DOI: 10.5604/01.3001.0010.5208

NUMERICAL COMPUTATIONS OF THE FRACTIONAL DERIVATIVE IN IVPS, EXAMPLES IN MATLAB AND MATHEMATICA

Marcin Sowa
Silesian University of Technology, Faculty of Electrical Engineering, Institute of Electrical Engineering and Computer Science

Abstract. *The paper concerns a numerical method that deals with the computations of the fractional derivative in Caputo and Riemann-Liouville definitions. The method can be applied in time stepping processes of initial value problems. The name of the method is SubIval, which is an acronym of its previous name – the subinterval-based method. Its application in solving systems of fractional order state equations is presented. The method has been implemented into an ActiveX DLL. Exemplary MATLAB and Mathematica codes are given, which provide guidance on how the DLL can be used.*

Keywords: fractional calculus, numerical analysis, circuit analysis, integro-differential equations

OBLICZENIA NUMERYCZNE POCHODNEJ UŁAMKOWEGO RZĘDU W ZAGADNIENIACH POCZĄTKOWYCH, PRZYKŁADY W PROGRAMACH MATLAB I MATHEMATICA

Streszczenie. *Artykuł dotyczy numerycznej metody, którą wykorzystać można do obliczeń pochodnej ułamkowego rzędu w definicji Caputo i Riemanna-Liouville'a. Metoda ta może być wykorzystana przy rozwiązywaniu zagadnień początkowych. Metoda nosi nazwę SubIval, co jest akronimem jej poprzedniej, anglojęzycznej nazwy „subinterval-based method" (metoda podprzedziałów). Przedstawiono jej zastosowanie w rozwiązywaniu równań stanu ułamkowego rzędu. Metoda została zaimplementowana w bibliotece DLL z obsługą ActiveX. Przedstawiono przykładowe kody obliczeniowe (w oprogramowaniach MATLAB i Mathematica), które zawierają wskazówki dotyczące zastosowania biblioteki.*

Słowa kluczowe: pochodna ułamkowego rzędu, analiza numeryczna, analiza obwodów, równania całkowo-różniczkowe

Introduction

The increasing popularity of fractional calculus in science is owed to its many applications e.g. in circuit theory [8, 19, 20], control theory [16], fractional order filter design [10], electromagnetic field analysis [6] and temperature field computations [3]. Fractional calculus introduces the concept of a fractional order derivative and integral (or, in general, an integro-derivative). Among the many definitions that can be found in literature [9], the paper concerns the application of the most commonly used fractional derivatives, which are the Riemann-Liouville definition [14] and that of Caputo [4]. The first, for $\alpha \in (0, 1)$, is:

$$ {}^{\mathrm{RL}}_{t_\mathrm{a}}\mathrm{D}^{\alpha}_{t_\mathrm{b}} x(t) = \frac{1}{\Gamma(1-\alpha)} \frac{\mathrm{d}}{\mathrm{d}t} \int_{t_\mathrm{a}}^{t_\mathrm{b}} \frac{x(\tau)}{(t-\tau)^{\alpha}} \mathrm{d}\tau. \tag{1} $$

For that same range Caputo's definition of the fractional derivative is defined by:

$$ {}^{\mathrm{C}}_{t_\mathrm{a}}\mathrm{D}^{\alpha}_{t_\mathrm{b}} x(t) = \frac{1}{\Gamma(1-\alpha)} \int_{t_\mathrm{a}}^{t_\mathrm{b}} \frac{x^{(1)}(\tau)}{(t-\tau)^{\alpha}} \mathrm{d}\tau. \tag{2} $$

Moreover, in the most common occurrence – the fractional derivative appears as a derivative in time, expressing a component with memory.

The various successes in attempts to apply the fractional time derivative are not the only reason why its study has been very popular of late. The appearance of the fractional time derivative introduces a component, which adds some additional computational problems. Recently, however, many methods have been introduced, which allow to deal with the computations of the fractional derivative. Lead researchers mention:

- the Adomian decomposition method [13],
- the CAS wavelet method [18],
- the differential transform method [2],
- the Taylor expansion method [7],
- the collocation method [17],
- in general – Fractional Linear Multistep Methods [12],
- methods that base on the fractional difference operator [5, 11].

The development of clearly described methods dealing with fractional derivative computations allows to conveniently study the application of the fractional derivative.

The current paper concerns the application of a method called SubIval, which originates from it being called the **sub**inter**val**-based method in its first introduction in [21]. It is a numerical method, generally dealing with the computation of the fractional derivative in time stepping processes of initial value problems.

1. Method overview

To better describe the fundamentals of SubIval – the following notation is used for Caputo's fractional derivative:

$$ {}^{\mathrm{C}}\mathrm{d}^{\alpha}_{\Xi} x(t) = {}^{\mathrm{C}}_{t_\mathrm{a}}\mathrm{D}^{\alpha}_{t_\mathrm{b}} x(t), \tag{3} $$

while for the Riemann-Liouville fractional derivative:

$$ {}^{\mathrm{RL}}\mathrm{d}^{\alpha}_{\Xi} x(t) = {}^{\mathrm{RL}}_{t_\mathrm{a}}\mathrm{D}^{\alpha}_{t_\mathrm{b}} x(t), \tag{4} $$

where instead of the integration bounds t_a and t_b the integration interval $\Xi = [t_\mathrm{a}, t_\mathrm{b}]$ is referenced. For equations that are valid for both fractional derivative definitions one can then use the notation $\mathrm{d}^{\alpha}_{\Xi} x(t)$, skipping the left side indices.

When dealing with an IVP, in a computed time step, the total integration interval $\Xi_{\mathrm{tot}} = [t_0, t_{\mathrm{now}}]$ is divided into $M = S + 1$ consecutive subintervals Ξ_s:

$$ \Xi_s = [t_{s,\mathrm{start}}, t_{s,\mathrm{end}}], \quad s > 1 : t_{s,\mathrm{start}} = t_{s-1,\mathrm{end}}, \tag{5} $$

with $s = 1, 2, \ldots S + 1, M$. The partition into subintervals yields:

$$ \mathrm{d}^{\alpha}_{\Xi_{\mathrm{tot}}} x(t) = \sum_{s=1}^{S} \mathrm{d}^{\alpha}_{\Xi_s} x(t) + \mathrm{d}^{\alpha}_{\Xi_M} x(t). \tag{6} $$

For each of the subintervals an interpolation is performed on the nodes $t_{s,1}, t_{s,2}, \ldots t_{s,n_s}$ of other subintervals denoted by Θ_s (such that $\Xi_s \subseteq \Theta_s$) resulting in local polynomials $\tilde{x}_s$. Equation (6) then yields:

$$ \mathrm{d}^{\alpha}_{\Xi_{\mathrm{tot}}} x(t) \approx \sum_{s=1}^{S} \mathrm{d}^{\alpha}_{\Xi_s} \tilde{x}_s(t) + \mathrm{d}^{\alpha}_{\Xi_M} \tilde{x}_M(t), \tag{7} $$

where:

$$ \tilde{x}_s(t) = \sum_{i=1}^{n_s} x_{s,i} L_{s,i}(\xi_s), \quad \xi_s = c_s (t - t_{s,1}), \tag{8} $$

with $L_{s,i}$ being Lagrange basis polynomials:

$$ L_{s,i}\left(c_s (t_{s,k} - t_{s,1})\right) = \begin{cases} 0 & \text{if} \quad i \neq k, \\ 1 & \text{if} \quad i = k. \end{cases} \tag{9} $$

defined on axes of local variables ξ_s (introduced in order to avoid numerical errors), with c_s being the normalizing coefficient:

$$ c_s = \frac{1}{(t_{s,n_s} - t_{s,1})}. \tag{10} $$

An assumption is made that only Θ_M contains the rightmost time node $t = t_{\mathrm{now}}$ (whose variables are treated implicitly). This

allows to separate the unknown value $x(t_{\text{now}}) = x_{M,n_M}$ from the terms dependent on known, previous values of x:

$$\begin{aligned} \mathrm{d}^{\alpha}_{\Xi_{\text{tot}}} x(t) \approx & \sum_{s=1}^{S} \mathrm{d}^{\alpha}_{\Xi_s} \sum_{i=1}^{n_s} x_{s,i} L_{s,i}(c_s(t-t_{s,1})) + \\ & \mathrm{d}^{\alpha}_{\Xi_M} \sum_{i=1}^{n_M-1} x_{M,i} L_{M,i}(c_M(t-t_{M,1})) + \\ & \mathrm{d}^{\alpha}_{\Xi_M} x_{M,n_M} L_{M,n_M}(c_M(t-t_{M,1})). \end{aligned} \tag{11}$$

When extracting the node values from the integrodifferentiations one obtains the relation:

$$\mathrm{d}^{\alpha}_{\Xi_{\text{tot}}} x(t) \approx a x_{M,n_M} + b, \tag{12}$$

where:

$$\begin{aligned} b = & \sum_{s=1}^{S} \mathrm{d}^{\alpha}_{\Xi_s} \sum_{i=1}^{n_s} x_{s,i} L_{s,i}(c_s(t-t_{s,1})) + \\ & \mathrm{d}^{\alpha}_{\Xi_M} \sum_{i=1}^{n_M-1} x_{M,i} L_{M,i}(c_M(t-t_{M,1})), \end{aligned} \tag{13}$$

and:

$$a = \mathrm{d}^{\alpha}_{\Xi_M} L_{M,n_M}(c_M(t-t_{M,1})). \tag{14}$$

Finally, when knowing the Lagrange polynomial coefficients one can use the formula for monomial integrodifferentiation:

$${}^{\mathrm{C}}\mathrm{d}^{\alpha}_{\Xi_s}(g\xi_s^k) = \frac{k g c_s^k (\Delta T_s)^{k-\alpha}}{\Gamma(1-\alpha)} B_{\frac{\Delta t_{\text{loc},s}}{\Delta T_s}}(k, 1-\alpha) \tag{15}$$

for Caputo's fractional derivative, where the auxiliary variables:

$$\Delta t_{\text{loc,s}} = t_{s,\text{end}} - t_{s,\text{start}}, \tag{16}$$

and:

$$\Delta T_s = t - t_{s,\text{start}}, \tag{17}$$

while the incomplete beta function can be computed according to the formula:

$$\begin{aligned} & B_{\rho}(k, 1-\alpha) = \\ & \frac{\Gamma(k)\Gamma(1-\alpha)}{\Gamma(k+1-\alpha)} \left(1-(1-\rho)^{1-\alpha} \sum_{j=0}^{k-1} \frac{\rho^j \prod_{i=0}^{j-1}(1-\alpha+i)}{j!}\right), \end{aligned} \tag{18}$$

If the Riemann-Liouville definition is used then the above formulae can also be used with the difference that the following coefficient needs to be added:

$$b_{\text{RL}} = \frac{x(t_0)}{\Gamma(1-\alpha)} (t_{\text{now}} - t_0)^{-\alpha}, \tag{19}$$

connecting both of the fractional derivative definitions [20]. In conclusion – when for Caputo's derivative one obtains:

$${}^{\mathrm{C}}\mathrm{d}^{\alpha}_{\Xi_{\text{tot}}} x(t) \approx x_{M,n_M} a + b_{\mathrm{C}}, \tag{20}$$

then for the Riemann-Liouville fractional derivative:

$${}^{\text{RL}}\mathrm{d}^{\alpha}_{\Xi_{\text{tot}}} x(t) \approx x_{M,n_M} a + b_{\mathrm{C}} + b_{\text{RL}}. \tag{21}$$

So far it has been explained how to obtain the approximation (12) in each considered time step, after the subintervals had been established. The manner in which the subintervals are established is determined by the algorithm presented in figure 1. It considers the following types of subinterval pairs:

- **moving** subinterval pair (or simply the moving subinterval, since $\Theta_M = \Xi_M$) is the rightmost interval, which occupies $n_M = p_{\text{mov}} + 1$ solution nodes (along with the implicit node $t = t_{\text{now}}$; p_{mov} is the polynomial order),
- **built** subinterval pair – appears when the ones existing are not enough to cover all the solution nodes (i.e. if a maximum polynomial order p_s had been reached) then a new subinterval pair is built on the left side of the moving subinterval on the time axis,
- the subinterval pair is classified as a **closed** subinterval pair when its Θ_s can no longer be expanded, yet $\Theta_s \neq \Xi_s$,
- **sealed** is the final state of each newly built subinterval pair, a closed subinterval pair becomes a sealed pair when finally $\Theta_s = \Xi_s$.

Typically the maximum polynomial order p_s can be the same for all subinterval pairs. It's variability (through adaptivity) has not yet been discussed and is a subject for future research.

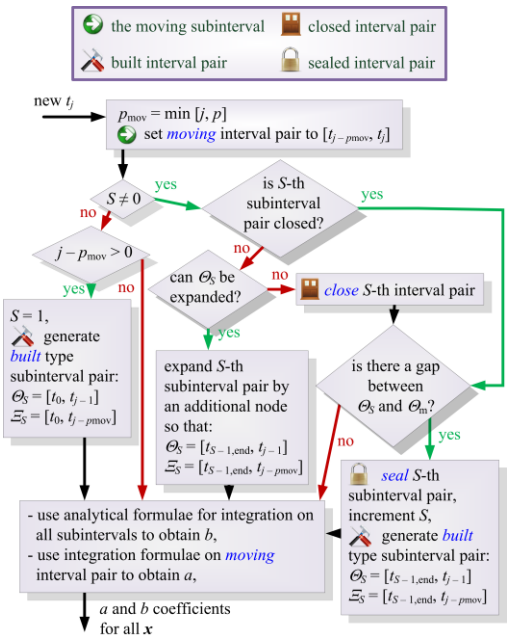


Fig. 1. The algorithm by which the subinterval pairs are established in SubIval

An example of how the subinterval dynamics are performed for a fixed maximum polynomial order p (which applies for all built subinterval pairs and the moving subinterval) is presented in figure 2.

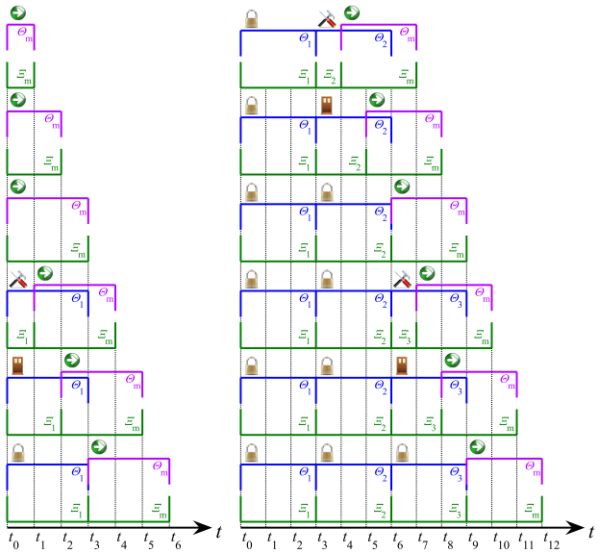


Fig. 2. Subinterval dynamics for a fixed maximum polynomial order p = 3, valid for all built subintervals and the moving subinterval

A time stepping solver does not need to contain only one subinterval dynamics process – e.g. it is possible to perform two of them in cooperation, leading to a predictor-corrector method [21].

2. Relation to state equations

This chapter gives directions on how SubIval can be applied to solve fractional order state equations. The following system is considered:

$$\mathrm{d}^{\alpha}_{\Xi} \boldsymbol{x}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{v}(t), \tag{22}$$

where the left-hand side contains a vector of the fractional derivatives of the state variables:

$$\mathbf{d}^{\boldsymbol{\alpha}}_{\Xi}\boldsymbol{x}(t)=[\mathrm{d}^{\alpha_1}_{\Xi}x_1(t)\quad \mathrm{d}^{\alpha_2}_{\Xi}x_2(t)\quad \ldots\quad \mathrm{d}^{\alpha_n}_{\Xi}x_n(t)]^{\mathrm{T}}. \tag{23}$$

When SubIval is applied, for a currently computed state vector $\boldsymbol{x}_j$, in the time instance t_j, the left-hand side turns equation (22) into:

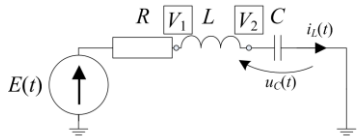
$$\boldsymbol{a}_{\mathrm{diag}}\boldsymbol{x}_j+\boldsymbol{b}=\boldsymbol{A}\boldsymbol{x}_j+\boldsymbol{B}\boldsymbol{v}(t_j), \tag{24}$$

where a_{diag} is a diagonal matrix filled with the a coefficients obtained for each respective state variable; $\boldsymbol{b}$ on the other hand is a vector containing the b coefficients (i.e. b_{C} for Caputo's definition, b_{C} + b_{RL} for the Riemann-Liouville fractional derivative). Therefore, in a computed time step the state vector values can be obtained by solving the following system of equations:

$$(\boldsymbol{A}-\boldsymbol{a}_{\mathrm{diag}})\boldsymbol{x}_j=\boldsymbol{b}-\boldsymbol{B}\boldsymbol{v}(t_j). \tag{25}$$

3. Example

The simple circuit problem given in figure 3 is considered.



$E(t)=\mathbf{1}(t)$, $R=2\,\Omega$,
$C=5.97\,\mu\mathrm{F}\cdot\mathrm{s}^{\alpha-1}$, $L=0.6\,\mathrm{H}\cdot\mathrm{s}^{\alpha-1}$, $\alpha=0.8$

Fig. 3. Fractional order circuit example – RLC circuit with fractional order capacitor and coil

The circuit contains both a fractional order coil and a fractional order capacitor. Both are of the order α.

The circuit equations can be given in a matrix form:

$$\begin{cases}\boldsymbol{M}_{\mathrm{I}}\boldsymbol{y}(t)+\boldsymbol{M}_{\mathrm{II}}\boldsymbol{x}(t)=\boldsymbol{T}\boldsymbol{v}(t),\\ \mathbf{d}^{\boldsymbol{\alpha}}_{\Xi}\boldsymbol{x}(t)=\boldsymbol{M}_{\mathrm{III}}\boldsymbol{x}(t)+\boldsymbol{M}_{\mathrm{IV}}\boldsymbol{y}(t),\end{cases} \tag{26}$$

where the state vector is:

$$\boldsymbol{x}(t)=[i_L(t)\quad u_C(t)]^{\mathrm{T}}. \tag{27}$$

$\boldsymbol{y}$(t) is a vector of the remaining useful variables:

$$\boldsymbol{y}(t)=[V_1(t)\quad V_2(t)\quad u_L(t)\quad i_C(t)]^{\mathrm{T}}, \tag{28}$$

and $\boldsymbol{v}(t)$ is generally a vector of source time functions – in this case, however, represented by a single function:

$$\boldsymbol{v}(t)=[E(t)], \tag{29}$$

hence:

$$\boldsymbol{M}_{\mathrm{I}}=\begin{bmatrix}\frac{1}{R} & 0 & 0 & 0\\ 0 & 0 & 0 & -1\\ 1 & -1 & -1 & 0\\ 0 & 1 & 0 & 0\end{bmatrix}, \tag{30}$$

while the state vector multipliers:

$$\boldsymbol{M}_{\mathrm{II}}=\begin{bmatrix}1 & 0\\ 1 & 0\\ 0 & 0\\ 0 & -1\end{bmatrix}, \tag{31}$$

and the source multipliers:

$$\boldsymbol{T}=\begin{bmatrix}\frac{1}{R} & 0 & 0 & 0\end{bmatrix}^{\mathrm{T}}. \tag{32}$$

As for the second equation of (26):

$$\boldsymbol{M}_{\mathrm{III}}=\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}, \tag{33}$$

while:

$$\boldsymbol{M}_{\mathrm{IV}}=\begin{bmatrix}0 & 0 & \frac{1}{L} & 0\\ 0 & 0 & 0 & \frac{1}{C}\end{bmatrix}. \tag{34}$$

The system of equations (25) can be brought to the form of state equations. With respect to the notations in (22) one obtains:

$$\boldsymbol{A}=\boldsymbol{M}_{\mathrm{III}}-\boldsymbol{M}_{\mathrm{IV}}\boldsymbol{M}_{\mathrm{I}}^{-1}\boldsymbol{M}_{\mathrm{II}}, \tag{35}$$

and:

$$\boldsymbol{B}=\boldsymbol{M}_{\mathrm{IV}}\boldsymbol{M}_{\mathrm{I}}^{-1}\boldsymbol{T}. \tag{36}$$

4. ActiveX SubIval DLL

The ActiveX DLL has been built in such a way as to support standard integer, double and string operations between the higher level program and the DLL. This allows it to be used in a larger variety of computational programs even in their older versions.

The DLL has been written in C# and even though nowadays many programs allow the usage of external .NET libraries older software did not support such functionality. For example, MATLAB has had this functionality introduced in version 7.8 (2009a) [15].

The code applying SubIval should have an order as suggested by the following guidelines (assuming an application in MATLAB):

– first a COM object (in here named `hSubIval`) needs to be obtained by creating a COM object (note that the SubIval DLL needs to be registered first):

```
hSubIval =
  actxserver('SubIvalNS.SubIvalLauncher');
```

– now from this moment on all functions are run as methods of the `hSubIval` object,

– one can define how many independent subinterval dynamics processes will be run independently by executing the first function, e.g. here one initializes one process:

```
hSubIval.Initialize(1);
```

– subsequently – each process needs to be initialized by referring to it through its zero based index `p_id`; the number of variables under fractional derivatives `n` needs to be given, along with the polynomial order `p`, the initial time instance `t0` and the type of subinterval dynamics process (0 stands for constant step, which is the only one discussed in this paper):

```
hSubIval.
  InitializeComputer(p_id, n, p, t0, 0);
```

– each of the fractional derivative orders needs to be given by entering its zero based index `j` and value `alfaj`:

```
hSubIval.InitAlpha(p_id, alfaj, j);
```

– then the initial values need to be entered, also through a zero based index entry:

```
hSubIval.Putxi(p_id, x0j, j);
```

– at this point the initialization is complete and the time stepping process can begin; in every new time instance `tnext` one needs to run:

```
hSubIval.newt(p_id, tnext);
```

– to perform the core SubIval computations one needs to execute:

```
hSubIval.ComputeSimple(p_id);
```

– in the case of a predictor-corrector scheme – the functions are different, this, however, will not be discussed in this paper,

– after the computations had been performed the a and b coefficients are available; to obtain them one can run:

```
hSubIval.getab(p_id, j);
```

for the variable with the zero based index `j`,

– after a solution has been obtained then each state variable value can be entered also with `Putxi`.

Exemplary codes in both MATLAB and Mathematica, which allow to perform computations for the discussed problem, are presented in table 1. Both code fragments start from the stage where the matrices `MI`, `MII`, `MIII`, `MIV` (representing M_{I}, M_{II}, M_{III} and M_{IV}) and `Tsrc` (representing T) have already been filled with the proper values according to equations (30) to (34).

Table 1. Code fragments for solving fractional order state equations

MATLAB

```
Bstate=MIV*(MI\T);
Astate=MIII-MIV*(MI\MII);
n=2; (*number of x variables*)
iL0 = 0; uC0 = 0;
nt = 400; t0 = 0; tmax = 1e-2;
dt = (tmax-t0)/(nt-1);
t=0:dt:tmax;

x=zeros(n, nt);

(*the ActiveX DLL needs to be
registered first!*)

p = 3; (*method order*)
solverType = 0;

hSubIval = actxserver

('SubIvalNS.SubIvalLauncher');
hSubIval.Initialize(1);
hSubIval.InitializeComputer
 (0, n, p, t0, solverType);
hSubIval.
 InitAlpha(0, alpha, 0);
hSubIval.
 InitAlpha(0, alpha, 1);
hSubIval.Putxi(0, iL0, 0);
hSubIval.Putxi(0, uC0, 1);
i1 = 2;
x(1, 1) = iL0;
x(2, 1) = uC0;
while i1<=nt
  hSubIval.newt(0, t(i1));
  hSubIval.ComputeSimple(0);
  aa = zeros(n,1);
  bb = zeros(n,1);
  for in=1:n
   Composite =
    hSubIval.getab(0, in-1)
   aa(in) = Composite(2);
   bb(in) = Composite(3);
  end

  A = Astate - diag(aa);
  b = bb -
Bstate*vFunc(t(i1));
  x(:,i1)=A\b;

  for i_n = 1:n
   hSubIval.
    Putxi(0, x0(in, i1), in-
1);
  end
  i1=i1+1;
end
```

Mathematica

```
IMI = Inverse[MI];
BState = MIV.IMI.T;
AState = MIII-MIV.IMI.MII;
n=2; (*number of x variables*)
iL0 = 0; uC0 = 0;
nt = 400; t0 = 0; tmax = 10^-2;
dt = (tmax-t0)/(nt-1);
t = Table[(t0 + dt*i)*1.0,
    {i, 0, nt - 1}];
x = ConstantArray[0, {n, nt}];
Needs["NETLink`"]
InstallNET[]
p = 3; (*method order*)
solverType = 0;
ThePath = NotebookDirectory[]
          <> "SubIval.dll";
SubIval =
      LoadNETAssembly[ThePath]
SIL = NETNew
["SubIval_NS.SubIvalLauncher"];
SIL@Initialize[1]
SIL@
 InitializeComputer
  [0, n, p, t0, solverType]
SIL@InitAlpha[0, alpha, 0]
SIL@InitAlpha[0, alpha, 1]
SIL@Putxi[0, iL0, 0]
SIL@Putxi[0, uC0, 1]
i1 = 2;
x[[1, 1]] = iL0;
x[[2, 1]] = uC0;
While[i1 <= nt,
  SIL@newt[0, t[[i1]]];
  SIL@ComputeSimple[0];
  aa = ConstantArray[0, n];
  bb = ConstantArray[0, n];
  For[in = 1, in <= n, in++,
   Composite =
    SIL@getab[0, in - 1];
    aa[[in]] = Composite[[2]];
    bb[[in]] = Composite[[3]];
  ];
  A =
   AState - DiagonalMatrix[aa];
  bv =
   bb - BState*vFunc[t[[i1]]];
  xtemp = LinearSolve[A, bv];
  For[in = 1, in <= n, in++,
   x[[in, i1]] =
    xtemp[[in]][[1]];
    SIL@Putxi
    [0, 1.0*x[[in, i1]],
     in - 1];
  ];
  i1++;
];
```

The obtained time functions for the state variables are presented in figure 4. The results obtained by means of SubIval have been compared with an analytical solution (basing on the Mittag-Leffler function) found in literature [22].

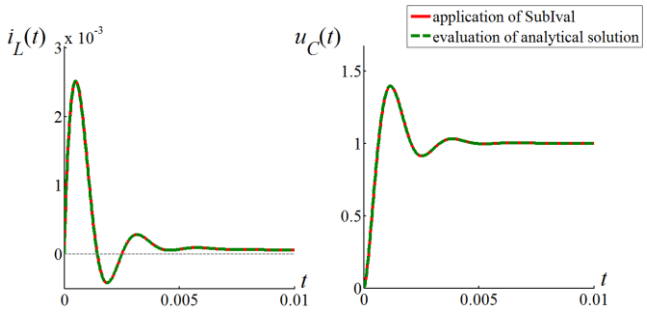


Fig. 4. Comparison of the computation results

The obtained results present a close resemblance to the referential, analytical solution.

5. Summary

A numerical method for the computation of the fractional derivative in initial value problems has been discussed. The method is now known as SubIval. Its brief explanation has been given in section 2. In section 3 a simple circuit example has been introduced, which requires dealing with the fractional derivative. The state equations have been formulated through a matrix method. A short introduction has been given to the functions that need to be executed from the SubIval DLL through MATLAB. Exemplary scripts have been presented in both MATLAB and Mathematica. These scripts allow to obtain the time functions for the state variables of the discussed problem. In this paper only a constant step variant of SubIval has been used. An adaptive time step predictor-corrector scheme cooperating with the SubIval DLL will be discussed in a future paper.

Literatura

[1] Abdeljawad T.: On Riemann and Caputo fractional differences. Computers and Mathematics with Applications 62/2011, 1602–1611.

[2] Arikoglu A., Ozkol I.: Solution of fractional integro-differential equations by using fractional differential transform method. Chaos, Solitons & Fractals 40(2)/2007, 521–529.

[3] Brociek R., Słota D., Wituła R.: Reconstruction of the Thermal Conductivity Coefficient in the Time Fractional Diffusion Equation. Advances in Modelling and Control of Non-integer-Order Systems 2016, 239–247.

[4] Caputo M.: Linear models of dissipation whose Q is almost frequency independent – II. Geophysical Journal International 13(5)/1967, 529–539.

[5] Cui M.: Compact finite difference method for the fractional diffusion equation. Journal of Computational Physics 228/2009, 7792–7804.

[6] Ducharne B., Sebald G., Guyomar D., Litak G.: Dynamics of magnetic field penetration into soft ferromagnets. Journal of Applied Physics 117/2015, 243907.

[7] Huang L., Xian-Fang L., Zhao Y., Duan X.Y.: Approximate solution of fractional integro-differential equations by Taylor expansion method. Computers and Mathematics with Applications 62/2011, 1127–1134.

[8] Jakubowska A., Walczak J.: Analysis of the Transient State in a Series Circuit of the Class $RL_\beta C_\alpha$. Circuits, Systems and Signal Processing 35/2016, 1831–1853.

[9] Katugampola U.N.: A new approach to generalized fractional derivatives. Bull. Math. Anal. Appl. 6(4)/2014, 1–15.

[10] Kawala-Janik A., Podpora M., Gardecki A., Czuczwara W., Baranowski J., Bauer W.: Game controller based on biomedical signals. Methods and Models in Automation and Robotics (MMAR) 2015, 20th International Conference, 934–939.

[11] Klamka, J., Czornik, A., Niezabitowski, M., Babiarz, A.: Controllability and minimum energy control of linear fractional discrete-time infinite-dimensional systems. Control & Automation (ICCA) 2014, 11th IEEE Conference, 1210–1214.

[12] Lubich C.: Fractional linear multistep methods for Abel-Volterra integral equations of the second kind. Math. Comput. 45/1985, 463–469.

[13] Momani S., Noor M.A: Numerical methods for fourth order fractional integro-differential equations. Appl. Math. Comput. 182/2006, 754–760.

[14] Munkhammar J.D.: Riemann-Liouville fractional derivatives and the Taylor-Riemann series. UUDM Project Report 7/2004, 1–18.

[15] New MATLAB External Interfacing Features in 2009a. MathWorks.http://www.mathworks.com/videos/new-external-interfacing-features-in-r2009a-101547.html (available 15.06.2016).

[16] Ostalczyk P. W., Duch P., Brzeziński D. W., Sankowski D.: Order Functions Selection in the Variable-, Fractional-Order PID Controller. Advances in Modelling and Control of Non-integer-Order Systems 2014, 159–170.

[17] Rawashdeh E.A.: Numerical solution of fractional integro-differential equations by collocation method. Applied Mathematics and Computation 176/2006, 1–6.

[18] Saeedi H., Mohseni Moghadam M.: Numerical solution of nonlinear Volterra integro-differential equations of arbitrary order by CAS wavelets. Commun. Nonlinear Sci. Numer. Simulat. 16/2011, 1216–1226.

[19] Schäfer I., Krüger K.: Modelling of lossy coils using fractional derivatives. Phys. D: Appl. Phys. 41/2008, 1–8.

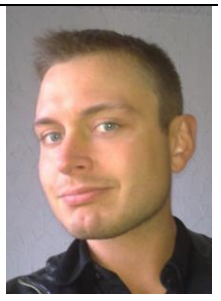
[20] Skruch P., Mitkowski W.: Fractional-order models of the ultracapacitors. Advances in the Theory and Applications of Non-Integer Order Systems. Springer International Publishing 2013, 281–293.

[21] Sowa M.: A subinterval-based method for circuits with fractional order elements. Bull. Pol. Ac.: Tech. 62(3)/2014, 449–454.

[22] Włodarczyk M., Zawadzki A.: RLC circuits in aspect of positive fractional derivatives. Scientific Works of the Silesian University of Technology: Electrical Engineering 1/2011, 75–88.

Ph.D. Eng. Marcin Sowa
e-mail: marcin.sowa@polsl.pl

Assistant professor at the Faculty of Electrical Engineering at the Silesian University of Technology in Gliwice (Poland). Is the sole author or co-author in around 30 papers published in international conference proceedings or journals. Specializes in numerical methods in electrical engineering, fractional derivative computations, electromagnetic field computations and C# programming. He is the author of SubIval – the subinterval-based method for numerical computations of the fractional derivative in initial value problems.



otrzymano/received: 15.06.2016 *przyjęto do druku/accepted: 14.08.2017*