Short tutorial: Solving fractional differential equations by Matlab codes

Roberto Garrappa
Department of Mathematics
University of Bari - Italy
roberto.garrappa@uniba.it

June 30, 2014

1 Introduction

Fractional differential equations (FDEs) are becoming a very popular topic and several real–life phenomena are described and modeled by means of scalar or systems of FDEs.

Whilst most of the mathematical packages provide well designed and robust routines for solving ordinary differential equations, very few codes are available for the numerical treatment of FDEs.

The aim of this short tutorial is to describe some Matlab codes recently written for solving FDEs and provide some examples for their use.

2 The Matlab codes `fde12.m` and `flmm.m`

The Matlab code `fde12.m` and `flmm.m` are devised to numerically solve FDEs and they are freely available on the file exchange service of Matlab central.

The code `fde12.m` implements the Predictor–Corrector method proposed by Diethelm and Freed in [1]. This methods is a combination of some product integration rules, usually known as fractional Adams–Bashforthm–Moulton methods and its stability properties were studied in [3]. The code is available in the file exchange service of Matlab central at http://www.mathworks.com/matlabcentral/fileexchange/32918

The code `flmm.m` implements some fractional linear multistep methods (FLMMs) introduced by Lubich in [5] of the second order. In particular the code implements three different implicit methods: a generalization of the classical Trapezoidal rule (sometimes referenced as the Tustin method), a generalization of the Newton-Gregory formula and the generalization of a Backward Differentiation Formula (BDF). We refer to [2] for a description of the way in which the methods are implemented and for a discussion of the various cases in which one method is preferable to the other. Also this code is available in the file exchange service of Matlab central at

http://www.mathworks.com/matlabcentral/fileexchange/47081

All the methods are based on discrete convolution quadrature rules. To keep at the lowest possible level the amount of computation required by the solution of FDEs, the convolution quadrature rule are evaluated by means of the FFT algorithm described in [4] allowing to reduce the computational effort from N^2 to $N(\log N)^2$, where N is the number of time–points on the interval of integration $[t_0, T]$.

We suggest to carefully read the explanations included in each code for a complete description of the parameters and their use. The explanations can be also read by means of the Matlab `help` command as `help fde12` or `help flmm2`.

2.1 Use of the code `fde12.m`

To solve a FDE by the `fde12.m` code the main command is

```
> [T,Y] = FDE12(ALPHA,FDEFUN,T0,TFINAL,Y0,h)
```

where `ALPHA` is the order of the fractional derivative, `FDEFUN` the vector field of the FDE, `T0` the starting point, `TFINAL` the ending point of the interval of integration, `Y0` the initial value and `h` the stepsize.

For a scalar `T` and a vector `Y`, `FDEFUN(T,Y)` must return a column vector corresponding to the vector field $f(t, y)$. It is possible to specify some parameters for `FDEFUN` by introducing the optional argument `PARAM`; in this case the vector field is evaluated as `FDEFUN(T,Y,PARAM)` and `fde12.m` is called as

```
> [T,Y] = FDE12(ALPHA,FDEFUN,T0,TFINAL,Y0,h,PARAM)
```

We refer to `help fde12` for the meaning and the use of the other parameters.

2.2 Use of the code `flmm.m`

The code `flmm.m` is used in a very similar way to `fde12.m` but there are two main differences:

- since `flmm.m` implements three different methods, the parameter `METHOD` is used to specify the selected method;
- being the methods implemented by `flmm.m` of implicit type, it is necessary to define the Jacobian of the vector field in order to solve at each time–step the (usually nonlinear) system of algebraic equations.

The basic syntax for solving and FDE by the code `flmm.m` is therefore

```
> [T,Y] = FLMM2(ALPHA,FDEFUN,JFDEFUN,T0,TFINAL,Y0,h,PARAM,METHOD)
```

where `JFDEFUN` defines the Jacobian of the vector field `FDEFUN` and `METHOD` the method selected for solving the FDE. The options for `METHOD` are: 1 for the Trapezoidal method, 2 for the Newton-Gregory formula and 3 for the BDF-2 (see [5, 2]). The parameter `METHOD` and is optional; when not defined by default it is assumed `METHOD=3`.

We refer to `help flmm2` for the meaning and the use of the other parameters.

3 Examples

We provide here an example of application of the codes presented in this tutorial. In particular we consider, as test problem, the fractional version of the Brusselator model problem

$$\left\{ \begin{array}{l} {}_{t_0}D_t^{\alpha} y_1(t) = a - (\mu+1) y_1(t) + \left(y_1(t)\right)^2 y_2(t) \\ {}_{t_0}D_t^{\alpha} y_2(t) = \mu y_1(t) - \left(y_1(t)\right)^2 y_1(t) \\ y_1(t_0) = y_{1,0} \\ y_2(t_0) = y_{2,0} \end{array} \right. , \quad 0 < \alpha < 1 \tag{1}$$

We assume that we need to integrate this FDE for the order $\alpha = 0.8$ in the interval $[0, 100]$ with a step–size $h = 2^{-6}$. Moreover, we assume that the main parameters has the values $a = 1$ and $\mu = 4$.

The vector field of the test problem is

$$f(t, y(t)) \equiv f(y(t)) = \left(\begin{array}{c} f_1(y(t)) \\ f_2(y(t)) \end{array} \right) = \left(\begin{array}{c} a - (\mu+1) y_1(t) + \left(y_1(t)\right)^2 y_2(t) \\ \mu y_1(t) - \left(y_1(t)\right)^2 y_1(t) \end{array} \right)$$

and it is easy to evaluate that the Jacobian

$$J_f(t, y(t)) = \begin{pmatrix} \dfrac{\partial}{\partial y_1} f_1(y(t)) & \dfrac{\partial}{\partial y_2} f_1(y(t)) \\ \dfrac{\partial}{\partial y_1} f_2(y(t)) & \dfrac{\partial}{\partial y_2} f_2(y(t)) \end{pmatrix}$$

of this function is given by

$$J_f(t, y(t)) \equiv J_f(y(t)) = \begin{pmatrix} -(\mu + 1) + 2y_1(t)y_2(t) & (y_1(t))^2 \\ \mu - 2y_1(t) & -(y_1(t))^2 \end{pmatrix}$$

There are two main ways to describe the above vector field and the corresponding Jacobian. With recent versions of Matlab it is possible to define

```
a = 1 ; mu = 4 ;
fdefun = @(t,y) [ a-(mu+1)*y(1)+y(1)^2*y(2) ; mu*y(1)-y(1)^2*y(2) ] ;
Jfdefun = @(t,y) [ -(mu+1)+2*y(1)*y(2) , y(1)^2 ; mu-2*y(1)*y(2) , -y(1)^2 ] ;
```

Note that the Jacobian `Jfdefun` is used only by the `flmm2.m` code; in the case of `fde12.m` it is sufficient to define only `fdefun`.

For the other parameters (order, interval of integration, initial value and step–size) we use the assignments

```
alpha = 0.8 ;
t0 = 0 ; tfinal = 100 ; y0 = [ 0.2 ; 0.03] ;
h = 2^(-6) ;
```

It is hence possible to solve the test problem by means of the following calls

```
[t, y_flmm2] = flmm2(alpha,fdefun,Jfdefun,t0,tfinal,y0,h) ;

[t, y_fde12] = fde12(alpha,fdefun,t0,tfinal,y0,h) ;
```

To plot the solution the following few Matlab commands can be used

```
figure(1)
plot(t,y_flmm2(1,:),t,y_flmm2(2,:)) ;
xlabel('t') ; ylabel('y(t)') ;
legend('y_1(t)','y_2(t)') ;
title('FDE solved by the FLMM2.m code') ;
```

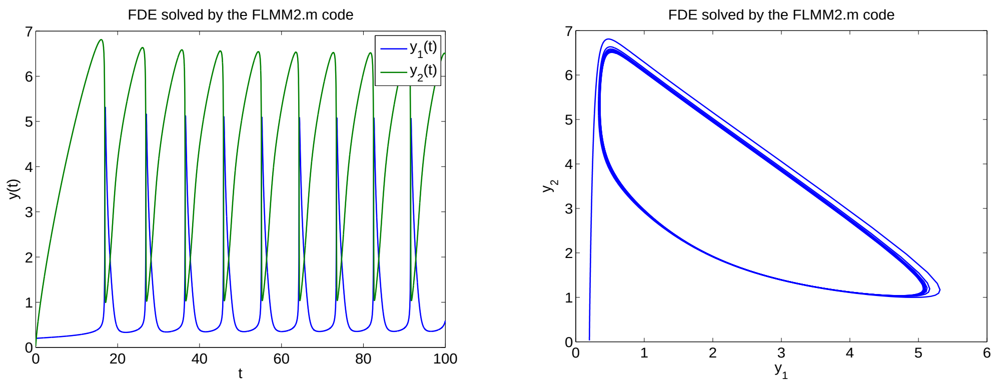

Figure 1: Solution of the Brusselator test problem in the plane (t, y) and in the phase plane.

and similarly for plotting the results from the `fde12.m` code. The results are shown in Figure 1.

An alternative approach (usually necessary in old versions of Matlab) to define the vector field is by means of external functions of this kind

```
function y = f_Bruss(t,x,param)
a = param(1) ; mu = param(2) ;
y(1,1) = a -(mu+1)*x(1)+x(1)^2*x(2) ;
y(2,1) = mu*x(1)-x(1)^2*x(2) ;
```

which must be saved in a file named as `f_Bruss.m`. similarly, for the corresponding Jacobian matrix we define the external function

```
function J = Jf_Bruss(t,x,param)
a = param(1) ; mu = param(2) ;
J(1,1) = -(mu+1)+2*x(1)*x(2) ;
J(1,2) = x(1)^2 ;
J(2,1) = mu-2*x(1)*x(2) ;
J(2,2) = -x(1)^2 ;
```

which will be saved in a file named as `Jf_Bruss.m`.

The way in which the FDE is solved by the codes `flmm2.m` and `fde12.m` is slighly modified since the vector field and its Jacobian are given now as the strings containing their names and the optional variable `PARAM` must be used in order to communicate the values of the parameters a and μ

```
a = 1 ; mu = 4 ;
param = [a , mu] ;
[t, y_flmm2] = flmm2(alpha,'f_Bruss','Jf_Bruss',t0,tfinal,y0,h,param) ;
[t, y_fde12] = fde12(alpha,'f_Bruss',t0,tfinal,y0,h,param) ;
```

References

[1] Kai Diethelm and Alan D. Freed. The FracPECE subroutine for the numerical solution of differential equations of fractional order. In S.Heinzel and T.Plesser, editors, *Forschung und wissenschaftliches Rechnen 1998*, pages 57–71. 1999.

[2] R. Garrappa. Trapezoidal methods for fractional differential equations: Theoretical and computational aspects. *Mathematics and Computers in Simulation*, 2013. In press. DOI http://dx.doi.org/10.1016/j.matcom.2013.09.012.

[3] Roberto Garrappa. On linear stability of predictor–corrector algorithms for fractional differential equations. *Int. J. Comput. Math.*, 87(10):2281–2290, 2010.

[4] E. Hairer, Ch. Lubich, and M. Schlichte. Fast numerical solution of weakly singular Volterra integral equations. *J. Comput. Appl. Math.*, 23(1):87–98, 1988.

[5] Christian Lubich. Discretized fractional calculus. *SIAM J. Math. Anal.*, 17(3):704–719, 1986.