Applied Numerical Mathematics 2 (1986) 109–118
North-Holland

A C^1 INTERPOLANT FOR CODES BASED ON BACKWARD DIFFERENTIATION FORMULAE

M. BERZINS
Department of Computer Studies, University of Leeds, Leeds LS2 9JT, U.K.

This note is concerned with the provision of an interpolant for o.d.e. initial value codes based upon backward differentiation formulae (b.d.f.) in which both the solution and its first time derivative are continuous over the range of integration—a C^1 interpolant. The construction and implementation of the interpolant is described and the continuity achieved in practice is illustrated by two examples.

1. Introduction

At present many of the most popular library programs for the solution of the stiff initial value problem

$$y'(t)=f(t, y(t)), \quad y(a) \text{ given}, \qquad a \leqslant t \leqslant b$$

are based on the b.d.f. integration method of Gear [7]. In this paper we shall consider the fixed step form of the b.d.f. methods with interpolatory step changing as implemented in codes such as LSODE and LSODI [8]. Codes for initial value problems generally produce solution values, y_j, in a stepwise fashion at a set of points

$$a=t_0<t_1<t_2< \cdots <t_m=b$$

where y_j denotes the computed approximation to the solution $y(t_j)$. Although, in general, each individual step-size is chosen so as to satisfy a local error criterion we shall assume for the moment that the points t_j are equally spaced in time. In the case when the method of order k is used to integrate from t_{j-1} to t_j the assumption used by the b.d.f. method is that the solution between the points t_{j-k} and t_j can be approximated by a polynomial of degree k. This polynomial interpolates these $k+1$ solution values as well as the derivative y_j and so forms the natural interpolant for the method. The values of the solution and its derivatives at a point t where $t_{j-1} \leqslant t \leqslant t_j$ are found by interpolation using this polynomial. It should be noted that the polynomial calculated at the end of the step to t_j passes through the previous k solution values and y_j' but does not pass through y_{j-1}'. This interpolant is continuous over the interval of integration $[a,b]$ but has a discontinuity in the first derivative at each of the points t_i, $i=1, 2,\ldots, m-1$. In the case when the mesh is not evenly spaced the earlier data is that given by the interpolation polynomials from previous time steps. The polynomial is stored in the Nordsieck vector form given by

$$\left[y_j,\ h_{j+1}y_j',\ \frac{h_{j+1}^2}{2!}y_j'',\ldots,\ \frac{h_{j+1}^k}{k!}y_j^{(k)}\right]^{\mathrm{T}} \tag{1}$$

where each of the derivatives $y_j^{(i)}$ is an approximation to the ith derivative of the solution at t_j.

In many applications the solution to the initial value problem is continuous and so is the function $f(t, y)$—thus implying continuity of the first derivative. In this case it is possible to generate a continuous interpolant of the derivative by using the continuous interpolant in evaluating the function $f(t, y)$. However, since the b.d.f. code does not solve the differential equation exactly, the derivative values thus generated at the mesh points differ from the mesh point derivative values used by the code by an amount which is proportional to the iteration error, see [12]. This inconsistency makes it desirable to interpolate the derivative directly from the approximating polynomial.

The situation is more difficult when b.d.f. codes are applied to implicit differential-algebraic equations e.g. LSODI, see [8]. In this case it will, in general, not be possible to compute y' directly from the differential equation using y if there are algebraic equations present.

The problem of providing a sufficiently continuous interpolant also occurs in the Adams method implemented in the DE/STEP code of Shampine and Gordon [10]. In this case, however, it is the interpolant of the computed solution which is discontinuous at the mesh points and the first derivative which is continuous. Recent work by Shampine and Watts [13], Watts [14] and by Watts and Shampine [15] has described how the interpolation routine inside DE/STEP can be modified so as to produce a C^1 interpolant. The aim of this note is to provide a similar interpolant for the fixed step b.d.f. method with interpolatory step changing. Although the ideas used here could probably be extended to the variable step b.d.f. formulas implemented in codes such as EPISODE, Byrne and Hindmarsh (1975) the motivation behind this work is to provide a satisfactory interpolant for the new b.d.f. integrator (based on LSODI), in the SPRINT code of Berzins and Furzeland (1984) and the related code that is to be released in the N.A.G. Library. Although these codes solve differential algebraic equations rather than the explicit o.d.e. problem described above, the interpolant that we describe can be applied without modification to them.

The practical importance of this interpolant lies in applications such as finding the root of a function.

$$g(y, y', t) = 0$$

by using interpolation to calculate values of y and y' or in graphical output, see [13]. In both cases discontinuous values of the derivative may produce unexpected or incorrect results.

2. Outline of integration algorithm

For ease of notation we shall consider only a single equation of the form of (1) and only describe briefly those parts of the integration algorithm that are relevant to this discussion. A complete description of the b.d.f. algorithm is given in the report of Dew and West [5]. We shall assume that the solution has been computed as far as the point t_j using a the fixed step form of the b.d.f. with interpolatory step changing, and that the last step was taken with a step size h_j using a method of order k. Suppose that the step-size for the next step is predicted to be h_{j+1} and the order for the next step is q where q is $k-1$, k or $k+1$ depending on the order selected by the algorithm. Once the order is selected, the algorithm generates the Nordsieck vector given by

$$\left[y_j, h_{j+1}y_j', \frac{h_{j+1}^2}{2!}y_j'', \dots, \frac{h_{j+1}^q}{q!}y_j^{(q)}\right]^{\mathrm{T}}. \tag{2}$$

This vector is used in truncated Taylor's series expansions to predict the solution and its first q derivatives at t_{j+1} by the calculation

$$\bar{y}_{j+1}^{(l)}=\sum_{i=l}^{q}\frac{y_j^{(i)}h_{j+1}^{i-l}}{(i-l)!},\quad l=0,\dots,q$$

where $\bar{y}_{j+1}$ is the predicted solution at t_{j+1} and its derivatives are defined analogously. The predicted form of the vector is then denoted by

$$\left[\bar{y}_{j+1},\ h_{j+1}\bar{y}'_{j+1},\ \frac{h_{j+1}^2}{2!}\bar{y}''_{j+1},\dots,\ \frac{h_{j+1}^q}{q!}\bar{y}_{j+1}^{(q)}\right]^{\mathrm{T}}. \tag{3}$$

As the qth derivative is approximated by a constant, i.e. $\bar{y}_{j+1}^{(q)}=y_j^{(q)}$, we can reverse the Taylor's series mapping on the vector given by (3) to recover all the components of (2), i.e.

$$y_j^{(l)}=\sum_{i=l}^{q}\frac{\bar{y}_{j+1}^{(i)}\left(-h_{j+1}\right)^{i-l}}{(i-l)!},\quad l=0,\dots,q. \tag{4}$$

The predicted values $\bar{y}_{j+1}$ and $\bar{y}'_{j+1}$ are used in a Newton type nonlinear equations solver to solve the single equation (more generally a system of equations)

$$y'_{j+1}=f\left(t_{j+1},\ y_{j+1}\right)$$

where y'_{j+1} is approximated by the backward differentiation replacement

$$y'_{j+1}=\left(y_{j+1}-\sum_{i=1}^{q}y_{j+1-i}\alpha_i\right)/h_{j+1}e_0,$$

the constant e_0 is defined below and the coefficients α_i are defined by Gear [7]. We shall assume that the iteration is successful and that the new solution and derivative values are accepted. (Should be step fail the variable step/variable order algorithm will at worst only change the values of q or h_{j+1} in (3) and retry the step.) A discussion of how this system of equations should be solved is included in [12].

Denote the sum of the corrections to the predicted value $\bar{y}'_{j+1}$ by $\bar{\alpha}$, i.e.

$$y'_{j+1}=\bar{y}'_{j+1}+\bar{\alpha}.$$

The corrections to the other components of the Nordsieck vector are also defined in terms of $\bar{\alpha}$ (see [5]), i.e.

$$\frac{h_{j+1}^i y_{j+1}^{(i)}}{i!}=\frac{h_{j+1}^i\bar{y}_{j+1}^{(i)}}{i!}+e_i h_{j+1}\bar{\alpha},\quad i=0,\dots,q \tag{5}$$

where

$$e_i=c_i/c_i$$

and c_i is the coefficient of x^i in the polynomial $p(x)$ which is defined by

$$p(x)=(x+1)(x+2)\cdots(x+q). \tag{6}$$

It follows directly from this definition that

$$\sum_{i=0}^{q}e_i(-1)^i=p(-1)/c_1=0 \tag{7}$$

as the polynomial $p(x)$ is zero at $x = -1$. The form of $p(x)$ ensures that interpolation using the new Nordsieck vector is consistent with the old solution values at $t = t_{j+1} - ih_{j+1}$, $i = 1, \dots, q$ and also with the new solution and derivative values at t_{j+1}. For example, let w_j and w_j' be the values of the solution and its first derivative at time t_j obtained by interpolation, then

$$w_j = \sum_{i=0}^{q} \frac{y_{j+1}^{(i)}(-h_{j+1})^i}{i!},$$

Substitution using (5) gives

$$w_j = \sum_{i=0}^{q} \frac{\bar{y}_{j+1}^{(i)}(-h_{j+1})^i}{i!} + h_{j+1}\bar{\alpha} \sum_{i=0}^{q} e_i(-1)^i.$$

From (4) and (7) it follows that

$$w_j = y_j. \tag{8}$$

Using (4) with $l = 1$ and with $\bar{y}_{j+1}$ replaced by y_{j+1} gives

$$w_j' = \sum_{i=1}^{q} \frac{y_{j+1}^{(i)}(-h_{j+1})^{i-1}}{(i-1)!}.$$

Substitution for $y_{j+1}^{(i)}$ using (5) gives

$$w_j' = \sum_{i=1}^{q} \left\{ \frac{\bar{y}_{j+1}^{(i)}(-h_{j+1})^{i-1}}{(i-1)!} + \bar{\alpha} e_i i(-1)^{i-1} \right\},$$

and so, using (4) with $l = 1$, we get

$$w_j' = y_j' + \bar{\alpha} \sum_{i=1}^{q} e_i i(-1)^{i-1}.$$

The value of the derivative of $p(x)$ at $x = -1$ allows this to be written as

$$w_j' = y_j' + \bar{\alpha}(q-1)!/c_1$$

where c_1 may be calculated from (6) to be

$$c_1 = q!\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{q}\right).$$

This shows the magnitude of the discontinuity in the first derivative at the previous time level when the normal interpolation procedure is used.

3. A modified interpolation procedure

In this section we show now the interpolation procedure based on the Nordsieck vector can be modified so as to produce an interpolant which is continuous and has a continuous derivative at the time integration points t_j as well as elsewhere. The discontinuity in the first derivative may be thought of as occurring because the polynomial $p(x)$ has a non-zero derivative at the point

$x = -1$. We shall now construct a modified polynomial that corrects this deficiency. Consider a polynomial $r(x)$ which satisfies

$$r(-1) = -1, \quad r(0) = r'(0) = 0.$$

The simplest (and lowest-order) form for $r(x)$ is

$$r(x) = -x^2. \tag{9}$$

The modified interpolation procedure is based on incrementing the Nordsieck vector using the polynomial $(1 + r(x))p(x)$ rather than just $p(x)$ as in (5). Define the coefficients d_i by

$$d_0 = 0, \quad d_1 = 0, \qquad d_i = -e_{i-2}, \quad i = 2,\dots, q+2. \tag{10}$$

In other words d_i is the coefficient of x^i in the polynomial

$$-x^2 p(x)/c_1.$$

Now define an *augmented Nordsieck vector* that represents the new interpolating polynomial $z(x)$ for the interval $[t_{j-1}, t_j]$ in Nordsieck form by

$$\left.\begin{aligned} \frac{h_{j+1}^i z_{j+1}^{(i)}}{i!} &= \frac{h_{j+1}^i \bar{y}_{j+1}^{(i)}}{i!} + e_i h_{j+1}\bar{\alpha} + d_i h_{j+1}\bar{\alpha}, && i = 0,\dots, q, \\ \text{and} \quad \frac{h_{j+1}^i z_{j+1}^{(i)}}{i!} &= d_i h_{j+1}\bar{\alpha}, && i = q+1, q+2. \end{aligned}\right\} \tag{11}$$

We can show that interpolation on this augmented vector provides values of the solution and its time derivative that are consistent with the actual solution values at t_j and t_{j+1}. Let v_j and v_j' be the interpolated solution and first derivative values at time t_j that are obtained by using the *augmented Nordsieck vector* defined by (11) at time t_{j+1}. Using (10) and (11) it follows that

$$v_j = \sum_{i=0}^{q} \frac{\bar{y}_{j+1}^{(i)}(-h_{j+1})^i}{i!} + h_{j+1}\bar{\alpha} \sum_{i=0}^{q} e_i\big((-1)^i - (-1)^{i+2}\big),$$

and so, using (4) with $l = 0$,

$$v_j = y_j.$$

In the same way

$$v_j' = y_j' + \bar{\alpha} \sum_{i=1}^{q} e_i\big(i(-1)^{i-1} - (i+2)(-1)^{i+1}\big) + 2\bar{\alpha} e_0,$$

and so, by cancelling and re-arranging,

$$v_j' = y_j' + 2\bar{\alpha} \sum_{i=0}^{q} e_i(-1)^i,$$

and using (7) gives

$$v_j' = y_j'.$$

It is equally straight-forward to show that the correct values of the solution and its time derivative are generated at t_{j+1}.

4. Implementation details

In practice there is no need to directly form the augmented Nordsieck vector as the correction to the existing vector can be easily calculated in the interpolation routine. The extra information needed is the constant $\bar{\alpha}$ (which is a vector when systems of initial value problems are solved) and the coefficients e_i. The local error estimate produced by the code at time t_{j+1}, E_l has the form

$$E_l = h_{j+1}\bar{\alpha}\gamma \quad \text{where } \gamma = e_0/(q+1), \tag{12}$$

and as the constants γ and e_i are available via COMMON blocks in most codes of interest, the only other information needed by the interpolation routine, via its parameter list, is the local error estimate E_l (which, again, is normally a vector).

There are three minor difficulties to overcome in implementing interpolation using the augmented form of the Nordsieck vector inside the SPRINT b.d.f. code. The first of these is that in codes such as LSODE the Nordsieck vector saved at the end of a step is scaled by the proposed step-size for the next step. The form of the augmented vector used in interpolation thus has the form

$$\left.\begin{aligned} \frac{h^i_{j+2} z^{(i)}_{j+1}}{i!} &= \frac{h^i_{j+2} y^{(i)}_{j+1}}{i!} + (e_i + d_i) h_{j+1}\bar{\alpha}\frac{h^i_{j+2}}{h^i_{j+1}}, && i = 0,\ldots,q,\\ \frac{h^i_{j+2} z^{(i)}_{j+2}}{i!} &= d_i h_{j+1}\bar{\alpha}\frac{h^i_{j+2}}{h^i_{j+1}}, && i = q+1,\, q+2, \end{aligned}\right\} \tag{13}$$

where h_{j+2} is the proposed step-size for the next step. This may increase the possible rounding errors if h_{j+2} is larger than h_{j+1}.

The second difficulty occurs at time t_j if the order is to be increased for the proposed step to t_{j+1}. An extra column is computed for the Nordsieck vector and the order is incremented by one. This results in the interpolant based on this new vector being discontinuous at time t_{j-1}, see [3] and [4]. The obvious solution is to use only those parts of the vector that correspond to the previous order in the interpolation routine.

The third difficulty occurs on a proposed order decrease on the next step to t_{j+1}. In this case only those parts of the Nordsieck vector which correspond to the new order are re-scaled by the new step size. As the interpolation routine assumes that all components of the vector at the old order are so scaled, it is necessary to extend the scaling loop to scale what are otherwise superfluous components for the next step.

The SPRINT b.d.f. module that implements these changes differs from LSODE in a number of other respects. In particular the stepsize/order strategy follows the ideas of Shampine [11] and Petzold [9] in that a potential order decrease to a formula with better stability properties is considered on every step. One result of this approach is greater efficiency than is usual for some b.d.f. codes on the B5 test problem of Enright et al. [6] used as an example below.

5. Evaluating the new interpolant

There are two features of the new interpolant that must be investigated. The first of these is the difference between the old and new interpolants while the second is the effect of rounding error on the continuity achieved in practice.

Suppose that the augmented vector is being used to interpolate the solution at a point t in the interval $[t_j, t_{j+1}]$ where

$$t = t_{j+1} - h \quad \text{and} \quad |h| \leqslant |h_{j+1}|,$$

then the difference between the two interpolants can be seen by comparing (5) and (11) to be

$$\delta = \sum_{i=0}^{q+2} d_i h_{j+1} \bar{\alpha} \left\{ \frac{-h}{h_{j+1}} \right\}^i$$

which can be written as

$$\delta = -h_{j+1} \bar{\alpha} x^2 p(-x)/c_1$$

where $x = h/h_{j+1}$ and $0 \leqslant x \leqslant 1$. Now, as the step has been accepted, it follows from (12) that

$$|h_{j+1}\bar{\alpha}| \leqslant \tau/\gamma$$

where τ is the local error tolerance and γ is defined above. Consequently,

$$|\delta| \leqslant (\tau/\gamma c_1)m \quad \text{where} \quad m = \max|-x^2 p(-x)|, \quad 0 \leqslant x \leqslant 1.$$

The polynomial $p(x)$ depends on the order of the backward differentiation formula being used; in most applications the order is less than 6. From(12) and (6) we see that

$$\gamma c_1 = q!/(q+1).$$

It follows that

$$\delta \leqslant \tau M \quad \text{where} \quad M = m(q+1)/q!.$$

The values of the two constants m and M were computed numerically and are tabulated below for orders 1 to 5 in Table 1. The table shows that the maximum difference between the solution values produced by the two interpolants is bounded above by the local error.

The continuity achieved in practice with the new scheme is subject to the rounding error that occurs in summing Nordsieck vector components which may themselves contain rounding errors. Suppose that the rounding error in component $h^i z^{(i)}/i!$ of the augmented Nordsieck vector is denoted by r_i. The rounding error in the interpolated solution at the end of the previous time-step, $v(t_{j-1})$ is given by r_v where

$$r_v = \sum_{i=0}^{q+2} (-1)^i r_i$$

The component $h^i z^{(i)}/i!$ is multiplied by i and divided by h when the derivative is interpolated. The rounding errors already present are similarly multiplied giving the total rounding error as

$$r_{v'} = \frac{1}{h} \sum_{i=0}^{q+2} (-1)^i i r_i,$$

and so the derivative will normally exhibit a larger discontinuity, particular if h is 'small'.

Table 1

Order	1	2	3	4	5
m	0.14815	0.20176	0.48557	1.66841	7.43606
M	0.29630	0.30264	0.32371	0.34758	0.37180

In order to illustrate the continuity of the new interpolant the test problem B5 of the test set of Enright et al. [6] was used. The problem is defined by

$$y_1' = -10y_1 + 100y_2,\quad y_2' = -100y_1 - 10y_2,\quad y_3' = -4y_3,$$
$$y_4' = -y_4,\quad y_5' = -0.5y_5,\quad y_6' = -0.1y_6.$$

The initial conditions are given by

$$y_i(0) = 1,\quad i = 1,\ldots,6.$$

The problem was integrated from $t = 0$ to $t = 20$ using the b.d.f. integrator of the SPRINT package (see [2]) with an absolute error test. At each time level, t_j reached by the integrator the errors incurred in interpolating back to the previous time level t_{j-1} were measured and used in the following overall error measures.

$$\mathrm{E0} = \max_j \| v(t_{j-1}) - y(t_{j-1}) \|_\infty,$$
$$\mathrm{E1} = \max_j \| v'(t_{j-1}) - y'(t_{j-1}) \|_\infty$$

and

$$\mathrm{E2} = \max_j \| w'(t_{j-1}) - y'(t_{j-1}) \|_\infty,\quad j = 2,\ldots,m$$

where $v(t_{j-1})$ and $v'(t_{j-1})$ are the solution and its time derivative vectors that were computed using the *augmented Nordsieck vector* at time t_j: $y(t_{j-1})$ and $y'(t_{j-1})$ are the solution and derivative values which were directly calculated by the b.d.f. code in the integration step to time t_{j-1} and $w'(t_{j-1})$ is the vector of derivative values computed using the original Nordsieck vector (with the correct order) at time t_j. The computer used was an Amdahl 5850 and the code was a double precision version. Table 2 shows that the maximum continuity error in the solution is of the order of the unit round-off error (2.2 E − 16) while the continuity error in the derivative is larger.

Similar results were obtained on a number of other test problems but rather less satisfactory numerical results were obtained on the Van der Pol Equation which is defined by

$$y_1' = y_2,\qquad y_2' = 100(1 - y_1^2)y_2 - y_1$$

where

$$y_1(0) = 2.0,\quad y_2(0) = 0.0\quad \text{and } t \in (0, 165).$$

Table 2

TOL	STEPS	FCN	DECOMP	E0	E1	h	E2
0.1 E-2	143	305	21	6.8 E−16	8.1 E−14	4.8 E−3	8.89
0.1 E-3	233	418	23	4.8 E−16	5.6 E−14	7.2 E−4	1.20
0.1 E-4	363	611	32	5.9 E−16	6.0 E−14	5.3 E−4	1.19
0.1 E-5	545	849	39	7.9 E−16	9.9 E−14	1.4 E−3	1.4 E−2
0.1 E-6	911	1342	55	9.0 E−16	9.2 E−14	9.4 E−4	3.0 E−3
0.1 E-7	1279	1896	77	4.4 E−16	7.1 E−14	4.7 E−4	6.5 E−4
0.1 E-8	1912	2733	105	6.2 E−16	6.5 E−14	4.9 E−4	5.9 E−5

TOL is the local error tolerance, STEPS is the number of integration time-steps taken, FCN is the number of O.D.E. function calls, DECOMP is the number of L-U decompositions of the Jacobian matrix and h is the step-size taken at the given value of E1.

Table 3

TOL	STEPS	FCN	DECOMP	E0	E1	h	E2
0.1 E−2	278	734	92	9.2 E−13	5.4 E−12	9.9 E−4	2.2 E−1
0.1 E−3	410	882	87	9.1 E−13	8.2 E−12	5.9 E−4	7.4 E−2
0.1 E−4	562	1071	94	8.8 E−13	4.3 E−11	3.4 E−4	2.3 E−2
0.1 E−5	773	1299	100	9.9 E−13	3.6 E−11	2.1 E−4	1.1 E−2
0.1 E−6	1131	1867	142	1.0 E−12	3.2 E−11	3.2 E−4	3.8 E−4
0.1 E−7	1518	2264	149	1.0 E−12	2.9 E−11	1.6 E−4	8.0 E−5
0.1 E−8	2086	2943	180	9.2 E−13	3.6 E−11	1.1 E−4	6.4 E−6

Table 4

TOL	E0	E1	h	E1.1
0.1 E−2	6.7 E−16	4.6 E−12	7.5 E−4	3.5 E−15
0.1 E−3	6.6 E−16	1.1 E−11	1.4 E−4	3.5 E−15
0.1 E−4	7.0 E−16	3.0 E−11	8.1 E−5	3.4 E−15
0.1 E−5	7.0 E−16	1.4 E−11	2.0 E−4	3.5 E−15
0.1 E−6	6.6 E−16	2.5 E−11	1.2 E−4	3.5 E−15
0.1 E−7	7.7 E−16	3.9 E−11	8.2 E−5	3.5 E−15
0.1 E−8	7.3 E−16	4.1 E−11	8.4 E−5	3.5 E−15

In this case the size of the solution components varies widely and so the ith component in the L_∞ norms used in E0, E1 and E2 is weighted, when assessing the interpolated values at t_{j-1}, by

$$\left[\max\left(1, |y_i(t_{j-1})|\right)\right]^{-1} \quad \text{and} \quad \left[\max\left(1, |y_i(t_{j-1})|\right)\right]^{-1} \quad \text{respectively.}$$

The continuity measures E0, E1 and E2 are shown in Table 3.

The relatively poor continuity as shown by this table is partly caused by the scaling performed by the code as described in Section 4, (13). This was verified by using a version of the integrator in which the Nordsieck vector was not rescaled by the new step-size until the beginning of the next step. Table 4 shows the new values of E0 and E1.

The continuity in the solution is much improved but the derivative continuity is still poor. Column E1.1 of the table provides the value of E1 calculated using hy' and hv' instead of y' and v'. A comparison of E1 and E1.1. shows that most of the increase in the discontinuity from E0 to E1 is due to division by h.

6. Adams methods

The same interpolation procedure can also be applied to the Adams Method in codes such as LSODE. This method and its implementation is very different to that in the DE/STEP code, see Shampine (1978). The Adams methods used in LSODE make use of the Nordsieck vector to store information from previous timesteps and differ from the b.d.f. algorithms only in the values of the constants e_i, where $i = 1,\dots,q$, and γ. The corresponding polynomial for the Adams methods to that in (5) for the backward differentiation formulas is defined by

$$p(-1) = p'(-j) = 0, \quad j = 1,\dots,q.$$

unless the order q is one in which case

$$p(x) = (1 + x).$$

Consequently the Adams formulae implemented in LSODE already have a C^1 interpolant if the order is two or greater. In order to cater for the order 1 case and to provide a consistent interpolant the scheme above can be applied. The only difference lies in the values of the constants m, M, c_1 and γ that appear in Section 5 above. The degree of numerical continuity obtained is almost identical to that obtained with the backward differentiation methods.

Acknowledgement

Thanks are due to Ian Gladwell for motivating this work and to both Ian and to Peter Dew and to the referees for their helpful comments on earlier versions of this note.

References

[1] M. Berzins, P.M. Dew and R.M. Furzeland, Software for time-dependent problems, Report 180, Department of Computer Studies, University of Leeds, 1983; a shortened version appeared in B. Engquist and T. Smedsaas, Eds., *PDE Software, Modules, Interfaces and System* (North-Holland, Amsterdam, 1984).

[2] M. Berzins and R.M. Furzeland, A users manual for SPRINT. Part 1. Algebraic and ordinary differential equations, Report 199, Department of Computer Studies, University of Leeds, 1984.

[3] R.L. Brown, Recursive calculation of corrector coefficients, *ACM SIGNUM Newsletter 8* (1973) 12–13.

[4] G.D. Byrne and A.C. Hindmarsh, A polyalgorithm for the numerical solution of ordinary differential equations, *A.C.M. Trans Math. Software 1* (1975) 71–96.

[5] P.M. Dew and M. West, A package for integrating stiff systems of differential equations based on Gear's Method: Part 1. Department of Computer Studies, Report 111, The University of Leeds, 1978.

[6] W.H. Enright, T.E. Hull and B. Lindberg, Comparing numerical methods for stiff systems of O.D.E.s, *BIT 15* (1975) 10–48.

[7] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations* (Prentice Hall, Englewood Cliffs, NJ, 1971).

[8] A.C. Hindmarsh, ODE Solvers for use with the Method of Lines, in: R. Vichnevetsky and R.S. Stepleman, Eds. *Advances in Computer Methods for Partial Differential Equations IV* (IMACS, 1981).

[9] L. Petzold, A description of dassl, Report SAND82-8367, Applied Math. Division, Sandia National Laboratories, Livermore, CA, 1982.

[10] L.F. Shampine and M.K. Gordon, *Computer Solution of O.D.E.s* (Freeman, San Francisco, CA, 1975).

[11] L.F. Shampine, Stability properties of adams methods codes, *ACM Trans. Math. Software 4* (4) (1978).

[12] L.F. Shampine, Implementation of Implicit O.D.E. Formulas for the solution of O.D.E.s, *SIAM J. Sci. Statist. Comput. 1* (1) (1980).

[13] L.F. Shampine and H.A. Watts, A smoother interpolant of DE/STEP; INTRP and DEABM, Sandia Report SAND83-1226, SANDIA National Labs., Albuquerque, NM, 1983.

[14] H.A. Watts, A smoother interpolant of DE/STEP; INSTRP and DEABM (11), Sandia Report SAND64-293, SANDIA National Labs, Albuquerque, NM, 1984.

[15] H.A. Watts and L.F. Shampine, Smoother interpolants for Adams Codes, *SIAM J. Sci. Statist. Comput.*, to appear.