:1018990601750
33. van der Houwen, P.J., Sommeijer, B.P., de Swart, J.J.: Parallel predictor-corrector methods. J. Comput. Appl. Math. **66**, 53–71 (1996). https://doi.org/10.1016/0377-0427(95)00158-1
34. Li, C., Yi, Q., Chen, A.: Finite difference methods with non-uniform meshes for nonlinear fractional differential equations. J. Comput. Phys. **316**, 614–631 (2016). https://doi.org/10.1016/j.jcp.2016.04.039
35. Lubich, Ch.: Fractional linear multistep methods for abel-volterra integral equations of the second kind. Math. Comp. **45** No. 172, 463–469 (1985). https://doi.org/10.1090/S0025-5718-1985-0804935-7
36. Podlubny, I.: Fractional differential equations. An introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications. Academic Press, Inc., San Diego, CA, (1999)
37. Stynes, M., O'Riordan, E., Gracia, J.L.: Error analysis of a finite difference method on graded meshes for a time-fractional diffusion equation. SIAM J. Numer. Anal. **55**, 1057–1079 (2017). https://doi.org/10.1137/16M1082329
38. Zayernouri, M., Karniadakis, G.E.: Exponentially accurate spectral and spectral element methods for fractional ODEs. J. Comput. Phys. **257**, 460–480 (2014). https://doi.org/10.1016/j.jcp.2013.09.039
39. Mazzia, F., Magherini, C.: Test Set for Initial Value Problem Solvers. Release 2.4, February 2008, Department of Mathematics, University of Bari and INdAM, Research Unit of Bari, Italy, available at: https://archimede.uniba.it/~testset/testsetivpsolvers/
40. https://people.dimai.unifi.it/brugnano/fhbvm/