Chiang Mai J. Sci. 2006; 33(2) : 153-159
www.science.cmu.ac.th/journal-science/josci.html
Contributed Paper

1-Point Implicit Code of Adams Moulton Type to Solve First Order Ordinary Differential Equations

Zanariah A. Majid* [a] and Mohamed B. Suleiman [b]

[a] Technical Teacher's Training College, Jalan Yaacob Latif, Bandar Tun Abdul Razak, 56000 Kuala Lumpur, Malaysia.

[b] Department of Mathematics, Faculty of Science, Universiti Putra Malaysia, Selangor Darul Ehsan, Malaysia.

*Author for correspondence; e-mail : zanamajid@hotmail.com

Received : 27 July 2005
Accepted : 18 April 2006.

ABSTRACT

In this paper, a 1 point implicit code in the form of Adams Moulton Method is developed for solving a system of ordinary differential equations (ODEs) with variable step size. The existence multistep method involves the computations of the divided differences and integration coefficients in the code when using the variable step size or variable step size and order. The idea of the developed method is to store all the coefficients involved in the code. Therefore, this strategy can avoid the lengthy computation of the coefficients during the implementation of the code as well as to improve the execution time. Numerical results are given to compare the efficiency of the developed method to the 1PVSO method in Omar [1].

Keywords : 1-point implicit method. Adams Moulton Method, ordinary differential equations.

1. INTRODUCTION

In this paper, we are interested in the numerical solution of non stiff initial value problems (IVPs) for systems of first order ODEs of the form

$$y' = f(x, y),\ y(x_0) = y_0,\ x \in [a, b]. \quad (1)$$

The current multistep method for variable step (VS) or variable step and order (VSVO) technique as described in Omar [1], Gear [2], Lambert [3] and Suleiman [4] involved tedious computations of divided difference and recurrence relation in computing the integration coefficients.

The 1PVSO code in Omar [1] is a 1-point implicit code that involved computations of the divided differences and integration coefficients of $g_{i,t}$. We are going to explain briefly the computations of the integration coefficients involved as follows:-

The interpolation polynomial $P_{k,n}(x)$ which interpolates the values $F_n, F_{n-1}, \ldots, F_{n-k+1}$ of a function F at the points $x_n, x_{n-1}, \ldots, x_{n-k+1}$, in terms of divided differences, has the form,

$$P_{k,n}(x) = F_{[n]} + (x - x_n)F_{[n,n-1]} + \ldots + (x - x_n)(x - x_{n-1})\ldots (x - x_{n-k+2})F_{[n,n-1,\ldots,n-k+1]} \quad (2)$$

where the i – th divided differences are defined by

$$F_{[n,n-1,\ldots,n-i]} = \frac{F_{[n,n-1,\ldots,n-i+1]} - F_{[n-1,n-2,\ldots,n-i]}}{(x_n - x_{n-i})}. \quad (3)$$

Define $P_{n,i}(x) = (x - x_n)(x - x_{n-1})\ldots(x - x_{n-i+1})$ for $i = 1,2, \ldots, k$ and $P_{n,0}(x) = 1$. (4)

Also define $g_{i,t}$, t >0, to be the t-fold integral and the integration coefficients can be obtained by using the following definition:

$$g_{i,t} = \underbrace{\int_{x_n}^{x_{n+1}} \int_{x_n}^{x} \cdots \int_{x_n}^{x}}_{\text{t times}} P_{n,i}(x)\, dxdx\ldots dx \qquad (5)$$

and

$$g_{i,0} = P_{n,i}(x_{n+1}) \text{ for } i = 1,2, ..., k. \qquad (6)$$

Substitute (4) into (5) and then solve using integration by parts yields

$$g_{i,t} = (x_{n+1} - x_{n-i+1})g_{i-1,t} - tg_{i-1,t+1}. \qquad (7)$$

For $i = 0$,

$$g_{0,t} = \underbrace{\int_{x_n}^{x_{n+1}} \int_{x_n}^{x} \cdots \int_{x_n}^{x}}_{\text{t times}} dxdx\ldots dx = \frac{(x_{n+1} - x_n)^t}{t!} \qquad (8)$$

See Omar [1] and Suleiman [4] for detail. Given (7) and (8) can be used to compute the required coefficients in triangular arrays as in Table 1, starting with the first row and from left to right.

Table 1. Integration coefficients.

i \ t	0	1	2	.	.	.	$G_{\max}+1$
0	g_{00}	g_{01}	g_{02}	.	.	.	$g_{0,G_{\max}+1}$
1	g_{10}	g_{11}	g_{12}	.	.	$g_{1,G_{\max}}$	
2	g_{20}						
.	.	.	.				
.	.	.					
.	.						
$k_{\max}+1$	$g_{k_{\max}+1,0}$						

The actual number of coefficients required will depend upon the problem and the order of the method being used in solving the problem. $G_{\max}$ in Table 1 is defined as $G_{\max} = \max_i(k_i + d_i)$ where k_i is the current k_i-step method used and d_i is the order of the differential equation. The maximum value of k_i is denote by $k_{\max}$.

The aim in this paper is to investigate the performance of the propose 1-point implicit code presented in the simple form of the Adams Moulton Method for solving (1) using variable step size. The method is in a simple form but we intend for efficiency and for calculating the method is faster. The idea in the code is to prevent tedious and repetitive computation of the integration coefficients as in Table 1 that can be very costly. Hence, the developed code will store all the coefficients of the method. However because of the step size variations, many formulae need to be stored and may cause a high initial overhead but as the computational work increases the advantage of the method will be evident when the execution time is compared with the code 1PVSO.

2. MATERIALS AND METHOD

2.1 Formulation of the 1-Point Implicit Method

In Figure 1, the computed step has the step size h but in each previous step the length of the interval is rh, qh and ph.

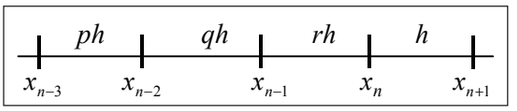

Figure 1. 1-Point Implicit Method.

The point, y_{n+1} can be obtained by integrating (1) therefore,

$$\int_{x_n}^{x_{n+1}} y' \, dx = \int_{x_n}^{x_{n+1}} f(x,y) \, dx$$

or
$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(x,y) \, dx \tag{9}$$

The function $f(x,y)$ in (9) will be approximated using Lagrange interpolating polynomial. The interpolation points involved are $(x_{n-3}, f_{n-3}), \ldots, (x_{n+1}, f_{n+1})$. The value of y_{n+1} can be obtained by integrating (9) over the interval $[x_n, x_{n+1}]$ using MAPLE and the following corrector formulae can be obtained,

$$\begin{aligned} y(x_{n+1}) = y(x_n) + h\Bigg[& \frac{-\left(3+5q+10\left(r+qr+r^2\right)\right)}{60p(p+q)(p+q+r)(1+p+q+r)} f_{n-3} + \frac{\left(3+5(p+q)+10\left(r+pr+qr+r^2\right)\right)}{60pq(q+r)(1+q+r)} f_{n-2} \\ & - \frac{\left(3+5(p+2q)+10\left(pq+q^2+r+pr+2qr+r^2\right)\right)}{60rq(p+q)(1+r)} f_{n-1} \\ & + \frac{\left(3+5(p+2q+3r)+10\left(pq+q^2+2pr+4qr\right)+30\left(pqr+q^2r+r^2+pr^2+r^3+2qr^2\right)\right)}{60r(q+r)(p+q+r)} f_n \\ & + \frac{\left(12+15(p+2q+3r)p+20\left(pq+q^2+2pr+4qr\right)+30\left(pqr+q^2r+2r^2+pr^2+r^3+2qr^2\right)\right)}{60(1+r)(1+q+r)(1+p+q+r)} f_{n+1} \Bigg] \end{aligned} \tag{10}$$

The method is the combination of predictor of order 4 and the corrector of order 5. The predictor formulae were similarly derived where the interpolation points involved are $(x_{n-3}, f_{n-3}), \ldots, (x_n, f_n)$.

2.2 Variable Step Size Strategy

The step size strategy used in the code is a modified version of Shampine [5]. The choices for the next step size will be restricted to half, double or the same as the previous step size and the successful step size will remain constant for at least two blocks before considered it to be doubled. The step size strategy helps to minimize the choices of r, q and p.

Those possible step size ratios of r, q and p have ten combinations. In case of successful step size, the choices during constant step size are (r=1, q=1, p=1), (r=1, q=2, p=2), (r=1, q=1, p=2), (r=1, q=0.5, p=0.5) or (r=1, q=1, p=0.5). When the step size is double the possible ratios are (r=0.5, q=0.5, p=1), (r=0.5, q=0.5, p=0.25) or (r=0.5, q=0.5, p=0.5). In case of step size failure the possible ratios are (r=2, q=2, p=1) or (r=2, q=2, p=2). Substituting the common ratios of r, q and p in (10) will give the corrector formulae for the 1-point implicit method.

2.3 Absolute Stability

For a method to be of practical i mportance it must have a region of absolute stability which will ensure that the method will be able to solve at least the mildly stiff problems. Here we will discussed the absolute stability of the 1-point implicit method (1PI) derived earlier using a linear first order test problem,

$$y' = f = \lambda y. \tag{11}$$

The stability region is investigated when the step size is constant, doubled and halved for the method. The test equation (9) is substituted into the corrector formulae. The stability polynomial of the 1-point implicit method when (r=1, q=1, p=1), (r=0.5, q=0.5, p=1) and (r=2, q=2, p=1) are as follows where $\bar{h} = h\lambda$.

The stability polynomial of the 1-point implicit method when (r=1, q=1, p=1),

$$\left(1 - \frac{25}{720}\bar{h}\right)t^4 - (1 + \frac{646}{720}\bar{h})t^3 + \frac{264}{720}\bar{h}\,t^2 - \frac{106}{720}\bar{h}\,t + \frac{19}{720}\bar{h} = 0.$$

The stability region is shown in Figure 2.

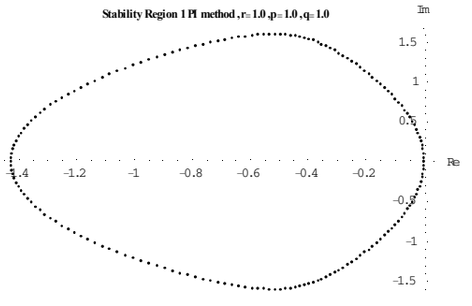

Figure 2. Stability Region for 1PI at $r = 1, p = 1, q = 1$.

The stability polynomial of the 1-point implicit method when (r=0.5, q=0.5, p=1),

$$\left(1 - \frac{329}{1080}\bar{h}\right)t^4 - (1 + \frac{1539}{1080}\bar{h})t^3 + \frac{1216}{1080}\bar{h}\,t^2 - \frac{459}{1080}\bar{h}\,t + \frac{31}{1080}\bar{h} = 0$$

The stability region is shown in Figure 3.

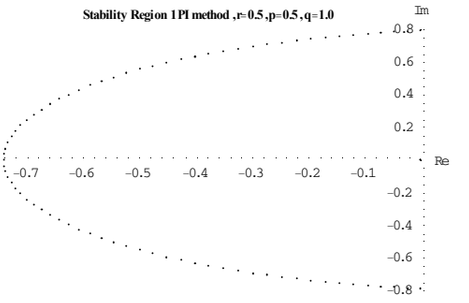

Figure 3. Stability Region for 1PI at $r = 0.5, p = 0.5, q = 1$.

The stability polynomial of the 1-point implicit method when (r=2, q=2, p=1),

$$\left(1 - \frac{4274}{10800}\bar{h}\right)t^4 - (1 + \frac{7371}{10800}\bar{h})t^3 + \frac{1240}{10800}\bar{h}\,t^2 - \frac{621}{10800}\bar{h}\,t + \frac{226}{10800}\bar{h} = 0$$

The stability region is shown in Figure 4.

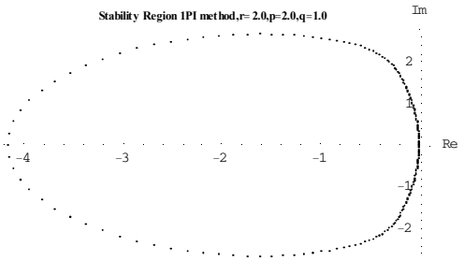

Figure 4. Stability Region for 1PI at $r = 2, p = 2, q = 1$

The closed regions in Figures 2 – 4 are the absolute stability regions for the methods derived in the previous section. It is observed that the stability region when the step size is halved at $r = 2, p = 2, q = 1$ is larger compared to the region of the step size being doubled at $r = 0.5, p = 0.5, q = 1.0$ or kept constant at $r = 1.0, p = 1.0, q = 1.0$. This is expected since the region should get better with smaller step sizes.

3. RESULTS AND DISCUSSION

In order to study the efficiency of the developed code, we present some numerical experiments for the following three problems:

Tested Problems: The 1PI and 1PVSO were applied to the following test problems:

Problem 1:

$$y_1' = y_2,\ y_2' = 2y_2 - y_1,$$
$$y_1(0) = 0,\ y_2(0) = 1,\ [0,50]$$

Exact Solution: $y_1(x) = xe^x,$
$y_2(x) = (1+x)e^x,$

Source: Bronson [6].

Problem 2:

$$y_1' = y_2,\ y_2' = y_1,$$
$$y_1(0) = 1,\ y_2(0) = 1,\ [0,100]$$

Exact Solution: $y_1(x) = e^x$
$y_2(x) = e^x$

Source: Bronson [6]

Problem 3:

$$y_1' = y_2,\ y_2' = -2y_2 - 5y_3 + 3,$$
$$y_3' = y_2 + 2y_3.$$
$$y_1(0) = 0,\ y_2(0) = 0,\ y_3(0) = 1,\ [0,4\pi]$$

Exact Solution:

$$y_1(x) = 2\cos(x) + 6\sin(x) - 6x - 2,$$
$$y_2(x) = -2\sin(x) + 6\cos(x) - 6,$$
$$y_3(x) = 2\sin(x) - 2\cos(x) + 3.$$

Source: Suleiman [7].

The following notations are used in the tables:

TOL	Tolerance
MTD	Method employed
TS	Total steps taken
FS	Total failure step
MAXE	Magnitude of the maximum error of the computed solution
AVERR	Average error
FCN	Total function calls
TIME	The execution time taken in microseconds
1PI	Implementation of the 1-point implicit method code using variable step size
1PVSO	Implementation of the 1-point implicit block method code of variable step size and order using the integration coefficients $g_{i,t}$ in Omar [1].

The errors calculated are defined as

$$(e_i)_t = \left| \frac{(y_i)_t - (y(x_i))_t}{A + B(y(x_i))_t} \right|$$

where $(y)_t$ is the t- th component of the approximate y and for the 1-point implicit method we let t=1. The absolute error test corresponds to A=1, B=0, the mixed test corresponds to A=1, B=1 and finally A=0, B=1 corresponds to the relative error test. For Problems 1 and 2, the relative error test is used. The mixed error test is for Problem 3. The maximum error and average error are defined as follows:-

$$\text{MAXE} = \max_{1 \le i \le SSTEP} \left(\max_{1 \le i \le N} (e_i)_t \right) \text{ and}$$

$$\text{AVE} = \frac{\sum_{i=1}^{SSTEP} \sum_{t=1}^{N} (e_i)_t}{(N)(SSTEP)}$$

where N is the number of equations in the system, $SSTEP$ is the number of successful steps. In the code, we iterate the corrector to convergence. The convergence test employed were

$\left\| y^{(s+1)}_{n+1} - y^{(s)}_{n+1} \right\| < 0.1$ TOL and s is the number of iteration.

The codes were written in C language and executed on DYNIX/ptx operating system. The tables below show the numerical results for the three problems when solved using 1PI and 1PVSO.

Table 2. Comparison between the 1PI and 1PVSO for Solving Problem 1.

TOL	MTD	TS	FS	MAXE	AVERR	TIME
10^{-2}	1PI	159	0	1.37e-3	6.34e-4	8755
	1PVSO	90	0	2.26e-3	1.08e-3	10276
10^{-4}	1PI	373	0	3.80e-5	1.88e-5	17144
	1PVSO	178	0	1.47e-4	7.21e-5	18007
10^{-6}	1PI	903	0	1.32e-6	2.81e-7	41537
	1PVSO	737	0	1.32e-6	7.02e-7	68202
10^{-8}	1PI	2223	0	1.32e-7	3.40e-9	102496
	1PVSO	1609	0	2.90e-8	1.56e-8	148798
10^{-10}	1PI	5523	0	1.32e-8	3.98e-11	254439
	1PVSO	3997	0	3.13e-10	1.71e-10	370052

Table 3. Comparison between the 1PI and 1PVSO for Solving Problem 2.

TOL	MTD	TS	FS	MAXE	AVERR	TIME
10^{-2}	1PI	216	0	5.59e-3	2.51e-3	11199
	1PVSO	121	0	6.54e-2	2.85e-2	14231
10^{-4}	1PI	515	0	3.09e-4	1.46e-4	22936
	1PVSO	271	0	6.21e-4	2.94e-4	27667
10^{-6}	1PI	125	0	5.13e-6	2.50e-6	55803
	1PVSO	1584	0	1.95e-5	9.43e-6	59491
10^{-8}	1PI	3084	0	6.29e-8	3.11e-8	137704
	1PVSO	2253	0	2.88e-7	1.44e-7	210196
10^{-10}	1PI	7666	0	7.09e-10	3.46e-10	342924
	1PVSO	5978	0	2.37e-9	1.21e-9	557020

Table 4. Comparison between the 1PI and 1PVSO for Solving Problem 3.

TOL	MTD	TS	FS	MAXE	AVERR	TIME
10^{-2}	1PI	55	0	8.71e-3	1.41e-3	4698
	1PVSO	33	0	8.49e-2	3.63e-3	6882
10^{-4}	1PI	111	0	9.09e-5	1.86e-5	7747
	1PVSO	53	0	3.01e-3	1.87e-4	7370
10^{-6}	1PI	246	0	9.53e-7	2.20e-7	17301
	1PVSO	133	0	1.56e-5	1.35e-6	18379
10^{-8}	1PI	575	0	9.73e-9	2.43e-9	40692
	1PVSO	329	0	2.63e-7	1.85e-8	44939
10^{-10}	1PI	1393	0	9.98e-11	2.57e-11	98960
	1PVSO	803	0	2.52e-9	1.73e-10	109807

In most cases the maximum error of 1PI is better than 1PVSO. The results indicated that the total number of steps taken by 1PVSO is much lesser than 1PI in all problems. This is expected since 1PVSO is a variable step and order method with various order of k, and therefore the method will generally have larger step size and hence lesser total steps.

However, the code 1PI is better in terms of execution times compared to 1PVSO even though the total steps taken in 1PI is double than the steps taken by 1PVSO in the tested problems. Therefore, we can conclude that the cost of computing the divided differences and integration coefficients in the 1PVSO code is the major disadvantage when permitting random variations in the step size, therefore the computational cost increases when the codes are implemented in variable step size and order.

Table 5. The ratios execution times of the 1PVSO Method to the 1PI Method for Solving Problem 1 to 3.

TOL	PROB 1	PROB 2	PROB 3
10^{-2}	1.17	1.27	1.46
10^{-4}	1.05	1.21	0.95
10^{-6}	1.64	1.07	1.06
10^{-8}	1.45	1.53	1.10
10^{-10}	1.45	1.62	1.11

In Table 5, the ratios of the execution times are greater than one show that the 1PI code is efficient than 1PVSO. Therefore, these demonstrate the advantage of the 1PI code in the form of standard multistep method because the cost per step is cheaper.

We have shown the efficiency of the 1-point implicit code presented as in the form of Adams Moulton Method with variable step size is suitable for solving ODEs.

REFERENCES

[1] Omar, Z. *Developing Parallel Block Methods For Solving Higher Order ODEs Directly*, Ph.D. Thesis, University Putra Malaysia, Malaysia, 1999.

[2] Gear, C.W. *Numerical Initial Value Problems in Ordinary Differential Equations*. New Jersey: Prentice Hall, Inc., 1971.

[3] Lambert, J.D. *Numerical Methods For Ordinary Differential Systems. The Initial Value Problem*. New York: John Wiley & Sons, Inc., 1993.

[4] Suleiman, M.B. Generalised Multistep Adams and Backward Differentiation Methods for the Solution of Stiff and Non-Stiff Ordinary Differential Equations. Ph.D. Thesis. University of Manchester, 1979.

[5] Shampine, L.F. and Gordon, M.K. *Computer Solution of Ordinary Differential Equations*: The Initial Value Problem, W. H. Freeman and company, San Francisco, 1975.

[6] Bronson, R. *Modern Introductory Differential Equation: Schaum's Outline Series*. USA: McGraw-Hill Book Company, 1973.

[7] Suleiman, M.B. Solving Higher Order ODEs Directly by the Direct Integration Method. *Applied Mathematics and Computation* 33, 1989: 197-219.