Proceeding of 2nd International Science Postgraduate Conference 2014 (ISPC2014)

Implicit Second and Third Orders Runge-Kutta for Handling Discontinuities in Delay Differential Equations

[1]LIM RUI SIH, [2*]ROHANIN AHMAD AND [3]YEAK SU HOE

[1,2,3]Department of Mathematical Sciences, Faculty of Science
Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia

[1]ruisih64@gmail.com, [2*]rohanin@utm.my, [3] s.h.yeak@utm.my

*Corresponding author

Abstract. Implicit Runge-Kutta (RK) methods have been developed and implemented in solving Delay Differential Equations (DDEs) systems which often encounter discontinuities. These discontinuities might occur after and even before the initial solution. The methods are chosen because they can be modified to handle discontinuities by means of mapping of past values and they are in fact the most well-organized way to handle the so-called stiff differential equations, which are differential equations usually characterized by a rapidly decaying solution. The advantage of implicit Runge-Kutta methods is in their superior stability compared to the explicit methods, more so when applied to stiff equations. Our objective is to develop a scheme for solving DDEs using implicit RK2 and RK3. Our numerical scheme is able to successfully handle discontinuities in the system and produces results with acceptable error. We compare the result from [1] which used explicit RK2 and RK4 with our findings. Our result is markedly better than [1] even in the lower order RK.

Keywords Delay differential equations, Runge-Kutta method, dde23.

1.0 INTRODUCTION

Delay differential equations (DDEs) are currently found in problems of engineering as well as in biological and physical interests around us subjected to time delay which may instigate instability or poor performance. There are large and useful classes of dynamics systems among which some are very important. On the other hand, to make the model more consistent with the real system, it is

necessary to modify this equation by including past values of state variables, Bellen and Zennaro [2]. In this work we consider the implicit third-order Runge-Kutta (RK3) method for solving the DDEs. This is because the implicit numerical schemes are more effective than their explicit counterpart in approximating the solution of differential equations; they are also more stable and able to solve efficiently stiff problems. When problems become increasingly stiff, stability rather than accuracy becomes the dominant consideration, and implicit methods become the more suitable option. Furthermore, the RK3 method is stable without damping. Here, we will focus only on systems of delay differential equations of the form

$$y'(t) = f(t, y(t), y(t-\tau_1), y(t-\tau_2), \ldots, y(t-\tau_i), \ldots) \qquad (1)$$

where the quantities $\tau_i,\ i = 1,\ 2,\ \ldots,\ m$ are positive constants and $y(t) = S(t)$ are given history/past values; the history/past values will make the model more consistent. Most of the delay differential equations are nonlinear systems, such as predator-prey systems, Xu and Chen [3], continuously-stirred tank reactors, Cao and Frank [4], tracking component-wise continuity, Willé and Baker [5] and infectious disease model from Hairer, Norsett, and Wanner [6]. This paper only concentrates on these kinds of systems and we analyze example systems with two delays; the stability in DDEs with two delays was discussed in Li, Ruan and Wei [7].

We developed a method based on implicit RK3 that is able to generate acceptable output. This output will be compared to numerical results obtained from dde23 in Matlab which is based on explicit second and third orders Runge-Kutta and interpolant of ode23 and output values from second and fourth orders Runge-Kutta discussed in, Lim, Rohanin and Yeak [1]; we expect the values of our output naturally should be as accurate as the output of dde23 if not better.

2.0 NUMERICAL DISCRETIZATION

A general form of DDE with two time lags τ_1 and τ_2 is

$$\left.\begin{aligned} y'(t) &= f(t, y(t), y(t-\tau_1), y(t-\tau_2)), t \in -\tau, T \\ y(t) &= S(t) \qquad\qquad\qquad\qquad , t \in -\tau, 0 \end{aligned}\right\} \tag{2}$$

where history function, $S(t)$ corresponds to the initial function on the interval $-\tau \le t \le 0$ with time delays τ_1, τ_2 are non-negative constant delays or time-dependent delays, $\tau_i(t)$. Here, we shall only consider the constant delays again. The presence of the initial function $S(t)$ in system (2) may contribute to discontinuities in the derivatives and affect the numerical treatment. These discontinuities can be present at times both before and after the initial point. Discontinuities will cause inaccuracy and inefficiency of numerical methods employed to solved DDEs and the exact solution of DDEs is hard to find. So the implicit Runge-Kutta methods are chosen in current paper because this method can be easily modified to handle the discontinuities by using mappings; easier to apply than other popular numerical methods; and they are in fact the well-organized way to handle the so-called stiff differential equations. Some numerical methods are known for becoming numerically unstable when solving these kinds of problems, unless the step size taken is really small. Implicit Runge-Kutta methods are more stable and effortlessly adaptable to a variable step-size which can be adjusted from step to step in a simple way. In this paper the Runge-Kutta methods used are of lower order, but the result is better than the higher order explicit Runge-Kutta in [1].

Here, the RK2 and the implicit RK3 methods will be used to calculate the node-points. RK2 will solve the problem with step-size, $\frac{h}{2}$; whereas RK3 will solve the problem with step-size, h.

The general implicit RK3 method:

$$\left.\begin{aligned} y(t+h) &\approx y(t)+\frac{1}{6}F_1^{\,N}+\frac{2}{3}F_2^{\,N}+\frac{1}{6}F_3^{\,N} \\ F_1^{\,n+1} &= hf\left(t, y+\frac{1}{6}F_1^{\,n}-\frac{1}{6}F_2^{\,n}\right) \\ F_2^{\,n+1} &= hf\left(t+\frac{1}{2}h, y+\frac{1}{6}F_1^{\,n+1}+\frac{1}{3}F_2^{\,n}\right) \\ F_3^{\,n+1} &= hf\left(t+h, y+\frac{1}{6}F_1^{\,n+1}+\frac{5}{6}F_2^{\,n+1}\right) \end{aligned}\right\} (3)$$

where $y' = f(t,y)$ with h being the step size and N is final iteration.
The RK2 method also called Heun's method is as follows:

$$\begin{aligned} y(t+h) &\approx y(t)+\frac{1}{2}\ k_1+k_2 \\ k_1 &= hf(t,y) \\ k_2 &= hf(t+h, y+k_1) \end{aligned}$$

When we apply $H=\frac{h}{2}$ in RK2,

$$\left.\begin{aligned} y(t+h) &\approx y(t)+\frac{1}{2}\ k_1+k_2 \\ k_1 &= Hf(t,y) \\ k_2 &= Hf(t+H, y+k_1) \end{aligned}\right\} (4)$$

General process for DDEs Algorithm

Step 0: Set the history/past values and initial value which are given.
Step 1: Identity the time delays τ_i by using Greatest Common Divisor (GCD). The biggest time delay will be set as the "Step".

Step 2: Solve the discretized state system in Equation (2) by using Equations (3) and (4).
Step 3: Set $t = t + h$ and go to Step 2.

The Runge-Kutta methods mentioned are combination of explicit and implicit recipes for computations y_{n+1} given y_n and have the ability to evaluate the equation or system. For reasons of efficiency, the longest delay τ_i will be set as "Step", then the h will set as "node", hence this will yield the specified accuracy. Normally the shortest delay τ_i will be set as h, all the needed values of the solution in the span of the step will be stored automatically and easy to recall when we update $t = t + h$.

3.0 NUMERICAL EXAMPLES

In this section, we use problems from the literature to show solution of DDEs with dde23 from Matlab and our current method using implicit RK 2 and RK 3, and compare their output. For the purpose of comparison of results, examples from [1] are selected and reproduced here.

Example 1: Tracking component-wise solution continuity.
The system is defined as

$$\begin{aligned} y_1'(t) &= y_1(t-1) \\ y_2'(t) &= y_1(t-1) + y_2(t-0.2) \\ y_3'(t) &= y_2(t) \end{aligned}$$

to be solved over the interval [0,3] with history $y_1(t) = 1,\ y_2(t) = 1,\ y_3(t) = 1$ for $t \leq 0$.

In the numerical computations, the following parameters were used: the number of steps, steps = 0, 1, 2, 3, where steps = 0 is history; the number of

nodes, node = 1, 2, 3, …, 11; h = shortest time delay τ = 0.2, and $\frac{h}{2} = 0.1$.

The result from using dde23 in Matlab is tabulated in Table 1 below. The format for all tables in this paper will be; first column represents time, the following columns are for the output of y_1, y_2, and y_3 respectively.

Table 1: Output values of dde23

Output from dde23			
Time	y_1	y_2	y_3
0.0000	1.0000	1.0000	1.0000
0.2000	1.2000	1.4000	1.2400
0.4000	1.4000	1.8400	1.5627
0.6000	1.6000	2.3627	1.9815
0.8000	1.8000	2.9815	2.5141
1.0000	2.0000	3.7141	3.1816
1.2000	2.2200	4.6016	4.0101
1.4000	2.4800	5.6900	5.0356
1.6000	2.7800	7.0155	6.3019
1.8000	3.1200	8.6218	7.8606
2.0000	3.5000	10.5604	9.7728
2.2000	3.9213	12.8939	12.1110
2.4000	4.3907	15.7014	14.9619
2.6000	4.9160	19.0775	18.4294
2.8000	5.5053	23.1343	22.6382
3.0000	6.1667	28.0044	27.7373

The following Table 2 tabulates the result from using our scheme.

Table 2: Output values of Implicit RK2 and RK3

Output values of Implicit RK2 and RK3			
Time	y_1	y_2	y_3
0.0000	1.0000	1.0000	1.0000
0.0200	1.0000	1.0000	1.0000
0.0400	1.0400	1.0800	1.0416
0.0600	1.0400	1.0800	1.0416
0.0800	1.0800	1.1600	1.0864

0.1000	1.0800	1.1600	1.0864
0.1200	1.1200	1.2400	1.1344
0.1400	1.1200	1.2400	1.1344
0.1600	1.1600	1.3200	1.1856
0.1800	1.1600	1.3200	1.1856
0.2000	1.2000	1.4000	1.2400
0.2200	1.2000	1.4000	1.2400
0.2400	1.2400	1.4816	1.2976
0.2600	1.2400	1.4816	1.2976
0.2800	1.2800	1.5664	1.3586
0.3000	1.2800	1.5664	1.3586
0.3200	1.3200	1.6544	1.4230
0.3400	1.3200	1.6544	1.4230
0.3600	1.3600	1.7456	1.4910
0.3800	1.3600	1.7456	1.4910
0.4000	1.4000	1.8400	1.5627
⋮	⋮	⋮	⋮
0.6000	1.6000	2.3627	1.9815
⋮	⋮	⋮	⋮
0.8000	1.8000	2.9815	2.5141
⋮	⋮	⋮	⋮
1.0000	2.0000	3.7141	3.1816
⋮	⋮	⋮	⋮
2.0000	3.5000	10.5606	9.7729
⋮	⋮	⋮	⋮
2.8000	5.5053	23.1351	22.6387
⋮	⋮	⋮	⋮
3.0000	6.1667	28.0053	27.7379

In Table 2, the step-size remains the same that is 0.02; and all the values will be stored automatically making it easier to be recalled. The step size was chosen as deemed appropriate for the problem. If a smaller step-size is chosen, the values of output will be more accurate because it involves more calculation. Table 3 below displays the error between dde23 and implicit RK 2 with RK 3.

Table 3: Error between dde23 and implicit RK 2 with RK 3

Error			
Time	y_1	y_2	y_3
0.0000	0.0000	0.0000	0.0000
0.2000	0.0000	0.0000	0.0000
0.4000	0.0000	0.0000	0.0000
0.6000	0.0000	0.0000	0.0000
0.8000	0.0000	0.0000	0.0000
1.0000	0.0000	0.0000	0.0000
1.2000	0.0000	0.0000	0.0000
1.4000	0.0000	0.0001	0.0000
1.6000	0.0000	0.0001	0.0000
1.8000	0.0000	0.0002	0.0001
2.0000	0.0000	0.0002	0.0001
2.2000	0.0000	0.0003	0.0002
2.4000	0.0000	0.0004	0.0002
2.6000	0.0000	0.0006	0.0003
2.8000	0.0000	0.0007	0.0005
3.0000	0.0000	0.0010	0.0006

From the table, the values from time 0.0000 to 1.2000 are exactly the same as the values from the application of dde23. This signifies that our coding of implicit RK 2 and RK3 is correct. By computing the difference between the output values from Table 1 and Table 2, we find that the maximum error is 0.0010. This means that the implicit RK 2 and RK 3 is acceptable to use. Notice that the error starts materializing from $t = 1.4000$ onwards. In [1], the maximum error between dde23 and RK 2 with RK 4 is 0.0027 when time is 3.000. This shows that the scheme is better. From Table 3, we noted that the values of y_2 contributed to the largest error.

The following Example 2 is an example where the increment of time is nonuniform when using Matlab; this is the instance when discontinuities may occur. But it never will happen in our scheme.

Example 2: An infectious disease model.

The system is presented as

$$y_1'(t) = -y_1(t)y_2(t-1) + y_2(t-10)$$
$$y_2'(t) = y_1(t)y_2(t-1) - y_2(t)$$
$$y_3'(t) = y_2(t) - y_2(t-10)$$

to be solved over the interval [0, 30] with history $y_1(t) = 5,\ y_2(t) = 0.1,\ y_3(t) = 1$ for $t \le 0$, shortest time delay $\tau = 1.0,\ h = 0.2$, and $\frac{h}{2} = 0.1$. Again we will solve the problem using both dde23 from Matlab and implicit RK 2 and RK 3 from our scheme. Table 4 below is the output from the application dde23.

Table 4: Output values of dde23

Output of dde23			
Time	y_1	y_2	y_3
0.0000	5.0000	0.1000	1.0000
0.0200	4.9920	0.1079	1.0001
0.1200	4.9523	0.1450	1.0028
0.3287	4.8706	0.2102	1.0192
0.5838	4.7732	0.2714	1.0554
0.7919	4.6955	0.3094	1.0952
1.0000	4.6194	0.3387	1.1419
⋮	⋮	⋮	⋮
3.0000	2.2250	1.3302	2.5448
⋮	⋮	⋮	⋮
10.0000	0.3325	0.0389	5.7286
⋮	⋮	⋮	⋮
11.0000	0.5538	0.0274	5.5187
⋮	⋮	⋮	⋮
12.0000	1.0755	0.0259	4.9986
⋮	⋮	⋮	⋮
12.8000	1.8462	0.0327	4.2211
⋮	⋮	⋮	⋮
20.0000	0.1700	0.8729	5.0571
⋮	⋮	⋮	⋮
21.0000	0.0707	0.3950	5.6343
⋮	⋮	⋮	⋮

28.6888	3.3104	0.0225	2.7671
28.9945	3.7665	0.0288	2.3047
29.2890	4.1642	0.0371	1.8987
29.5786	4.5000	0.0483	1.5518
29.7893	4.7053	0.0588	1.3359
30.0000	4.8765	0.0718	1.1516

From Table 4, the increment of time is nonuniform, hence we cannot tract past values of the system at any particular instance. Again, the discontinuities may occur. For instance, we cannot get the past value at $t = 0.7000$ because dde23 would not allow it. This is due to users' inability to influence the step size in dde23.

In contrast, our scheme allows us to determine the number of nodes as needed by varying the step size. Hence all values can be read, passed, and stored automatically. In Table 5 we tabulate the result from applying our method. The increment of time remains linear with $h = 0.2,\ \frac{h}{2} = 0.1.$

Table 5: Output values of Implicit RK2 and RK3

Output values of Implicit RK2 and RK3			
Time	y_1	y_2	y_3
0.0000	5.0000	0.1000	1.0000
0.1000	5.0000	0.1000	1.0000
0.2000	4.9208	0.1718	1.0074
0.3000	4.9208	0.1718	1.0074
0.4000	4.8432	0.2291	1.0277
0.5000	4.8432	0.2291	1.0277
0.6000	4.7671	0.2747	1.0583
0.7000	4.7671	0.2747	1.0583
0.8000	4.6925	0.3106	1.0970
0.9000	4.6925	0.3106	1.0970
1.0000	4.6194	0.3386	1.1420
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
2.0000	3.7136	0.7986	1.5878
⋮	⋮	⋮	⋮
3.0000	2.2254	1.3302	2.5443
⋮	⋮	⋮	⋮

10.0000	0.3329	0.0392	5.7280
⋮	⋮	⋮	⋮
11.0000	0.5540	0.0277	5.5184
⋮	⋮	⋮	⋮
12.0000	1.0752	0.0262	4.9986
⋮	⋮	⋮	⋮
12.8000	1.8456	0.0330	4.2214
⋮	⋮	⋮	⋮
20.0000	0.1707	0.8643	5.0650
⋮	⋮	⋮	⋮
21.0000	0.0717	0.3919	5.6364
⋮	⋮	⋮	⋮
30.0000	4.8725	0.0733	1.1541

Obviously, the increment of time is linearly and controlled by the user through the choice of the step size, hence we can tract past values at any instant we desire. Say we require to get the past value at $t = 0.7500$ which is not in Table 5. All that we need to do is to change the step size to $h = 0.1,\ \frac{h}{2} = 0.05$ which we definitely cannot do with dde23.

The error between the result from dde23 and our scheme is shown in Table 6.

Table 6: Error between dde23 and implicit RK 2 and RK 3

Error			
Time	y_1	y_2	y_3
0.0000	0.0000	0.0000	0.0000
0.1000	0.0000	0.0001	0.0001
2.0000	0.0002	0.0001	0.0001
3.0000	0.0004	0.0001	0.0005
10.0000	0.0003	0.0003	0.0007
11.0000	0.0001	0.0002	0.0004
12.0000	0.0003	0.0002	0.0001
12.8000	0.0006	0.0003	0.0003
20.0000	0.0007	0.0086	0.0079
21.0000	0.0011	0.0032	0.0021
30.0000	0.0040	0.0015	0.0025

Performance-wise, our current method is just as good an approximator as dde23 and explicit RK method in [1]. This is evident from the errors tabulated in Table 6, where the maximum error is 0.0086; and in [1], the maximum error is 0.0089. The entries in Table 6 are only those nodes comparable from both methods.

4.0 CONCLUSION

We have successfully developed another scheme for overcoming the discontinuities specifically at the point of determination of the step size in DDE systems. From our findings lower order implicit Runge-Kutta has the better performance than higher order explicit Runge-Kutta. To authenticate our achievement, we always compare our results with results from dde23. Here, we presented two cases of DDEs. Numerical results indicate that our method based on implicit RK 2 and RK 3 performs quite well compared to dde23 from Matlab and the scheme from [1].

ACKNOWLEDGMENTS

We would like to acknowledge Kementerian Pendidikan Malaysia and Universiti Teknologi Malaysia for their support through MyBrain15 (MyPhD), and the research grant FRGS 4F181.

REFERENCES

[1] R. S. Lim, A. Rohanin and S. H. Yeak. "History Tracting Ability of Second and Fourth Orders Runge-Kutta in Solving Delay Differential Equations," *IGCESH Publication.* 878-882. 2013.

[2] Bellen, A. and Zennaro, M. "Numerical Methods for Delay Differential Equations," *Oxford: Oxford Science Publication.* 2003.

[3] R. Xu and L. Chen, "Persistence and stability for a two-species ratio-dependent predator-prey system with time delay in a two-patch environment," *Comput. Math. Applic.,* 40(4-5): 577-588,2000.

[4] Y. Y. Cao and P. M. Frank, "Analysis and synthesis of nonlinear time-delay systems via fuzzy control approach," *IEEE Trans. Fuzzy Syst.,* 8(2): 200-211, Apr. 2000.

[5] D.R. Willé and C.T.H. Baker, "DELSOL – a numerical code for the solution of systems of delay-differential equations," *Appl. Numer. Math.,*9: 223-234, 1992.

[6] E. Hairer, S. P. Norsett, and G. Wanner, Solving Ordinary Differential Equations I, *Springer-Verlag,* Berlin, 1987.

[7] X. Li, S. Ruan and J. Wei., "Stability and Bifurcation in Delay-Differential Equations with Two Delays," *Journal of Mathematical Analysis and Appl.,* 236: 254-280, 1999.

[8] Christopher, T. H. B., and Evelyn, B., "Numerical Analysis of Explicit One-Step Methods for Stochastic Delay Differential Equations," *LMS J. Comput. Math.* 3: 315-335, 2000.

[9] Kloeden, P. and Schropp, J., "Runge-Kutta methods for monotone differential and stochastic equations," *Proc. Appl. Math. Mesh.,* 3: 565-566, 2003.

[10] Küchler, U. and Platen, E., "Strong Discrete Time Approximation of Stochastic Differential Equations with Time Delay," *Mathematics and Computers in Simulation,* 54(1–3): 189-205, 2000.

[11] Platen, E., "An Introduction to Numerical Methods for Stochastic Differential Equations," *Acta Numerica,* 8: 197-246,1999.

[12] Kloeden, P. E. and Platen, E., "Numerical Solution of Stochastic Differential Equations," *Appl. Math. Springer,* 23. Third corrected printing, 1999.

[13] Shampine, L.F. and S. Thompson, "Solving DDEs in MATLAB," *Appl. Numer. Math.*, 37: 441-458, 2001.