Pertanika J. Sci. & Technol. 21 (1): 37 - 44 (2013)

SCIENCE & TECHNOLOGY

Journal homepage: http://www.pertanika.upm.edu.my/

Solving Delay Differential Equations by Using Implicit 2-Point Block Backward Differentiation Formula

Heng, S. C.*, Ibrahim, Z. B., Suleiman, M. and Ismail, F.

Mathematics Department, Faculty of Science, Universiti Putra Malaysia, 43400 Serdang, Selangor, Malaysia

ABSTRACT

In this paper, an implicit 2-point Block Backward Differentiation formula (BBDF) method was considered for solving Delay Differential Equations (DDEs). The method was implemented by using a constant stepsize via Newton Iteration. This implicit block method was expected to produce two points simultaneously. The efficiency of the method was compared with the existing classical 1-point Backward Differentiation Formula (BDF) in terms of execution time and accuracy.

Keywords: Block Backward Differentiation formula, Delay Differential Equations, Interpolations

INTRODUCTION

In the recent years, the Mathematics society is gradually shifting their interest into numerical treatment of DDEs due to its ability in Mathematical modelling of processes in various applications as it provides the best, if not the only realistic simulation of the observed phenomena (Ismail *et al*., 2002). This is because many physical systems possess the feature of having a delayed respond in input conditions so that the rate at which processes occur depends not only on the current state of the systems but also the past states (Ismail 1999).

Some of the known application areas of DDEs include the fields of Engineering, Biology and Economy, such as in mixing of liquid, population growth, prey-predator population model and electrodynamics (Driver 1976). The general form of the first order DDE is as follows:

$$y'(t) = f(t, y, y(t-\tau_1), \ldots y(t-\tau_n)), \quad t \geq t_0 \qquad (1)$$

$$y(t) = \phi(t), \quad t \leq t_0 \qquad (2)$$

where $\phi(t)$ is the initial function, and τ_i is the delay function with $i = 1, 2, \ldots, n$. According to Bellen and Zennaro (2003), the delays τ_i or

Article history:
Received: 18 March 2011
Accepted: 13 September 2011

E-mail addresses:
cerylyn@yahoo.com (Heng, S. C.),
zarina@math.upm.edu.my (Ibrahim, Z. B.),
msuleiman@science.upm.edu.my (Suleiman, M.)
fudziah@science.upm.edu.my (Ismail, F.)
*Corresponding Author

ISSN: 0128-7680

lags as known to some, and which are always non-negative, can be divided into three different situations according to the complexity of the phenomenon. The delays may be just constant, which is referred to as the constant delay case, or the function of t, where $\tau_i = \tau_i(t)$ which is referred to as the variable or time dependant delay case, or even the function of t and y itself, where $\tau_i = \tau_i\left(t, y\left(t\right)\right)$, which is referred to as the state dependant delay case.

Currently, most of the numerical methods for solving ordinary differential equations (ODEs) can be adapted to give corresponding techniques in solving DDEs. For example, stiff DDEs are recognized through the behaviour of ODEs. This is because of the solutions in the systems of DDEs, which may contain strongly damped components that rapidly approach equilibrium states (Roth, 1980). In general, the largest source of stiff DDEs is the stiffness in the ODEs components that are without the delay term. The stiffness of a linear system of ODEs $\tilde{y}' = A\tilde{y}$ is expressed as ratio $\left(\max_i |\operatorname{Re}\lambda_i| / \min i |\operatorname{Re}\lambda_i|\right)$ where λ_i are eigenvalues of A. For this reason, there are actually many numerical methods proposed for solving DDEs. The range of the methods comprises of one-step methods, as in Euler method and Runge-Kutta methods, as well as multistep methods and block implicit methods (Ismail, 1999). Among the proposed methods, the family of Runge-Kutta is the most famous numerical method used to solve DDEs.

Research has shown various families of Runge-Kutta methods and interpolation techniques which are used to solve DDEs. Among other, Oberle and Pesch (1981) developed two numerical methods known as the Runge-Kutta-Fehlberg methods of orders 4 and 7 to solve DDEs. Hermite interpolation is used to approximate the delay term. Ismail *et al*. (2002) solved delay differential equations by using embedded Singly Diagonally Implicit Runge–Kutta method and the delay term was obtained by using Hermite interpolation.

While most numerical methods to determine the solution for DDEs are based on the Runge-Kutta formulas, some opted for Adams and backward differentiation formula (BDF) methods to solve them. For instance, Bocharov *et al*. (1996) considered the application of the linear multistep methods for the numerical solution of initial value problem for stiff delay differential equations with several constant delays. As for the approximation of delayed variables, Bocharov *et al*. (1996) used Nordsieck's interpolation technique.

BLOCK BACKWARD DIFFERENTIATION FORMULA

The purpose of this paper is to solve stiff DDEs using 2-point block backward differentiation formula (BBDF) method derived by Ibrahim *et al*. (2007). The corrector formulas are formulated as follows:

$$y_{n+1} = -\frac{1}{3}y_{n-1} + 2y_n - \frac{2}{3}y_{n+2} + 2hf_{n+1} \tag{3}$$

$$y_{n+2} = \frac{2}{11}y_{n-1} - \frac{9}{11}y_n + \frac{18}{11}y_{n+1} + \frac{6}{11}hf_{n+2} \tag{4}$$

The numerical results were then compared to the classical 1-point backward differentiation formula (BDF). The corrector formula (Lambert, 1993) is given as:

$$y_{n+1} = \frac{2}{11}y_{n-2} - \frac{9}{11}y_{n-1} + \frac{18}{11}y_n + \frac{6}{11}hf_{n+1} \tag{5}$$

In general, most numerical methods for solving differential equations produce only one new approximation value. However, the implicit block method can produce two new values simultaneously at each step. Thus, it is logically safe to presume that this particular method can reduce the timing of the calculation and its computational cost as well.

IMPLEMENTATION OF THE METHOD

Newton iteration is implemented in the method to evaluate the values of y_{n+1} and y_{n+2} for solving equation [1]. The values for both y_{n+1} and y_{n+2} at $(i+1)$ th iterative are given as $y_{n+1}^{(i+1)}$ and $y_{n+2}^{(i+1)}$. According to Ibrahim *et al*. (2007), the Newton iteration takes the form of

$$y_{n+1}^{(i+1)} = y_{n+1}^{(i)} - \frac{F_1(y_{n+1}^{(i)})}{F_1^{'}(y_{n+1}^{(i)})} \tag{6}$$

and

$$y_{n+2}^{(i+1)} = y_{n+2}^{(i)} - \frac{F_2(y_{n+2}^{(i)})}{F_2^{'}(y_{n+2}^{(i)})} \tag{7}$$

Hence,

$$\left(1 - 2h\frac{\partial f_{n+1}}{\partial y_{n+1}}\right)\left(y_{n+1}^{(i+1)} - y_{n+1}^{(i)}\right) = -y_{n+1}^{(i)} - \frac{2}{3}y_{n+2}^{(i)} + 2hf_{n+1}^{(i)} + \varsigma_1 \tag{8}$$

$$\left(1 - \frac{6}{11}h\frac{\partial f_{n+2}}{\partial y_{n+2}}\right)\left(y_{n+2}^{(i+1)} - y_{n+2}^{(i)}\right) = \frac{18}{11}y_{n+1}^{(i)} - y_{n+2}^{(i)} + \frac{6}{11}hf_{n+2}^{(i)} + \varsigma_2 \tag{9}$$

where $\frac{\partial f}{\partial y}$ is a Jacobian and ς_1, ς_2 are the back values.

Let

$$e_{n+1}^{(i+1)} = y_{n+1}^{(i+1)} - y_{n+1}^{(i)} \tag{10}$$

$$e_{n+2}^{(i+1)} = y_{n+2}^{(i+1)} - y_{n+2}^{(i)} \tag{11}$$

Then, equations [8] and [9] can be rewritten in the matrix form, as follows:

$$\begin{bmatrix} 1-2h(\frac{\partial f_{n+1}}{\partial y_{n+1}}) & \frac{2}{3} \\ -\frac{18}{11} & 1-\frac{6}{11}h(\frac{\partial f_{n+2}}{\partial y_{n+2}}) \end{bmatrix} \begin{bmatrix} e_{n+1}^{(i+1)} \\ e_{n+2}^{(i+1)} \end{bmatrix} = \begin{bmatrix} -1 & -\frac{2}{3} \\ \frac{18}{11} & -1 \end{bmatrix} \begin{bmatrix} y_{n+1}^{(i)} \\ y_{n+2}^{(i)} \end{bmatrix} + h \begin{bmatrix} 2 & 0 \\ 0 & \frac{6}{11} \end{bmatrix} \begin{bmatrix} f_{n+1}^{(i)} \\ f_{n+2}^{(i)} \end{bmatrix} + \begin{bmatrix} \varsigma_1 \\ \varsigma_2 \end{bmatrix} \quad (12)$$

In order to solve the above matrix equation, LU decomposition, which is a matrix decomposition that rewrites the matrix in the upper triangular matrix and lower triangular matrix was used (William *et al*., 2007).

Let

$$A = \begin{bmatrix} 1-2h\left(\frac{\partial f_{n+1}}{\partial y_{n+1}}\right) & \frac{2}{3} \\ -\frac{18}{11} & 1-\frac{6}{11}h\left(\frac{\partial f_{n+2}}{\partial y_{n+2}}\right) \end{bmatrix} \quad (13)$$

$$B = \begin{bmatrix} -1 & -\frac{2}{3} \\ \frac{18}{11} & -1 \end{bmatrix} \begin{bmatrix} y_{n+1}^{(i)} \\ y_{n+2}^{(i)} \end{bmatrix} + h \begin{bmatrix} 2 & 0 \\ 0 & \frac{6}{11} \end{bmatrix} \begin{bmatrix} f_{n+1}^{(i)} \\ f_{n+2}^{(i)} \end{bmatrix} + \begin{bmatrix} \varsigma_1 \\ \varsigma_2 \end{bmatrix} \quad (14)$$

$$E = \begin{bmatrix} e_{n+1}^{(i+1)} \\ e_{n+2}^{(i+1)} \end{bmatrix} \quad (15)$$

$$A \cdot E = B \quad (16)$$

Suppose that matrix A is a product of the two matrices,

$$L \cdot U = A \quad (17)$$

where L is the lower triangular and U is the upper triangular. Thus,

$$(L \cdot U) \cdot E = B. \quad (18)$$

Let

$$X = U \cdot E. \quad (19)$$

Hence,

$$L \cdot X = B. \quad (20)$$

and then solving,

$$U \cdot E = X \tag{21}$$

to get the matrix value of E. Finally, the approximation values of y_{n+1} and y_{n+2} can be obtained. Furthermore, the existence of the delay term is the main difference between ODEs and DDEs. Thus, in order to get the numerical solutions for DDEs, the delay term needs to be solved in advance. For a situation where $\alpha \le t_0$, the initial function $\phi(t)$ is used to calculate the delay term, $y(\alpha)$. Otherwise, the delay term is calculated by using Newton Divided Difference interpolation.

Meanwhile, the formula to calculate the delay term (Richard *et al*., 1993) can be written as:

$$P_n(x) = z[x_0] + \sum_{k=1}^{n} z[x_0, x_1, \ldots, x_k](x - x_o)\ldots(x - x_{k-1}) \tag{22}$$

where $z[x_0] = z(x_0)$.

Equation [22] is known as the Newton's interpolatory divided-difference formula. Normally, the delay terms are obtained by interpolation on the values of the chosen support points. In order to obtain these support points, the delay argument should be close to the centre of the points. However, if the delay argument under consideration is outside the points, it will be treated as a special case and it can only be obtained through extrapolation.

PROBLEM TESTED

The followings are some of the tested problems to get the numerical results. All the tested problems are stiff DDEs. The numerical results are obtained by using C language with constant stepsize, H.

Problem 1:

$$y'(t) = -1000y(t) + y\left(t - (\ln(1000 - 1))\right), \quad 0 \le t \le 3,$$

$$y(t) = e^{-t}, \qquad t \le 0.$$

Exact solution is $y(t) = e^{-t}$.

Problem 2:

$$y'(t) = -1000y(t) + 997e^{-3}y(t-1) + \left(1000 - 997e^{-3}\right), \qquad 0 \le t \le 3,$$

$$y(t) = 1 + \exp(-3t), t \le 0.$$

Exact solution is $y(t) = 1 + \exp(-3t)$.

Problem 3:

$$y'(t) = -24y(t) - e^{(-25)}y(t-1), \qquad 0 \le t \le 3,$$

$$y(t) = e^{(-25)t}, t \le 0.$$

Exact solution is $y(t) = e^{-t}$.

NUMERICAL RESULTS AND DISCUSSION

In this section, the numerical results showed the performance of block method in solving the first order stiff DDEs. All the results were obtained using the Microsoft Visual Studio 2010 software on a personal computer with Intel Core2 Duo CPU. The abbreviations used are stated in Table 1.

TABLE 1: List of Abbreviations

Notation	Description
H	Stepsize
1BDF	1-point BDF method
2BBDF	2-point BBDF method
TS	total number of steps
MAXE	Maximum error
TIME	CPU time in seconds

The errors are calculated by using the following formulas:

$$err_t^i = \left| y_{exact}^i - y_{approximate}^i \right| \tag{23}$$

The maximum error, MAXE, is defined as:

$$\text{MAXE} = \max_{1 \le i \le TS} \left(err_t^i \right). \tag{24}$$

The numerical results are given in Table 2 to Table 4. In the results, the notation 9.75159e-004 represents the value of 9.75159 X 10^{-4}. From the numerical results, it is clear that the total number of steps in solving 2-point BBDF is half of the steps needed to solve the 1-point BDF. The execution time for the 2-point BBDF is reduced up to 77% as compared to the 1-point BDF. Meanwhile, the maximum errors for 2-point BBDF method are better than the 1-point BDF for all the tested problems.

TABLE 2: Numerical results for Problem 1[1]

H	METHOD	TS	MAXE	TIME(seconds)
10^{-2}	1BDF	300	9.75159e-004	0.00836748
	2BBDF	150	3.80236e-004	0.00190863
10^{-3}	1BDF	3000	3.22629e-007	0.01103256
	2BBDF	1500	2.61239e-007	0.00451134
10^{-4}	1BDF	30000	1.45149e-008	0.03742193
	2BBDF	15000	1.33568e-008	0.03079985
10^{-5}	1BDF	300000	1.73433e-010	0.29473164
	2BBDF	150000	1.57914e-010	0.27820763

[1]The time of the execution of the 2-point BBDF for Problem 1 is about 77%, 59%, 17%, and 5% faster for the step-sizes of 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}, respectively. The smaller step-size obtained a small difference in the execution time. This is due to the total calculation of LU decomposition that requires matrix in 2BBDF.

TABLE 3: Numerical results for Problem 2[2]

H	METHOD	TS	MAXE	TIME(seconds)
10^{-2}	1BDF	300	8.73499e-003	0.00843524
	2BBDF	150	3.40594e-003	0.00295110
10^{-3}	1BDF	3000	2.88750e-006	0.01167929
	2BBDF	1500	2.33921e-006	0.00715678
10^{-4}	1BDF	30000	1.30592e-007	0.04287257
	2BBDF	15000	1.20176e-007	0.03564649
10^{-5}	1BDF	300000	1.56085e-009	0.32652507
	2BBDF	150000	1.42118e-009	0.30610835

[2]The time of the execution of the 2 point BBDF for Problem 2 is about 65%, 38%, 16%, and 6% faster for the respective step-sizes of 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}

TABLE 4: Numerical results for Problem 3[3]

H	METHOD	TS	MAXE	TIME(seconds)
10^{-2}	1BDF	300	4.63438e-002	0.00857305
	2BBDF	150	4.41121e-002	0.00296303
10^{-3}	1BDF	2000	1.00359e-003	0.01145794
	2BBDF	1500	9.28409e-004	0.00553071
10^{-4}	1BDF	20000	1.10170e-005	0.03680099
	2BBDF	15000	9.97473e-006	0.02976354
10^{-5}	1BDF	200000	1.11197e-007	0.26814825
	2BBDF	150000	1.00464e-007	0.26649446

[3]The time of the execution of the 2-point BBDF for Problem 3 is about 65%, 51%, 19%, and 1% faster for the stepsizes of 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}, respectively

CONCLUSION

In this paper, it was observed that the 2-point BBDF achieved better results in terms of the accuracy of the method as well as the execution time. Therefore, it can be concluded that the 2-point BBDF is suitable for solving stiff DDEs.

ACKNOWLEDGEMENTS

This work was fully supported by the Institute of Mathematical Research, Universiti Putra Malaysia, under the Fundamental Scheme Research Grant (Project code: 01-09-09-681FR).

REFERENCES

Bellen, A., & Zennaro, M. (2003). *Numerical Methods for Delay Differential Equations*. New York: Oxford University Press.

Bocharov, G. A., Marchuk, G. I., & Romanyukha, A. A. (1996). Numerical solution by LMMs of Stiff Delay Differential systems modelling an Immnune Response. *Numer. Math., 73*, 131-148.

Driver, R. D. (1976). *Ordinary and Delay Differential Equations*. New Year: Springer-Verlag.

Ibrahim, Z. B., Othman, K. I., & Suleiman, M. (2007). Implicit r-point Block Backward Differentiation Formula for Solving First Order Stiff ODEs. *Applied Mathematics and Computation, 186*, 558-565.

Ismail, F. (1999). Numerical Solution of Ordinary and Delay Differential Equations by Runge-Kutta Type Methods (Ph.D Thesis dissertation). Universiti Putra Malaysia, Malaysia.

Ismail, F., Al-Khasawneh, R. A., Lwin, A. S., & Suleiman, M. (2002). Numerical Treatment of Delay Differential Equations by Runge-Kutta Method Using Hermite Interpolation. *Matematika, 18*(2), 79-90.

Lambert, J. D. (1993). *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*. Chichester, England: John Wiley and Sons.

Oberle, H. J., & Pesch, H. J. (1981). Numerical Treatment of Delay Differential Equations by Hermite Interpolation. *Numer. Math., 37*, 235-255.

Richard, L., Burden, J., & Douglas, F. (1993). *Numerical Analysis*. Boston: PWS-Kent Pub. Co.

Roth, M. G. (1980). Difference Methods for Stiff Delay Differential Equations (Master Thesis dissertation). University of Illinois, Illinois.

William, H. P., Saul, A. T., William, T. V., & Brian, P. F. (2007). *Numerical Recipes: The Art of Scientific Computing*. New York: Cambridge University Press.