International Journal of Pure and Applied Mathematics
Volume 119 No. 3 2018, 391-401
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
doi: 10.12732/ijpam.v119i3.1

USING GPU TO SPEED UP CALCULATION OF SOME APPROXIMATE METHODS FOR FRACTIONAL DIFFERENTIAL EQUATIONS

Angel Golev[1 §], Alexander Penev[2], Kremena Stefanova[3], Snezhana Hristova[4]
[1,2,3,4]Faculty of Mathematics and Informatics
"Paisii Hilendarski" University of Plovdiv
Plovdiv, BULGARIA

Abstract: The main aim of the paper is comparing the speed of calculation for a particular problem for fractional differential equations by parallel CUDA cores. The approximate solving of the initial value problem for scalar nonlinear Riemann-Liouville fractional differential equation is based on a suggested iterative scheme for constructing sequences of upper and lower solutions of the considered problem. In the general case the elements of these sequences are obtained by an integral containing the Mittag-Leffler function. It can cause the successive approximations to be difficult for practical applications. The computer program using the parallel calculations is built and applied to solve the problem.

Key Words: derivatives from fractional order, differential equations, iterative schemes, bottom and top solutions, software realization, parallel calculations

1. Introduction

Many processes with anomalous dynamics in science and engineering can be formulated mathematically by fractional differential operators [6], [9]. There are only a small number of fractional differential equations such as the scalar

Received: February 21, 2018
Revised: June 20, 2018
Published: July 11, 2018

§Correspondence author

linear equation with constant coefficients which can be solved in closed form so this causes some problems in practical applications.

In this paper is considered an initial value problem for a nonlinear scalar Riemann-Liouville fractional differential equation on a closed interval. In the case of the Riemann-Liouville derivative the observed functions generally have a singularity at the left-most endpoint, and therefore they are only C^0 on a half open interval, with a special property we will define below. An iterative technique combined with the method of lower and upper solutions is applied to find the approximate solution of the given problem. When the right hand side of the equations are monotone functions with respect to the time variable the elements of these sequences do not depend on the Mittag-Leffler functions and they can be obtained in closed form with the help of an appropriate software such as Wolfram Mathematica. In the general case the elements of these sequences are obtained with an integral containing the Mittag-Leffler function with two parameters so as a result finding successive approximations can be difficult in practice [1, 2]. In this case a special computer program is built and applied to solve particular problems.

2. Preliminary Notes

Consider the initial value problem (IVP) for the nonlinear *Riemann-Liouville-type fractional differential equation* (FrDE)

$$\begin{aligned} &{}^{RL}_{t_0}D^q_t\, x(t) = F(t, x(t)) \quad \text{for} \quad t \in [t_0, t_0 + T] \\ &(t - t_0)^{1-q} x(t)|_{t=t_0} = x_0, \end{aligned} \tag{1}$$

where $q \in (0, 1)$, $x, x_0 \in \mathbb{R}$.

Let $\tau_0 \in \mathbb{R}_+$, $\tau_0 \neq t_0$ be an initial time. Consider the following IVP for FrDE similar to (1) but with a different initial time (ITD):

$${}^{RL}_{\tau_0}D^q_t x(t) = F(t, x(t)) \quad \text{for } t \in [\tau_0, \tau_0 + T], \qquad (t - \tau_0)^{1-q} x(t)|_{t=\tau_0} = x_0. \tag{2}$$

The change of the initial time reflects not only the initial condition but also the fractional derivative of the solution. Let $\theta_0 \in \mathbb{R}_+$, $\theta_0 \neq t_0$, $\theta_0 \neq \tau_0$, be a different initial time. Consider the following IVP for FrDE

$${}^{RL}_{\theta_0}D^q_t x(t) = F(t, x(t)) \quad \text{for } t \in [\theta_0, \theta_0 + T], \qquad (t - \theta_0)^{1-q} x(t)|_{t=\theta_0} = x_0. \tag{3}$$

For any function $\xi \in C([t_0, t_0 + T], \mathbb{R})$ and a constant M define the operator

$$\Omega_M(\xi)(t) = \Gamma(q)(t - t_0)^{q-1} x_0 E_{q,q}(-M(t - t_0)^q)$$

$$+\int_{t_0}^{t}(t-s)^{q-1}E_{q,q}(-M(t-s)^q)\Big(F(s,\xi(s))+M\xi(s)\Big)ds.$$

In what follows we will use the following theoretical result:

Theorem 1. *[4, Theorem 1] Let the following conditions be fulfilled:*

1. *Let the points θ_0, t_0, τ_0 : $0 \le \theta_0 \le t_0 \le \tau_0$ be given and the function $v \in C_{1-q}([\theta_0, \theta_0+T])$ be a lower solution of the IVP for FrDE (3) on the interval $[\theta_0, \theta_0+T]$ and the function $w \in C_{1-q}([\tau_0, \tau_0+T])$ be an upper solution of the IVP for FrDE (2) on the interval $[\tau_0, \tau_0+T]$ such that $v(t-\eta) \le w(t+\xi)$ for $t \in [t_0, t_0+T]$ where $\eta = t_0-\theta_0 \ge 0$, $\xi = \tau_0-t_0 > 0$.*

2. *The function $F \in C([\theta_0, \theta_0+T] \cup [t_0, t_0+T] \cup [\tau_0, \tau_0+T] \times \mathbb{R}, \mathbb{R})$ is non-decreasing w.r.t. its first argument $t \in [t_0, t_0+T]$ and there exists a constant $M > 0$ such that for any $t \in [t_0, t_0+T]$, x, $y \in \mathbb{R}$: $v(t) \le x \le y \le w(t)$ the inequality $F(t,x) - F(t,y) \le -M(x-y)$ holds.*

Then there exist two sequences of functions $\{v^{(n)}(t)\}_0^\infty$ and $\{w^{(n)}(t)\}_0^\infty$, $t \in [t_0, t_0+T]$, such that:

a. *The sequences are defined by $v^{(0)}(t) = v(t-\eta)$, $w^{(0)}(t) = w(t+\xi)$ and for $n \ge 1$*

$$\begin{aligned} v^{(n)}(t) &= \Gamma(q)(t-t_0)^{q-1}x_0E_{q,q}(-M(t-t_0)^q) \\ &\quad + \int_{t_0}^{t}(t-s)^{q-1}E_{q,q}(-M(t-s)^q)\times \\ &\quad \times \Big(F(s, v^{(n-1)}(s)) + Mv^{(n-1)}(s))\Big)ds, \quad t \in (t_0, t_0+T], \\ (t-t_0)^{1-q}v^{(n)}(t)|_{t=t_0} &= x_0 \end{aligned} \tag{4}$$

and

$$\begin{aligned} w^{(n)}(t) &= \Gamma(q)(t-t_0)^{q-1}x_0E_{q,q}(-M(t-t_0)^q) \\ &\quad + \int_{t_0}^{t}(t-s)^{q-1}E_{q,q}(-M(t-s)^q)\times \\ &\quad \times \Big(F(s, w^{(n-1)}(s)) + Mw^{(n-1)}(s))\Big)ds, \quad t \in (t_0, t_0+T] \\ (t-t_0)^{1-q}w^{(n)}(t)|_{t=t_0} &= x_0. \end{aligned} \tag{5}$$

b. *The functions $v^{(n)}(t)$, $n = 0, 1, 2, \ldots$ satisfy on $[t_0, t_0+T]$ the inequalities $v^{(n)}(t) \le \Omega_M(v^{(n)})(t)$.*

c. *The functions* $w^{(n)}(t)$, $n = 0, 1, 2, \ldots$ *satisfy on* $[t_0, t_0+T]$ *the inequalities* $w^{(n)}(t) \leq \Omega_M(w^{(n)})(t)$.

d. *The sequence* $\{v^{(n)}(t)\}$ *is increasing on* $[t_0, t_0 + T]$, *i.e.* $v^{(k-1)}(t) \leq v^{(k)}(t)$ *for* $t \in [t_0, t_0 + T]$, $k = 1, 2, \ldots$.

e. *The sequence* $\{w^{(n)}(t)\}$ *is decreasing on* $[t_0, t_0+T]$, *i.e.* $w^{(k-1)}(t) \geq w^{(k)}(t)$ *for* $t \in [t_0, t_0 + T]$, $k = 1, 2, \ldots$.

f. *The inequality*

$$v^{(k)}(t) \leq w^{(k)}(t) \quad \text{for } t \in [t_0, t_0 + T],\ k = 1, 2, \ldots \tag{6}$$

holds.

g. *The sequences* $\{(t - t_0)^{1-q} v^{(n)}(t)\}_0^\infty$ *and* $\{(t - t_0)^{1-q} w^{(n)}(t)\}_0^\infty$ *converge uniformly on* $[t_0, t_0 + T]$ *and* $(t - t_0)^{1-q} V(t) = \lim_{k\to\infty}(t - t_0)^{1-q} v^{(n)}(t)$, $(t - t_0)^{1-q} W(t) = \lim_{k\to\infty}(t - t_0)^{1-q} w^{(n)}(t)$ *on* $[t_0, t_0 + T]$.

h. *The limit functions* $V(t)$ *and* $W(t)$ *are fixed points of the operator* Ω, *i.e.* $V(t) = \Omega(V)(t)$, $W(t) = \Omega(W)(t)$.

e. *The limit functions* $V(t)$ *and* $W(t)$ *are solutions of the IVP for FrDE (1) on* $[t_0, t_0 + T]$.

We will represent a computer realization with parallel calculations of this particular problem:

Example 1. Let $t_0 = 1,\ T = 2$ and consider the IVP for scalar Riemann-Liouville FrDE

$${}_1^{RL}D_t^{0.9} x(t) = x(t)(1 - x(t)) \ \text{ for } \ t \in (1, 3], \qquad (t - 1)^{0.1} x(t)|_{t=1} = 1. \tag{7}$$

The function $v(t) = 0.7t^{-0.1}$ is a mild lower solution on $[1, 3]$ of the IVP for FrDE (7) since the inequalities $(t - 1)^{0.1} v(t)|_{t=1} = 0 < 1$ and

$$0.7t^{-0.1} \leq (t - 1)^{-0.1} \tag{8}$$

$$+ \frac{1}{\Gamma(0.9)} \int_0^t (t - s)^{-0.1} (0.7s^{-0.1})(1 - 0.7s^{-0.1}) ds, \quad t \in [0, 1]. \tag{9}$$

are satisfied.

The function $w(t) = 2(t-2)^{-0.1}$ is not a mild upper solution on $[1, 3]$ of the IVP for FrDE (7) since the inequality $(t - 1)^{0.1} w(t)|_{t=2} = 0 > 1$ does not hold.

The function $w(t) = 2(t-2)^{-0.1}$ is a mild upper solution on $[2,4]$ of the IVP for FrDE (7) since the inequalities $(t-2)^{0.1}x(t)|_{t=2} = 2 > 1$ and

$$2(t-2)^{-0.1} \geq (t-2)^{-0.1} + \frac{2}{\Gamma(0.9)} \int_2^t (t-s)^{-0.1}(s-2)^{-0.1}(1-2(s-2)^{-0.1})ds, \ t \in [2,4] \tag{10}$$

are satisfied.

Also, the inequality $v(t) \leq w(t+1)$ holds on $[1,3]$.

Define the zero approximations by $v^{(0)}(t) = v(t) = 0.7t^{-0.1}$ and $w^{(0)}(t) = w(t+1) = 2(t-1)^{-0.1}$ for $t \in [1,3]$. We apply the iterative scheme suggested in Theorem 1 with $M = 10$ starting from the point $t = 1.0001$.

From equations (4) and (5) we get

$$\begin{aligned} v^{(n)}(t) &= \Gamma(0.9)(t-1)^{-0.1}E_{0.9,0.9}(-2.5(t-1)^{0.9}) \\ &\quad + \int_1^t (t-s)^{-0.1}E_{0.9,0.9}(-2.5(t-s)^{0.9})\times \\ &\quad \times \Big(v^{(n-1)}(s)(1-v^{(n-1)}(s)) + 2.5v^{(n-1)}(s))\Big)ds, \qquad (11) \\ &\quad \text{for } t \in (1,3], \ n = 1, 2, \ldots, \\ (t-1)^{0.1}v^{(n)}(t)|_{t=1} &= 1 \end{aligned}$$

and

$$\begin{aligned} w^{(n)}(t) &= \Gamma(0.9)(t-1)^{-0.1}E_{0.9,0.9}(-2.5(t-1)^{0.9}) \\ &\quad + \int_1^t (t-s)^{-0.1}E_{0.9,0.9}(-2.5(t-s)^{0.9})\times \\ &\quad \times \Big(w^{(n-1)}(s)(1-w^{(n-1)}(s)) + 2.5w^{(n-1)}(s))\Big)ds, \qquad (12) \\ &\quad \text{for } t \in (1,3], \ n = 1, 2, \ldots, \qquad (13) \\ (t-1)^{0.1}w^{(n)}(t)|_{t=1} &= 1. \end{aligned}$$

The presence of the Mittag-Leffler function in the integrals in (11) and (12) do not allow us to solve these integrals in closed form. We need a numerical method and to build a special computer program. Also, we consider the convergence of sequences $\{(t-1)^{0.1}v^{(n)}(t)\}$ and $\{(t-1)^{0.1}w^{(n)}(t)\}$ instead of $\{v^{(n)}(t)\}$ and $\{w^{(n)}(t)\}$, respectively.

The computer algorithm is given in Section 3 and a part of the numerical results obtained by running the program are given in Table 1.

t	1.0001	1.3000	1.6000	1.9000	2.1000	2.4000	2.7000	3.0000
$w^{(0)}(t)$	5.0237	2.2558	2.1048	2.0211	1.9810	1.9338	1.8966	1.8660
$w^{(1)}(t)$	2.5116	1.9239	1.8586	1.8074	1.7810	1.7491	1.7233	1.7025
$w^{(2)}(t)$	2.5098	1.6629	1.6752	1.6494	1.6329	1.6115	1.5935	1.5790
$w^{(3)}(t)$	2.5098	1.4551	1.5294	1.5264	1.5180	1.5048	1.4926	1.4826
$w^{(4)}(t)$	2.5098	1.2978	1.4087	1.4269	1.4257	1.4193	1.4117	1.4052
$v^{(4)}(t)$	2.5098	0.8776	0.7446	0.7251	0.7213	0.7170	0.7129	0.7099
$v^{(3)}(t)$	2.5098	0.8032	0.7186	0.7080	0.7045	0.6995	0.6946	0.6908
$v^{(2)}(t)$	2.5098	0.7382	0.6987	0.6913	0.6871	0.6810	0.6752	0.6706
$v^{(1)}(t)$	2.5063	0.6977	0.6826	0.6742	0.6690	0.6616	0.6549	0.6494
$v^{(0)}(t)$	0.6999	0.6818	0.6678	0.6564	0.6499	0.6413	0.6338	0.6271

Table 1: Example 1. Values of successive approximations $w^{(n)}(t)$ and $v^{(n)}(t)$, $n = 0$, 1, 2, 3, 4.

3. Computer Realization with Parallel Calculations using CUDA Cores

The sequences of successive approximations $v^{(k)}(t)$ and $w^{(k)}(t), k = 1, 2, \ldots$ often cannot be obtained implicitly as known mathematical functions because of the presence of Mittag-Leffler functions $E_{q,q}(x)$ and Γ functions in their formulas. In connection with this an algorithm for numerical solving of the integrals is used and implemented at C#. The algorithm is as follows.

Initially, the values of the Γ function are computed at all points necessarily to calculate the values of $E_{q,q}(x)$. In the case when the Γ function is included in the integral, then its values are also calculated and saved at all points in which the values of the successive approximations $v^{(k)}(t)$ and $w^{(k)}(t)$ are obtained. In our examples the number of these points is about 10000 for an interval with length 1. For calculating values of Γ function we use the method of the .NET Chart class, which works with sufficient accuracy. Using these values and the main definition with infinite sum we calculate also the values of the function $E_{q,q}(x)$ at all of these points. For the examples in this article, the calculation is fast enough and with sufficient accuracy. Note that it could be used for another implementation of Mittag-Leffler's approximate calculation.

To obtain the successive approximations, we first calculate the values of the function $\Gamma(qk+k)$ for every $k = 0, 1, \ldots maxnGm$. We use Lanczos approximation, but the function must not be recursive because of the CUDA code.

```
static int maxnGm = 470;
const double q = 0.9;
const int g = 7;
```

```
static double[] p =
  {0.99999999999980993, 676.5203681218851, -1259.1392167224028,
      771.32342877765313, -176.61502916214059,
      12.507343278686905, -0.13857109526572012,
      9.9843695780195716e-6, 1.5056327351493116e-7};
public static double Gamma(double z, double[] pp)
{
   double zz=z;
   if (z<0.5) z=1-z;
   z-=1;
   double x=pp[0];
   for (var i=1; i<g+2; i++) x+=pp[i]/(z+i);
   double t=z+g+0.5;
   double res = Math.Sqrt(2*Math.PI)*(Math.Pow(t,z+0.5))*
       Math.Exp(-t)*x;
   if (zz<0.5) res=Math.PI/(Math.Sin(Math.PI*zz)*res);
   return res;
}

[EntryPoint("calcgamma")]
public static void CalcGamma(int N, double[] g, double[] p)
{
   Parallel.For(0,N,i => {g[i]=Gamma(q*i+q,p);});
}
```

We use these values to obtain in advance the values of the Mittag-Leffler function $E_{q,q}(x)$ for all necessary points $x = -2.5 * t^q$. The calculation of Γ and $E_{q,q}$ for all these points is independently and it can be run on the parallel CUDA cores.

For a calculation of $E_{q,q}$ we use the definition with an infinite sum. Because of the way how CUDA works we can not use global arrays and have to set them as a parameters.

```
const double eps_e = 0.000001;
public static double Func_E_q(double z, double[] Gm)
{
   long k=0;
   double sum=0, zk=1, cr_el=1.0/Gm[k];
   for (var i=0; Math.Abs(cr_el)>eps_e; i++) {
      sum+=cr_el; zk*=z; k++; cr_el=zk/Gm[k];
   }
   sum+=cr_el;
   return sum;
}

[EntryPoint("calcmlf")]
public static void CalcMLF(int N, double[] Eqq, double[] Gm2)
{
```

```
    Parallel.For(0,N,t =>
        { Eqq[t] = Func_E_q(-M*Math.Pow(t*eps,q),Gm2); });
}
```

We will note that the execution of *Parallel.For* can be done in smaller ranges of the index t, but in our case this does not lead to faster speed.

First we calculate the initial iterations $v^{(0)}(t)$ and $w^{(0)}(t)$ for $t = (1, 3]$. We will emphasize on the fact that each next iteration depends on the values of the previous one.

```
const double eps = 0.00001;
const double M = 2.5;
const double t0 = 1;
const double T = 2;
const double tt = t0+T+eps;
const double q1 = 1-q;

[EntryPoint("approxall")]
public static void ApproxAll(int N, int n, double[][] v, double
    [][] w, double[] Eqq, double Gm_q)
{
    Parallel.For(1,N,t =>
        {v[0][t] = 0.7*Math.Pow(t0+t*eps,q1);});
    Parallel.For(1,N,t =>
        {w[0][t] = 2.0*Math.Pow(t0+t*eps-1,q1);});
    for (int i=1; i<=n; i++)
    {
        Parallel.For(1,N,t =>
            {v[i][t] = CalcInt(v,Eqq,i,t,t0+t*eps,Gm_q);});
        Parallel.For(1,N,t =>
            {w[i][t] = CalcInt(w,Eqq,i,t,t0+t*eps,Gm_q);});
    }
}
```

To store the results we use two jagged arrays $v[][]$ and $w[][]$. The first dimension is for the number of iterations, including the initial one. The length of the second dimension is T/eps. Note two dimensional arrays as parameters to CUDA core does not work in this case.

We calculate the next approximation in all point $t \in (t_0, t_0 + T)$. Since the integral function does not depend on its values in previous points, we can execute parallelly the current approximation of the functions v and w in each point. Partition of calculations to smaller group with a length which is comparable to the number of CUDA cores in this case does not lead to faster calculation.

In point t_0 functions v and w are undefined and because of that we compute the functions from the point $t_0 + eps$ forward, as the index of the arrays begins from 1.

It follows a sample code for calculating the integrals v and w using the Trapezoidal rule with *eps* accuracy. We emphasize on the fact that a more accurate numerical method may be used. The Trapezoidal rule uses simpler calculations and each calculation leads to a corresponding error. In practice it may turn out that the error is less than that of a theoretically better method. The considered function in this example is:

```
private static double CalcInt(double[][] vw, double[] Eqq,
    int n, long it, double tval, double Gm_q)
{
   double f, pf, sv, sum=0, s=t0+eps;
   long i=1, ie=it-1;
   sv = Math.Pow(s-1,q1)*vw[n-1][i];
   pf = Math.Pow(tval-s,q1)*Eqq[ie]*(sv*(1-sv)+M*sv);
   while (s<tval)
   {
      i++; ie--; s+=eps;
      sv = Math.Pow(s-1,q1)*vw[n-1][i];
      f = Math.Pow(tval-s,q1)*Eqq[ie]*(sv*(1-sv)+M*sv);
      sum += (pf+f)*eps;
      pf = f;
   }
   return sum/2*Math.Pow(tval-1,0.1)+Gm_q*Eqq[it];
}
```

We will give an example of a main function.

```
using Hybridizer.Runtime.CUDAImports;
...

      static void Main(string[] args)
{
   cudaDeviceProp prop;
   cuda.GetDeviceProperties(out prop, 0);
   // if .SetDistrib is not used, the default is .SetDistrib(
       prop.multiProcessorCount * 16, 128)
   HybRunner runner = HybRunner.Cuda().SetDistrib(256, 64);
   // create a wrapper object to call GPU methods instead of C#
   dynamic wrapped = runner.Wrap(new Program());

   double[] Gm2 = new double[maxnGm];
   wrapped.CalcGamma(maxnGm, Gm2, p);

   int n=4, MAX_PT=(int)(T/eps);

   double[] Eqq = new double[MAX_PT+1];
   wrapped.CalcMLF(MAX_PT, Eqq, Gm2);

   double[][] v = new double[n+1][];
```

```
        double[][] w = new double[n+1][];
        for (int i=0; i<=n; i++)
        {
            v[i] = new double[MAX_PT+1];
            w[i] = new double[MAX_PT+1];
        }
        double Gm_q = Gamma(q, p);
        wrapped.ApproxAll(MAX_PT, n, v, w, Eqq, Gm_q);
        ...
        }
    }
```

We use Microsoft Visual Studio C# with an installed Altimesh's Hybridizer C# [5] – multi-target framework for data parallel computing (with automatic retargeting/translation C# to GPU code). The CUDA-enabled hardware is NVIDIA Tesla K20m (GPU GK110 with 2496 CUDA cores, clock 706MHz, 5GB GDDR5 RAM). The host computer is with CPU AMD A10-6790K (4 cores, clock 4GHz), 16GB RAM, and Windows 7 Pro (with installed latest NVIDIA drivers and CUDA 9.2 Toolkit).

The speed of calculation depends on the parameters of function $.SetDistrib()$ which also depends on the particular GPU with CUDA cores.

In this case we obtained better results with parameters $(256, 64)$ and the program works about 9.8 times faster than the computer realization with consecutive calculations. The computation time is quadratic to the number of points in which we calculate the integrals, which means increasing of the accuracy of integration will reflect to quadratic increasing the time of calculations.

4. Conclusions

In this paper a computer programs using the parallel calculations is built and applied to approximately solve an initial value problem for scalar nonlinear Riemann-Liouville fractional differential equation. Using the parallel CUDA cores, the computations are about 10 times faster than the consecutive computations.

5. Acknowledgements

This work is partially supported by the FP17-FMI-008 project of the Scientific Fund of the University of Plovdiv "Paisii Hilendarski", Bulgaria.

References

[1] R. Agarwal, A. Golev, S. Hristova, D. O'Regan, and K. Stefanova, Iterative techniques with initial time difference and computer realization for the initial value problem for Caputo fractional differential equations, *Memoirs on Differential Equations and Mathematical Physics*, **72** (2017), 1-14.

[2] R. Agarwal, A. Golev, S. Hristova, D. O'Regan, and K. Stefanova, Iterative techniques with computer realization for the initial value problem for Caputo fractional differential equations, *J. Appl. Math. Comput.* (2017), 1-35, **doi:** 10.1007/s12190-017-1152-x.

[3] A. Golev, S. Hristova, A. Rahnev, An algorithm for approximate solving of differential equations with "maxima", *Computers and Mathematics with Applications*, **60** (2010), 2771-2778.

[4] R. Agarwal, A. Golev, S. Hristova, D. O'Regan, Iterative techniques with computer realization for initial value problems for Riemann-Liouville fractional differential equations, *Submitted*.

[5] Altimesh, *Altimesh's Hybridizer*, http://www.altimesh.com/, 2018.

[6] Sh. Das, *Functional Fractional Calculus*, Springer-Verlag, Berlin-Heidelberg, 2011.

[7] A. Golev, S. Hristova, K. Stefanova, *Approximatee Methods and Computer Realization, Differential, Difference and Fractional Differential Equations*, Lambert Academic Publishing, 2018.

[8] A. Golev, S. Hristova, Computer realization of some approximate methods for fractional differential equations, In: *Proceedings of the Scientific Conference "Innovative Software Tools and Technologies with Applications in Research in Mathematics, Informatics and Pedagogy of Education"*, Pamporovo, 23-24 November 2017, 15-24, In Bulgarian.

[9] I. Podlubny, *Fractional Differential Equations*, Academic Press, San Diego, 1999.