Numerical Solution of a Nonlinear Dynamical System
by a Collocation Method based on Linear B-spline Basis

Author : Francesca Pitolli

References :

1) F. Pitolli, Fractals&Fractionals, 2018
2) F. Pitolli, Axioms, 2018
3) E. Pellegrino, L. Pezza, F. Pitolli, submitted, 2019
4) F. Pitolli, Algorithms, 2019

Version: July, 2019

Define the Nonlinear Dynamical System

Test Problems :

1) Linear dynamical system
2) Nonlinear dynamical system

```
(* Known term of the dynamical system *)
Clear[y, z, w, F, JF]
Problem = 1;
Which[Problem == 1, F = {y + z + w, -y + z - w, -Gamma[3/2] w};,
 Problem == 2, F = {y + z + 100 Sqrt[π] y^2 + w, -y + z + 100 Sqrt[π] y^2 - w, w^2 - 1/200 Sqrt[π] w};
]
Y = {y, z, w};
nF = Dimensions[F][[1]];

(* Initial conditions *)
Clear[Y0]
Y0 = {0.05, 0.05, 0.05};

(* Jacobian matrix of F *)
Clear[JF]
JF = Outer[D, F, Y];
MatrixForm[JF]
```

(* Order of the time fractional derivative (0 < b < 1) *)

```
b = 0.5;
```

Basis Functions

Left-edge boundary function

```
In[ ]:= (* Left edge function *)
        Clear[N01]

        N01[x_] := Piecewise[{{(1 - x), 0 ≤ x ≤ 1}}]
```

Linear B - Spline

```
In[ ]:=
        Clear[B1]
        B1[x_] := Piecewise[{{x, 0 ≤ x ≤ 1}, {2 - x, 1 ≤ x ≤ 2}}]
```

Caputo derivative of fractional order b (0 < b < 1)

Caputo Derivative of the interior functions

```
In[ ]:= (* Derivative of the left edge function *)
        Clear[derN01]

        derN01[x_, b_] := 1/Gamma[2 - b] Piecewise[{{-x^(1-b), 0 ≤ x ≤ 1}, {-x^(1-b) + (x - 1)^(1-b), 1 ≤ x}}]
```

Caputo Derivative of the linear B – spline

In[•]:=
```
Clear[derB1]
```

$$\text{derB1}[x\_, b\_] := \frac{1}{\text{Gamma}[2-b]}\,\text{Piecewise}\left[\left\{\left\{x^{1-b}, 0 \le x \le 1\right\}, \left\{x^{1-b} - 2\,(x-1)^{1-b}, 1 \le x \le 2\right\}, \left\{x^{1-b} - 2\,(x-1)^{1-b} + (x-2)^{1-b}, 2 \le x\right\}\right\}\right]$$

Inputs for the collocation method

In[•]:=
```
(* Discretization interval: [0,T] *)
T = 8;

(* Refinement  step *)
js = 4;
h = 1/2^js

(* Number of interior basis functions *)
Nkint = T / h;

(* Number of left edge basis functions *)
Nkedge = 1;

(* Number of basis functions *)
Nk = Nkint + Nkedge
```

Collocation Points

```
In[•]:= Clear[TPoints]
        dt = h/2;
        Ns = T / dt
        TPoints = Table[i dt, {i, 1, Ns}]

        If[Ns < Nk, Print["ERROR: Ns should be greater than or equal to Nk"]]
```

Collocation Matrix for the Cubic B-spline Basis

```
In[•]:= (* Interior functions *)
        Clear[Mcoll0int]
        Mcoll0int = Table[N[B1[TPoints[[i]]/h - k]], {i, 2, Ns}, {k, 0, Nkint - 1}];

        (* Collocation matrix *)
        Mcoll0 = Transpose[Mcoll0int];
        Dimensions[Mcoll0]
        Clear[Mcoll0int]
```

Collocation Matrix for the Caputo Derivative of the B-spline Basis

```
In[ ]:= (* Normalization factor for the derivative*)
        normder = 2^(js b);

        (* Interior functions *)
        Clear[Mcoll1int]
        Mcoll1int = Table[N[normder derB1[TPoints[[i]]/h - k, b]], {i, 2, Ns}, {k, 0, Nkint - 1}];

        (* Collocation matrix *)
        Mcoll1 = Transpose[Mcoll1int];
        Dimensions[Mcoll1]
        Clear[Mcoll1int]
```

Solution of the Nonlinear System

Iteration loop

```
In[ ]:= (* Assign the initial guess and the parameters*)
        Clear[lambda0]
        lambda0 = Table[0., {i , 1, nF}, {j, 1, Nk - 1}] ;
        tau = 0.1;
        Normlam = 10 tau;
        maxiter = 10;
        ell = 0;

        While[Normlam > tau && ell < maxiter,

         (* Evaluate the approximate solution at the previous iteration step *)
         Clear[Ys];
         Ys = Table[Transpose[Mcoll0].lambda0[[n]] , {n, 1, nF}] + Y0;
```

```
Print[ListPlot[Table[{TPoints[[i]], Ys[[k, i - 1]]}, {k, 1, nF}, {i, 2, Ns}],
  Joined → True, PlotRange → All, PlotLabel → "Solution"]];

(* Evaluate the Jacobian matrix at the previous iteration step *)
JFNodes = Table[JF /. {y -> Ys[[1, i]], z -> Ys[[2, i]], w -> Ys[[3, i]]} , {i, 1, Ns - 1}];

(* Construct the second term of the Jacobian matrix
  J_G by multipling the Jacobian matrix J_F with the collocation B-spline matrix *)
JFblock = Table[Outer[Times, JFNodes[[i]], Mcoll0[[i]]], {i, 1, Nk - 1}];
JJ = ArrayFlatten[Transpose[JFblock, {4, 1, 2, 3}], 2];

(* Construct the Jacobian matrix J_G *)
Clear[Mcoll];
Mcoll = KroneckerProduct[IdentityMatrix[nF], Transpose[Mcoll1]] - JJ;

(* Construct the known term *)
Clear[BG];
BG = KroneckerProduct[IdentityMatrix[nF], Transpose[Mcoll1]] .Flatten[lambda0] -
  Flatten[Transpose[Table[F /. {y -> Ys[[1, i]], z -> Ys[[2, i]], w -> Ys[[3, i]]} , {i, 1, Ns - 1}]]];

(* Solve the linear system by the least squares method *)
Clear[GamK];
GamK = LeastSquares[Mcoll, N[BG]];

(* Update the unknown coefficients *)
Clear[lambda];
lambda = lambda0 - Partition[GamK, Nk - 1];

(* Evaluate the infinity norm of the relative error *)
Normlam = Norm[lambda - lambda0, Infinity] / Norm[lambda, Infinity];

(* Update the iteration counter and the privious solution *)
ell ++;
lambda0 = lambda;
```

```
 (* Print the relative error at iteration ell *)
 Print["Iteration = ", ell, "   Norm = ", Normlam];
 ]
```

Evaluate the Numerical Solution

In[•]:=
```
Clear[B1sint, B1s]

(* Points where to evaluate the numerical solution *)
h2 = h/4;
Nr = T/h2 + 1;
Tval = Table[i h2, {i, 0, Nr - 1}];

(* Function basis on the evaluation points *)
B1sint = Table[N[B1[Tval[[i]]/h - k]], {i, 1, Nr}, {k, 0, Nkint - 1}];
B1s = Transpose[B1sint];

(* Print the numerical solution *)
Clear[Yh]
Yh = Table[Transpose[B1s].lambda[[n]], {n, 1, nF}] + Y0;
ListPlot[Table[{Tval[[i]], Yh[[k, i]]}, {k, 1, nF}, {i, 1, Nr}], Joined → True, PlotRange → All]
```