A MATLAB Implementation of Spline Collocation Methods for Fractional Differential Equations

Angelamaria Cardone(✉), Dajana Conte, and Beatrice Paternoster

Dipartimento di Matematica, Università di Salerno, Fisciano, Italy
{ancardone,dajconte,beapat}@unisa.it

Abstract. The present paper illustrates a MATLAB program for the solution of fractional differential equations. It is based on a spline collocation method on a graded mesh, introduced by Pedas and Tamme in [J. Comput. Appl. Math. **255**, 216–230 (2014)]. This is the first program proposed to implement spline collocation methods for fractional differential equations, and it is one of the few algorithms available in the literature for these functional equations. An explicit formulation of the method is derived, and the computational kernel is a nonlinear system to be solved at each time step. Such system involves some fractional integrals, whose analytical expression is given; their computation requires the knowledge of the coefficients of some polynomials and the evaluation of some special functions. The method is written in a compact matrix form, to improve the efficiency of the MATLAB implementation. The overall algorithm is outlined and then the attention is focused on some routines, which are given. In particular, some MATLAB native routines are used to evaluate special functions and to compute the coefficients of some polynomials. The complete list of the input and output parameters is available. Finally, an example of usage of the MATLAB program on a test problem is provided and some numerical experiments are shown.

Keywords: Fractional differential equations · Spline collocation · Algorithm · Implementation · MATLAB

1 Introduction

The aim of the present paper is to describe a MATLAB implementation of a spline collocation method for the solution of a nonlinear fractional differential equation (FDE):

$$\begin{cases} D^{\alpha} y(t) = f(t, y(t)), \ 0 \le t \le b, \\ y^{(i)}(0) = \gamma_i, \qquad\qquad i = 0, \ldots, n-1, \end{cases} \tag{1}$$

Authors are members of the INdAM Research group GNCS and are supported by GNCS-INDAM project. A. Cardone and Dajana Conte are supported by PRIN2017-MIUR project.

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12949, pp. 387–401, 2021.
https://doi.org/10.1007/978-3-030-86653-2_29

where $n-1<\alpha<n$, $n\in\mathbb{N}$; $b,\gamma_i\in\mathbb{R}$; $b>0$. We assume that f is a continuous real-valued function defined in $[0,b]$ and that the fractional derivative is meant in the Caputo sense: [16,21,32]:

$$D^{\alpha}y(t)=\frac{1}{\Gamma(n-\alpha)}\int_0^t\frac{y^{(n)}(s)}{(t-s)^{\alpha+1-n}}ds.$$

Under suitable hypotheses, there exists a unique continuous solution of the initial value problem (IVP) (1). However the solution is generally not smooth around the initial point 0 [16,21,29,32].

In the last twenty years an increasing number of fractional differential models have been proposed in various fields, e.g. anomalous diffusion [24], anomalous transport models [3], viscoelastic materials [36], option pricing models [34]. Therefore, great attention was paid also to the numerical solution of FDEs and many methods have been proposed so far, e.g. Adomian decomposition methods [14], Petrov-Galerkin methods [25], product integration methods [20], fractional linear multistep methods [23], time-splitting schemes [6], collocation methods [1,11,22,29,30], spectral methods [4,37]. In particular, here we focus our attention on spline collocation methods, which is a very popular class of methods for ODEs and Volterra integral equations [2,7,8,13,15,26]. In the application to FDEs, spline collocation methods are highly accurate and have also good stability properties, as proved in [9–12,27–30].

On the other side, the development of mathematical software for FDEs is at an initial stage. An updated overview is given in [18] and some other references are [19,31,33,35]. In this scenario, our aim is to offer a new software tool based on a spline collocation method proposed in [29] and also studied in [9]. This software was applied in [9,11,12] for numerical experiments, but was nowhere illustrated so far.

The paper is organized as follows. Section 2 illustrates the spline collocation method introduced in [29] and gives a compact matrix form of it, suitable for implementation. We illustrate the mathematical implementation in Sect. 3, i.e. the main program, the auxiliary routines, the special MATLAB functions used in the codes. Input and output parameters are described is in Sect. 4. Section 5 contains an example of use of the MATLAB program and some numerical experiments. Some conclusion are drawn in Sect. 6.

2 The Spline Collocation Method

The solution of (1) can be written as

$$y=J^{\alpha}z+Q, \tag{2}$$

where $z=D^{\alpha}y$ is the solution of the nonlinear equation

$$z=f(t,J^{\alpha}z+Q). \tag{3}$$

In the previous relations:

$$(J^\alpha z)(t) = \frac{1}{\Gamma(\alpha)} \int_0^t (t-s)^{\alpha-1} z(s)\, ds, \quad t > 0, \tag{4}$$

$$Q(t) = \sum_{i=0}^{\lceil \alpha \rceil - 1} \frac{\gamma_i}{i!} t^i, \tag{5}$$

with $\lceil \alpha \rceil$ equal to the smallest integer not less than α.

In [29], authors solve problem (3) by a one-step collocation method and then compute the solution of (1) by (2). In particular, the approximate collocation solution $v(t) \in S_k^{(-1)}(I_N)$, with

$$S_k^{(-1)}(I_N) = \left\{ v : v|_{\sigma_j} \in \pi_k,\, j = 1, \ldots, N \right\}, \tag{6}$$

where π_k is the space of algebraic polynomials of degree not exceeding k.

Given a mesh of points $0 = t_0 < t_1 < \cdots < t_N = b$ with $h_j = t_j - t_{j-1}$ and a set of collocation abscissae $0 \le \eta_1 < \ldots < \eta_m \le 1$, the collocation solution $v(t)$ is represented as follows:

$$v(t) = \sum_{l=1}^{N} \sum_{k=1}^{m} z_{lk} L_{lk}(t), \quad t \in [0, b], \tag{7}$$

where $z_{jk} = v(t_{jk})$ and

$$L_{lk}(t) = \begin{cases} k-\text{th Lagrange fund. pol. wrt to } \{t_{li}\}_{i=1}^m, & t \in [t_{l-1}, t_l] \\ 0 & \text{otherwise} \end{cases}$$

$v(t)$ verifies equation (3) at the collocation points $t_{jk} = t_{j-1} + h_j \eta_k$, $j = 1, \ldots, N, k = 1, \ldots, m$, i.e.:

$$z_{jk} = f\left(t_{jk}, (J^\alpha v)(t_{jk}) + Q(t_{jk})\right), \quad k = 1, \ldots, m, \tag{8}$$

$j = 1, \ldots, N$. By definition of $L_{lk}(t)$, the system (8) may be recast as:

$$z_{jk} = f\left(t_{jk}, \sum_{\mu=1}^{m} (J^\alpha L_{j\mu})(t_{jk}) z_{j\mu} + \sum_{l=1}^{j-1} \sum_{\mu=1}^{m} (J^\alpha L_{l\mu})(t_{jk}) z_{l\mu} + Q(t_{jk})\right), \tag{9}$$

$k = 1, \ldots, m$.

To ensure the fastest error decay, a suitable graded mesh must be adopted, i.e.

$$t_j = b\left(\frac{j}{N}\right)^r, \qquad j = 0, \ldots, N. \tag{10}$$

The best value of the grading exponent $r \ge 1$ depends on the smoothness of the problem data (compare [12,29]).

In [9] it is provided an explicit expression for the fractional integrals appearing in (9). Namely:

$$(J^\alpha L_{j\mu})(t_{jk}) = h_j^\alpha \sum_{\nu=0}^{m-1} a_\nu^{(\mu)} \eta_k^{\nu+\alpha} \frac{\Gamma(1+\nu)}{\Gamma(1+\nu+\alpha)}, \tag{11}$$

$$(J^\alpha L_{l\mu})(t_{jk}) = \frac{1}{\Gamma(\alpha)} h_l^\alpha \sum_{\nu=0}^{m-1} a_\nu^{(\mu)} \left(\frac{t_{j-1} + \eta_k h_j - t_{l-1}}{h_l} \right)^{\nu+\alpha} \mathrm{B}\left(\frac{h_l}{t_{j-1} + \eta_k h_j - t_{l-1}}; 1+\nu, \alpha \right). \tag{12}$$

Function $B(x; b, c)$ is the incomplete beta function:

$$\mathrm{B}(x; a, b) := \int_0^x \sigma^{a-1} (1-\sigma)^{b-1} \, d\sigma,$$

while $\{a_\nu^{(\mu)}\}_{\nu=0}^{m-1}$ are the coefficients of the following Lagrange polynomial:

$$\prod_{\substack{i=1 \\ i \neq \mu}}^{m} \frac{\tau - \eta_i}{\eta_\mu - \eta_i} = \sum_{\nu=0}^{m-1} a_\nu^{(\mu)} \tau^\nu, \qquad \mu = 1, \ldots, m. \tag{13}$$

We observe that the coefficients of a polynomial $p(x) = q(x)r(x)$ are found by convolving the vectors $\mathbf{q}$ and $\mathbf{r}$ of the coefficients of polynomials $q(x)$ and $r(x)$, respectively. Therefore, coefficients $a_\nu^{(\mu)}$ can be computed by iteratively computing the convolution products of vectors $\left[\frac{1}{\eta_\mu - \eta_i}, \frac{-\eta_i}{\eta_\mu - \eta_i} \right]$, $i = 1, \ldots, m$, $i \neq \mu$ (cfr. (13)). This technique is applied in the routine `matrix Lagrange.m` (see Fig. 2).

The matrix form of system (9) is the following

$$\mathbf{z}_j = f\left(\mathbf{t}_j, h_j^\alpha \mathbf{A} \mathbf{z}_j + \sum_{l=1}^{j-1} h_l^\alpha \mathbf{E}^{[l,j]} \mathbf{z}_l + \mathbf{q}_j \right), \tag{14}$$

where

$$\mathbf{z}_j = [z_{j1}, \ldots, z_{jm}]^T, \quad \mathbf{t}_j = [t_{j1}, \ldots, t_{jm}]^T, \quad \mathbf{q}_j = [Q(t_{j1}), \ldots, Q(t_{jm})]^T, \tag{15}$$

$$(f(\mathbf{t}_j, \mathbf{x}))_k = f(t_{jk}, x_k), k = 1, \ldots, m, \mathbf{x} \in \mathbb{R}^m,$$

$$A_{k\mu} = \sum_{\nu=0}^{m-1} a_\nu^{(\mu)} \eta_k^{\nu+\alpha} \frac{\Gamma(1+\nu)}{\Gamma(1+\nu+\alpha)}, \tag{16}$$

$$E_{k\mu}^{[l,j]} = \frac{1}{\Gamma(\alpha)} \sum_{\nu=0}^{m-1} a_\nu^{(\mu)} \left(\frac{t_{j-1} + \eta_k h_j - t_{l-1}}{h_l} \right)^{\nu+\alpha} \mathrm{B}\left(\frac{h_l}{t_{j-1} + \eta_k h_j - t_{l-1}}; 1+\nu, \alpha \right), \tag{17}$$

$k, \mu = 1, \ldots, m.$

Once the approximate solution $v(t)$ of problem (3) has been computed, the numerical solution $y_N \approx y$ of problem (1) is found via (2), i.e.:

$$y_N = J^\alpha v + Q.$$

Therefore

$$y_N(t) = \sum_{\mu=1}^{m} (J^\alpha L_{j\mu})(t) z_{j\mu} + \sum_{l=1}^{j-1} \sum_{\mu=1}^{m} (J^\alpha L_{l\mu})(t) z_{l\mu} + Q(t), \quad t \in [t_{j-1}, t_j]. \tag{18}$$

By setting $\sigma = (t - t_j)/h_j$, the fractional integrals in (18) may be computed similarly as (11) and (12). Then, we obtain:

$$y_N(t) = h_j^\alpha \mathbf{b}^T \mathbf{z}_j + \sum_{l=1}^{j-1} h_l^\alpha (\mathbf{b}^{[l,j]})^T \mathbf{z}_l, \quad t \in [t_{j-1}, t_j], \tag{19}$$

where m-dimensional vectors $\mathbf{b}$ and $\mathbf{g}^{[l,j]}$ are defined as follows:

$$\mathbf{b}_\mu = \sum_{\nu=0}^{m-1} a_\nu^{(\mu)} \sigma^{\nu+\alpha} \frac{\Gamma(1+\nu)}{\Gamma(1+\nu+\alpha)}, \tag{20}$$

$$\mathbf{g}_\mu^{[l,j]} = \frac{1}{\Gamma(\alpha)} \sum_{\nu=0}^{m-1} a_\nu^{(\mu)} \left(\frac{t_{j-1} + \sigma h_j - t_{l-1}}{h_l} \right)^{\nu+\alpha} \mathrm{B}\left(\frac{h_l}{t_{j-1} + \sigma h_j - t_{l-1}}; 1+\nu, \alpha \right), \tag{21}$$

$\mu = 1, \ldots, m.$

3 The MATLAB Implementation

The main program `fcoll.m` is written in Fig. 1, auxiliary functions are shown in Figs. 2, 3, 4, 5, 6 and 7, auxiliary MATLAB built in functions are listed in Table 1. Input and output arguments are described in Sect. 4.

Lines 2 and 3 of the main program `fcoll.m` construct the vector `t` containing the points of the graded mesh (10) and the vector `h` of the stepsizes, by using the MATLAB function `diff`. The line 4 makes use of the auxiliary function `matrix_Lagrange.m` which computes the matrix *Alagr*, whose components contain the coefficients of Lagrange polynomial (13).

$$Alagr(\mu, \nu) = a_{\nu-1}^{(\mu)}.$$

At line 5, function `matrix_A.m` constructs matrix $\mathbf{A}$ (defined in (16)). Vector $\mathbf{b}$ defined in (20), is computed in correspondence of $\sigma = 1$, by function `vector_b.m`, at line 6.

The `for` loop at lines 11–20 constructs and solves the nonlinear system (14); evaluates the approximate solution at the mesh points $y_N(t_0) = \gamma_0$, $y_N(t_1)$, ..., $y_N(t_N)$, by formula (19). The nonlinear system is computed by function `F.m` (see Fig. 7), whose input argument `Bj` (equal to the summation in (14) is constructed by function `lag.m` at line 13, and input argument `Qj` (array equal

```
1 function [t,y]=fcoll(f,b,gam,alpha,eta,r,N)
2     t=b*([0:N]'/N).^r;
3     h=diff(t);
4     Alagr=matrix_Lagrange(eta);
5     A=matrix_A(alpha,eta,Alagr);
6     bvect=vector_b(alpha,Alagr);
7     options=optimset('TolFun',1e-14,'TolX',1e-14,'Display','off');
8     m=length(eta);
9     Z=zeros(m,N); y=zeros(N+1,1); y(1)=gam(1);
10    iniz=feval(f,t(1)+eta*h(1),gam(1)*ones(m,1))
11    for j=1:N
12       tj=t(j)+eta*h(j);
13       Bj=lag(Z,Alagr,t,h,eta,alpha,j);
14       Qj=Q(tj,alpha,gam);
15       Z(:,j)=fsolve(@system_F,iniz,options,f,tj,A,Bj,Qj,h,j,alpha);
16       Wj=lag_y(Z,Alagr,t,h,alpha,j);
17       Qjp1=Q(t(j+1),alpha,gam);
18       y(j+1)=h(j)^(alpha)*(bvect')*Z(:,j)+Wj+Qjp1;
19       iniz=Z(:,j);
20    end
21 end
```

Fig. 1. Main program `fcoll.m`

to the vector $\mathbf{q}_j$ defined in (15)) is constructed at line 14 by function `Q.m`. The nonlinear system (14) is solved by MATLAB inbuilt function `fsolve`, with high accuracy requested (10^{-14}) and initial value equal to the approximate solution in the previous time step. Finally, the approximate solution is derived by (19), where the summation is computed by function `lag_y.m`.

In all the functions, the formulation of the method in terms of products of vectors and matrices was adopted, to obtain a gain in efficiency of the overall algorithm.

4 Input and Output Parameters

The MATLAB code fcoll.m has the input arguments `f,b,gam,alpha,eta,r,N` and the output arguments `t,y`. The type and the meaning of each of them are illustrated in Subsects. 4.1 and 4.2.

4.1 Input Parameters

1. **f** - function handle or string containing name of m-file
   f must return the value of the function $f(t, y)$ at a given point (t, y).

```
function [Alagr]=matrix_Lagrange(eta)
   m=length(eta);
   mu=1;
   a=[1/(eta(mu)-eta(2)),-eta(2)/(eta(mu)-eta(2))];
   for i=3:m
       b=[1/(eta(mu)-eta(i)),-eta(i)/(eta(mu)-eta(i))];
       a=conv(a,b);
   end
   Alagr(mu,:)=a;
   for mu=2:m
       a=[1/(eta(mu)-eta(1)),-eta(1)/(eta(mu)-eta(1))];
       for i=[2:mu-1,mu+1:m]
           b=[1/(eta(mu)-eta(i)),-eta(i)/(eta(mu)-eta(i))];
           a=conv(a,b);
       end
       Alagr(mu,:)=a;
    end
    Alagr=fliplr(Alagr);
 end
```

Fig. 2. Function `matrix_Lagrange.m`

```
function[A]=matrix_A(alpha,eta,Alagr)
   m=length(eta);
   interv=[0:m-1];
   w=gamma(interv+1)./gamma(interv+1+alpha);
   A=((eta.^((alpha+interv))).*w)*Alagr';
end
```

Fig. 3. Function `matrix_A.m`

```
function[b]=vector_b(alpha,Alagr)
   m=size(Alagr,1);
   interv=[0:m-1]';
   w=gamma(interv+1)./gamma(interv+1+alpha);
   b=Alagr*((1.^((alpha+interv))).*w);
end
```

Fig. 4. Function `vector_b.m`

```
function Bj=lag(Z,Alagr,t,h,eta,alpha,j)
  m=length(eta);
  Bj=zeros(m,1);
  interv=[0:m-1];
  w=gamma(interv+1)./gamma(interv+1+alpha);
  for lambda=1:j-1
      d=(t(j)-t(lambda)+eta*h(j))/h(lambda);
      for k=1:m
          W(k,:)=betainc(1/d(k),1+interv,alpha);
      end
      E=(((d.^((alpha+interv))).*w).*W)*Alagr';
      Bj=Bj+(h(lambda)^alpha)*E*Z(:,lambda);
  end
end
```

Fig. 5. Function `lag.m`

```
function z=Q(t,alpha,gam)
 interv=0:ceil(alpha)-1;
 z=sum(((t.^interv)./factorial(interv)).*(gam(interv+1).'),2);
end
```

Fig. 6. Function `Q.m`

```
function F=system_F(x,f,tj,A,Bj,Qj,h,j,alpha)
 F=x-feval(f,tj,h(j)^(alpha)*A*x+Bj+Qj);
end
```

Fig. 7. Function `system_F.m`

```
function Wj=lag_y(Z,Alagr,t,h,alpha,j)
 Wj=0;
 m=size(Alagr,1);
 interv=[0:m-1]';
 w=gamma(interv+1)./gamma(interv+1+alpha);
 for lambda=1:j-1
     d=(t(j+1)-t(lambda))/h(lambda);
     W=betainc(1/d,1+interv,alpha);
     g=Alagr*(((d.^((alpha+interv))).*w).*W);
     Wj=Wj+(h(lambda)^alpha)*g'*Z(:,lambda);
 end
end
```

Fig. 8. Function `lag_y.m`

Table 1. Auxiliary MATLAB functions adopted in the algorithm in alphabetical order

Function	Task
betainc	The incomplete beta function
ceil	Function $\lceil \cdot \rceil$
conv	Convolution of two vectors
diff	Differences between adjacent elements of array
gamma	The gamma function
factorial	The factorial
fliplr	Flips array left to right
fsolve	Solves a nonlinear system

```
[result] = f(t,y)
```

Input Parameters

- **t** - double scalar
  The current value of the independent variable t.
- **y** - double scalar
  The current value of the independent variable y.

Output Parameters

- **result** - double scalar
  The value of $f(t, y)$.

2. **b** - double scalar
   b, the end point of the integration interval $[0, b]$.
3. **gam** - double array
   The vector of the initial values $[\gamma_0, \ldots, \gamma_{n-1}]^T$ of the IVP (1).
   Constraint: The length n of the array **gam** should be equal to $\lceil \alpha \rceil$.
4. **alpha** - double scalar
   α, the fractional index.
5. **eta(m)** - double array
   eta is equal to the vector $[\eta_1, \ldots, \eta_m]^T$ of the collocation parameters.
   Constraint: $0 \leq$ **eta(1)** $\leq \cdots \leq$**eta(m)**.
6. **r** - double scalar
   r, the grading exponent.
7. **N** - double scalar
   N, the number of mesh points.

4.2 Output Parameters

1. **t(N+1)** - double array
   t is equal to the graded mesh $[t_0, \ldots, t_N]^T$, defined in (10).

2. **Y(N+1)** - double array
   $Y(j)$ is the approximate value of the solution $y(t_j)$.

5 Numerical Examples

For the users' convenience, we report an example of usage of our program, on the Test problem 1 (see Figs. 9 and 10). Then, we compare the exact and the numerical solution, compute the absolute error for Test problems 1, 2 and 3 (see Figs. 11, 12 and 13). In all cases, we applied the spline collocation method (14) (19) with $m = 2$, collocation parameters $\eta = \left[\frac{3\pm\sqrt{3}}{6}\right]$, the grading exponent r suitably chosen in order to obtain superconvergence (compare [9,12,29]). The number of the mesh points is set to $N = b \cdot 2^5$.

Test Problem 1 [29]

$$D^{1/2}y(t) = y^2(t) + \frac{1}{\Gamma(1.5)}t^{0.5} - t^2,\ t \in [0,1] \quad y(0) = 0,$$

The exact solution is $y(t) = t$. The value of the grading exponent is $r = 2.5$.

Test Problem 2 [17]

$$D^{\alpha}y(t) = \frac{40320}{\Gamma(9-\alpha)}t^{8-\alpha} - 3\frac{\Gamma(5+\frac{\alpha}{2})}{\Gamma(5-\frac{\alpha}{2})}t^{4-\frac{\alpha}{2}} + \left(\frac{3}{2}t^{\frac{\alpha}{2}} - t^4\right)^3 \quad t \in [0,1],$$
$$+\frac{9}{4}\Gamma(\alpha+1) - (y(t))^{\frac{3}{2}},$$
$$y(0) = 0,$$

```
1 f=@(t,y) y.^2+1/gamma(1.5)*t.^0.5-t.^2;
2 b=10;
3 gam=[0];
4 alpha=1/2;
5 eta=[(3-sqrt(3))/6; 1-((3-sqrt(3))/6)];
6 r=2.6;
7 N=b/2^-5;
8 [t,y]=fcoll(f,b,gam,alpha,eta,r,N);
9 err=abs(y(end)-b)
```

Fig. 9. Script: test_example.m

```
>> test_example

err =

   1.2416e-05
```

Fig. 10. Output: test_example.m

$\alpha = 1/2$. Exact solution $y = t^8 - 3t^{4+\alpha/2} + \frac{9}{4}t^\alpha$. The method is applied with $r = 1.8$

Test Problem 3 [5]

$$D^\alpha y(t) = \lambda y + \rho y(1 - y^2) + g(t), \quad t \in (0, 8], \qquad y(0) = 2,$$

with $\alpha = 0.15$, $\lambda = -3$, $\rho = 0.8$. $g(t)$ is set in such a way that the solution is $y(t) = y_0 + \sum_{k=1}^{6} t^{\sigma_k}$, with $\sigma_k = k\alpha$, $k = 1, \ldots, 5$, and $\sigma_6 = 2 + \alpha$. The chosen value of the grading exponent is $r = 7.17$.

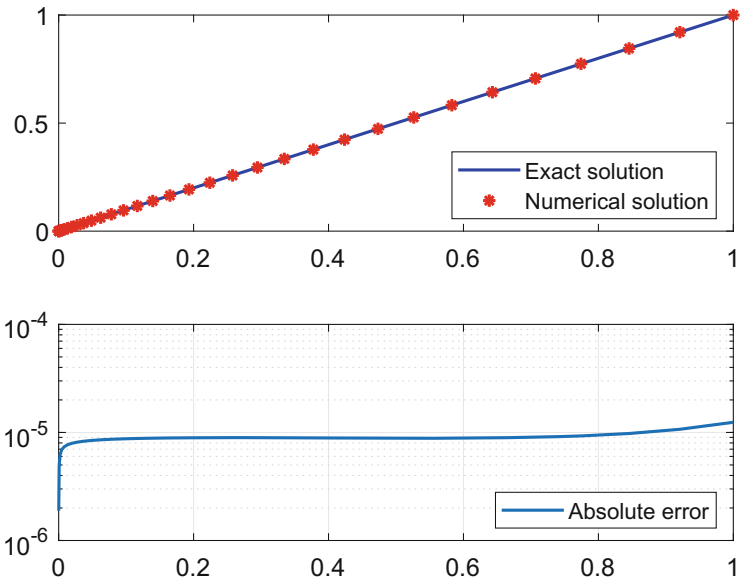

Fig. 11. Test example 1. Numerical and exact solution (on the top), absolute error (at the bottom).

6 Conclusions

We proposed a MATLAB algorithm to solve FDEs by spline collocation methods. We obtained an explicit formulation of the method, and explained how to compute some fractional integrals with high accuracy. To exploit MATLAB efficiency with matrix manipulation, we derived a matrix formulation of the method. We listed all the codes of the program, the input and output parameters. An example of use was included and some numerical examples were provided.

A further development of this work may regard the implementation of two-step collocation methods [10,11], which will require a suitable starting procedure, too.

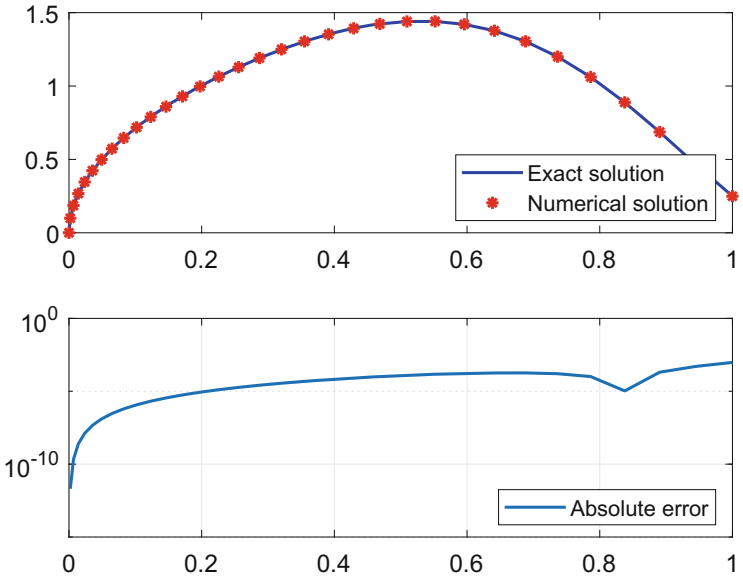

Fig. 12. Test example 2. Numerical and exact solution (on the top), absolute error (at the bottom).

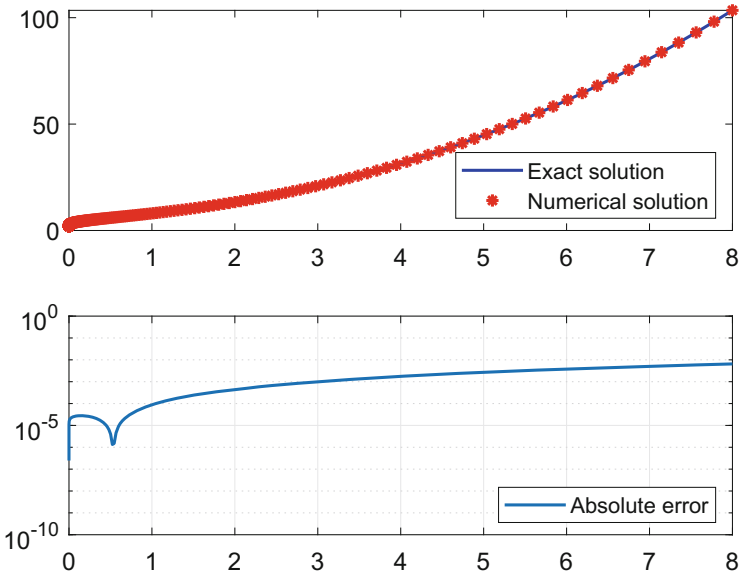

Fig. 13. Test example 3. Numerical and exact solution (on the top), absolute error (at the bottom).

References

1. Blank, L.: Numerical treatment of differential equations of fractional order. Technical report, University of Manchester, Department of Mathematics (1996). Numerical Analysis Report
2. Brunner, H.: Collocation methods for Volterra integral and related functional differential equations. Cambridge Monographs on Applied and Computational Mathematics, vol. 15. Cambridge University Press, Cambridge (2004). https://doi.org/10.1017/CBO9780511543234
3. Bueno-Orovio, A., Kay, D., Grau, V., Rodriguez, B., Burrage, K.: Fractional diffusion models of cardiac electrical propagation: role of structural heterogeneity in dispersion of repolarization. J. R. Soc. Interface **11**(97), 20140352 (2014)
4. Burrage, K., Cardone, A., D'Ambrosio, R., Paternoster, B.: Numerical solution of time fractional diffusion systems. Appl. Numer. Math. **116**, 82–94 (2017). https://doi.org/10.1016/j.apnum.2017.02.004
5. Cao, W., Zeng, F., Zhang, Z., Karniadakis, G.E.: Implicit-explicit difference schemes for nonlinear fractional differential equations with nonsmooth solutions. SIAM J. Sci. Comput. **38**(5), A3070–A3093 (2016). https://doi.org/10.1137/16M1070323
6. Cao, W., Zhang, Z., Karniadakis, G.E.: Time-splitting schemes for fractional differential equations I: smooth solutions. SIAM J. Sci. Comput. **37**(4), A1752–A1776 (2015). https://doi.org/10.1137/140996495
7. Cardone, A., Conte, D., Paternoster, B.: A family of multistep collocation methods for Volterra Integro-differential equations. AIP Conf. Proc. **1168**, 358–361 (2009). https://doi.org/10.1063/1.3241469
8. Cardone, A., Conte, D.: Multistep collocation methods for Volterra Integro-differential equations. Appl. Math. Comput. **221**, 770–785 (2013). https://doi.org/10.1016/j.amc.2013.07.012
9. Cardone, A., Conte, D.: Stability analysis of spline collocation methods for fractional differential equations. Math. Comput. Simulat. **178**, 501–514 (2020). https://doi.org/10.1016/j.matcom.2020.07.004
10. Cardone, A., Conte, D., Paternoster, B.: Stability analysis of two-step spline collocation methods for fractional differential equations, submitted
11. Cardone, A., Conte, D., Paternoster, B.: Two-step collocation methods for fractional differential equations. Discrete Contin. Dyn. Syst. Ser. B **23**(7), 2709–2725 (2018). https://doi.org/10.3934/dcdsb.2018088
12. Cardone, A., Conte, D., Paternoster, B.: Numerical treatment of fractional differential models. In: Abdel Wahab, M. (ed.) FFW 2020 2020. LNME, pp. 289–302. Springer, Singapore (2021). https://doi.org/10.1007/978-981-15-9893-7_21
13. Conte, D., Paternoster, B.: Multistep collocation methods for Volterra Integral equations. Appl. Numer. Math. **59**(8), 1721–1736 (2009). https://doi.org/10.1016/j.apnum.2009.01.001
14. Daftardar-Gejji, V., Jafari, H.: Adomian decomposition: a tool for solving a system of fractional differential equations. J. Math. Anal. Appl. **301**(2), 508–518 (2005). https://doi.org/10.1016/j.jmaa.2004.07.039
15. D'Ambrosio, R., Paternoster, B.: Multivalue collocation methods free from order reduction. J. Comput. Appl. Math. **387**, 112515 (2021). https://doi.org/10.1016/j.cam.2019.112515
16. Diethelm, K.: The Analysis of Fractional Differential Equations. Lecture Notes in Mathematics, vol. 2004. Springer, Berlin (2010)

17. Diethelm, K., Ford, N.J., Freed, A.D.: Detailed error analysis for a fractional Adams method. Numer. Algorithms **36**(1), 31–52 (2004). https://doi.org/10.1023/B:NUMA.0000027736.85078.be
18. Garrappa, R.: Numerical solution of fractional differential equations: a survey and a software tutorial. Mathematics **6**(2), 16 (2018). https://doi.org/10.3390/math6020016
19. Garrappa, R.: Short Tutorial: Solving Fractional Differential Equations by Matlab Codes. Department of Mathematics University of Bari, Italy (2014)
20. Garrappa, R., Popolizio, M.: On accurate product integration rules for linear fractional differential equations. J. Comput. Appl. Math. **235**(5), 1085–1097 (2011). https://doi.org/10.1016/j.cam.2010.07.008
21. Kilbas, A.A., Srivastava, H.M., Trujillo, J.J.: Theory and Applications of Fractional Differential Equations, North-Holland Mathematics Studies, vol. 204. Elsevier Science B.V, Amsterdam (2006)
22. Li, X.: Numerical solution of fractional differential equations using cubic B-spline wavelet collocation method. Commun. Nonlinear Sci. Numer. Simul. **17**(10), 3934–3946 (2012). https://doi.org/10.1016/j.cnsns.2012.02.009
23. Lubich, C.: Fractional linear multistep methods for Abel-Volterra integral equations of the second kind. Math. Comp. **45**(172), 463–469 (1985). https://doi.org/10.2307/2008136
24. Metzler, R., Klafter, J.: The random walk's guide to anomalous diffusion: a fractional dynamics approach. Phys. Rep. **339**(1), 77 (2000). https://doi.org/10.1016/S0370-1573(00)00070-3
25. Mustapha, K., Abdallah, B., Furati, K.: A discontinuous Petrov-Galerkin method for time-fractional diffusion equations. SIAM J. Numer. Anal. **52**(5), 2512–2529 (2014). https://doi.org/10.1137/140952107
26. Paternoster, B.: Phase-fitted collocation-based Runge-Kutta-Nystrom method. Appl. Numer. Math. **35**(4), 339–355 (2000). https://doi.org/10.1016/S0168-9274(99)00143-9
27. Pedas, A., Tamme, E.: On the convergence of spline collocation methods for solving fractional differential equations. J. Comput. Appl. Math. **235**(12), 3502–3514 (2011). https://doi.org/10.1016/j.cam.2010.10.054
28. Pedas, A., Tamme, E.: Spline collocation methods for linear multi-term fractional differential equations. J. Comput. Appl. Math. **236**(2), 167–176 (2011). https://doi.org/10.1016/j.cam.2011.06.015
29. Pedas, A., Tamme, E.: Numerical solution of nonlinear fractional differential equations by spline collocation methods. J. Comput. Appl. Math. **255**, 216–230 (2014). https://doi.org/10.1016/j.cam.2013.04.049
30. Pedas, A., Tamme, E.: Spline collocation for nonlinear fractional boundary value problems. Appl. Math. Comput. **244**, 502–513 (2014). https://doi.org/10.1016/j.amc.2014.07.016
31. Petrás, I.: Fractional derivatives, fractional integrals, and fractional differential equations in MATLAB. In: Assi, A.H. (ed.) Engineering Education and Research Using MATLAB, Chapter 10. IntechOpen, Rijeka (2011). https://doi.org/10.5772/19412
32. Podlubny, I.: Fractional differential equations, Mathematics in Science and Engineering, vol. 198. Academic Press Inc, San Diego, CA (1999). An introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications

33. Sowa, M., Kawala-Janik, A., Bauer, W.: Fractional differential equation solvers in Octave/MATLAB. In: 2018 23rd International Conference on Methods & Models in Automation & Robotics (MMAR), pp. 628–633. IEEE (2018)
34. Wang, W., Chen, X., Ding, D., Lei, S.L.: Circulant preconditioning technique for barrier options pricing under fractional diffusion models. Int. J. Comput. Math. **92**(12), 2596–2614 (2015). https://doi.org/10.1080/00207160.2015.1077948
35. Wei, S., Chen, W.: A MATLAB toolbox for fractional relaxation-oscillation equations. arXiv preprint arXiv:1302.3384 (2013)
36. Yang, W., Chen, Z.: Fractional single-phase lag heat conduction and transient thermal fracture in cracked viscoelastic materials. Acta Mechanica **230**(10), 3723–3740 (2019). https://doi.org/10.1007/s00707-019-02474-z
37. Zayernouri, M., Karniadakis, G.E.: Exponentially accurate spectral and spectral element methods for fractional ODEs. J. Comput. Phys. **257**(part A), 460–480 (2014). https://doi.org/10.1016/j.jcp.2013.09.039