Efficient block predictor-corrector methods with a small number of corrections

Kevin Burrage

Department of Mathematics, Centre for IAMPC, University of Queensland, Brisbane, Australia

Received 6 January 1992
Revised 24 February 1992

Abstract


Recently, various classes of predictor-corrector methods have been proposed as being suitable for solving nonstiff ordinary differential equations in a parallel environment. This paper shows that methods based on a low-order predictor and a Runge-Kutta corrector are not efficient and that if predictor-corrector methods are to be used efficiently for solving nonstiff problems in parallel, then high-order predictors are required. Examples of methods with high-order predictors are given and their efficiency properties are studied in terms of stability and local error theory.

Keywords: Block methods; prediction-correction; efficiency; parallel.

1. Introduction

In an attempt to exploit the parallelism which is available through many different types of hardware and architectures (transputers, hypercubes, connection machines, etc.) there is now a considerable body of research devoted to developing parallel numerical methods for the numerical solution of ordinary differential equations.

Roughly, two main areas can be identified. One is based on the concept of waveform relaxation and iteration in the function space (of which Picard iteration is perhaps the best-known example, see [10], for example) which allows blocks of components to be solved independently of one another. This approach allows the possibility of massive parallelism across the problem but there are substantial difficulties in, for example, knowing how to reorder and regroup the subsystems automatically (see [6] for example). However, for some classes of problems (such as in circuit modelling) where the physical components of the
problem suggest natural groupings based on tight couplings, some success has been achieved (see [14]).

The second approach, which is the one adopted in this paper, is based on the concept of block predictor-corrector methods and allows the possibility of moderate parallelism across the method rather than across the system. Predictor-corrector methods can be used in the solution of stiff or nonstiff problems depending on whether the iteration is implicit or explicit, but in this paper we will only be concerned with the numerical solution of nonstiff problems.

Predictor-corrector methods can be based on any appropriate class of discrete methods such as linear multistep, Runge–Kutta or multivalue methods, and have been proposed, for example, by Moulton [7] in conjunction with Adams methods and Rosser [9] in conjunction with Runge–Kutta methods. The underlying feature of a block method is that each application of the method generates a set of approximations to the solution. A block method can be one-step in nature where values in a new block depend only on the last value in the preceding block or can be multistep in nature so that values depend on information from previous blocks. Thus if \( Y_{n+1} \) represents a vector of solution values \( y_{n+c_1}, \ldots, y_{n+c_s} \) and \( F(Y_{n+1}) \) represents the vector of derivatives \( f(y_{n+c_1}), \ldots, f(y_{n+c_s}) \), where \( c_s \) may equal 1 if we use this last value as an output point, then an \( s \)-processor implementation would hope to gain a speedup of up to a factor of \( s \) by partitioning the problem and the method so that at each correction each processor simultaneously computes a subset of approximations.

The class of methods that we will consider for the solution of the nonstiff IVP

\[
y'(x) = f(y(x)), \quad y(x_0) = y_0, \quad f: \mathbb{R}^N \rightarrow \mathbb{R}^N,
\]

is based on a Hermite prediction given by

\[
Y^p = A_1 \otimes Y_n + hL_0 F \otimes (Y_n), \tag{1a}
\]

and a multivalue correction given by

\[
Y^c = A_2 \otimes Y_n + hL_1 F \otimes (Y_n) + hL_2 F(Y^p), \tag{2b}
\]

where \( A_1, A_2, L_0, L_1 \) and \( L_2 \) are all matrices of dimension \( s \). Note that we have removed the update stage by assuming \( c_s = 1 \) and using the approximation at this point as the update. This will be the formulation used henceforth in this paper.

We observe that, in general, this class of methods requires a starting procedure to generate \( Y_0 \) from \( y_0 \), the most natural being based on the correction of \( y_0 \) a certain number of times by an \( s \)-stage Runge–Kutta method. It was noted in [3,11] that if method (2) is corrected an arbitrary number of times, then it can be always written in a block explicit multivalue formulation and then the standard theories of order and stability for multivalue methods (see [1], for example) can be applied. Since the most important feature in the design of efficient explicit codes for the numerical solution of nonstiff problems is an acceptable local error behaviour, we cannot expect to design efficient and effective parallel methods without a comprehensive analysis of their local error properties. This is the main subject of this paper.

In Section 2 we present a numerical example which is illustrative of the performance of prediction-correction based on a low-order predictor and which together with the error analysis in [3] strongly suggests that if predictor-corrector methods are to be efficient solvers of nonstiff problems, then high-order predictors are required. The stability and local error behaviour of high-order predictor-corrector techniques with a small number of corrections is analysed in Section 3 and some conclusions and proposals for future work are given in Section 4.
2. A numerical example

Recently, predictor-corrector methods of the form described in (2) have been studied by a number of people including Tam [11] and van der Houwen and Sommeijer [12,13] who use a special subclass of (2) based on the trivial predictor of the solution value at the end of the step and a Runge-Kutta corrector, so that in (2)

\[ A_1 = A_2 = [0 \ e], \quad L_0 = L_1 = 0, \quad e = (1, \ldots, 1)^T, \]

where 0 represents rectangular matrices of zeros of appropriate dimensions.

It was noted in [12] that such methods can be written as a block explicit Runge-Kutta method and so their efficiency properties can be studied within the existing framework of Runge-Kutta theory. This was done by the author [3], who analysed the local error behaviour of methods based on the trivial predictor and a Runge-Kutta corrector, and in [2,5] where it was shown that if the starting procedure and predictor in (2) are of order \( p_1 \) and \( m \) corrections are performed with a Runge-Kutta corrector of order \( p \), then the order of (2) is at least \( \min\{p, p_1 + m\} \). A consequence of this is that if we use a trivial predictor and require a high-order method of order 8, say, then 8 corrections are needed.

We can judge the efficiency of such methods by comparing a method of order 8 based on the trivial predictor and a five-stage Lobatto corrector of order 8 which is corrected 8 times (called Lob8) with the thirteen-stage DORP7(8) Runge-Kutta method (see [8]), which is generally recognised as being a very efficient sequential method for tolerances within the range \( 10^{-6} \) to \( 10^{-13} \).

These two methods have been implemented with the same variable step strategy in Occam2 on a single T800 transputer running at 20 MHz on the two-body problem, on the interval \([0, 2\pi]\), with eccentricity 0.6, given by

\[
\begin{align*}
y'_1 &= y_3, & y_1(0) &= 1 - \epsilon, \\
y'_2 &= y_4, & y_2(0) &= 0, \\
y'_3 &= \frac{-y_4}{(y_1^2 + y_2^2)^{3/2}}, & y_3(0) &= 0, \\
y'_4 &= \frac{-y_2}{(y_1^2 + y_2^2)^{3/2}}, & y_4(0) &= \sqrt{\frac{1 + \epsilon}{1 - \epsilon}}. \\
\end{align*}
\]

The results obtained for a variety of different tolerances \( 10^{-2}, \ldots, 10^{-12} \) are given in Table 1 (with the time being given in terms of the number of ticks of the computer clock).

Comparing Lob8 and DORP7(8) we see that Lob8 takes approximately 3 times as many function evaluations, and a factor varying between 1.5 and 3.3 times as many ticks of the computer clock. Furthermore, as the error tolerance is made more stringent, the global error in Lob8 becomes approximately 4 times as large as that for DORP7(8). This problem is a very smooth one, and taking these results as typical we can make the following observations.

If function evaluations are expensive, so that they dominate computation time, then Lob8 is approximately 3 times as expensive as DORP7(8). But Lob8 is only a four-processor method (since the five-stage Lobatto method has as its first intermediate approximation the trivial calculation \( Y_1 = y_n \)), and so the maximum speedup is (ignoring communication time) at most \( \frac{4}{3} \).
Table 1

<table>
<thead>
<tr>
<th>Tol</th>
<th>Steps</th>
<th>Functions</th>
<th>Time</th>
<th>Global error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-2}$</td>
<td>8</td>
<td>128</td>
<td>2131</td>
<td>$1.79 \cdot 10^{-2}$</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>12</td>
<td>192</td>
<td>2127</td>
<td>$4.09 \cdot 10^{-4}$</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>18</td>
<td>318</td>
<td>2713</td>
<td>$4.46 \cdot 10^{-6}$</td>
</tr>
<tr>
<td>$10^{-8}$</td>
<td>28</td>
<td>496</td>
<td>3529</td>
<td>$6.15 \cdot 10^{-8}$</td>
</tr>
<tr>
<td>$10^{-10}$</td>
<td>48</td>
<td>816</td>
<td>5012</td>
<td>$9.97 \cdot 10^{-10}$</td>
</tr>
<tr>
<td>$10^{-12}$</td>
<td>82</td>
<td>1102</td>
<td>6321</td>
<td>$9.10 \cdot 10^{-12}$</td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>9</td>
<td>372</td>
<td>3283</td>
<td>$2.07 \cdot 10^{-2}$</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>13</td>
<td>564</td>
<td>4304</td>
<td>$7.07 \cdot 10^{-4}$</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>20</td>
<td>976</td>
<td>6500</td>
<td>$1.12 \cdot 10^{-5}$</td>
</tr>
<tr>
<td>$10^{-8}$</td>
<td>33</td>
<td>1572</td>
<td>9654</td>
<td>$2.52 \cdot 10^{-7}$</td>
</tr>
<tr>
<td>$10^{-10}$</td>
<td>53</td>
<td>2276</td>
<td>13347</td>
<td>$3.63 \cdot 10^{-9}$</td>
</tr>
<tr>
<td>$10^{-12}$</td>
<td>90</td>
<td>3784</td>
<td>21295</td>
<td>$3.78 \cdot 10^{-11}$</td>
</tr>
</tbody>
</table>

On the other hand, if function evaluations are cheap, then the more reliable figure is computer time and then at lax tolerances Lob8 is roughly only twice as expensive, and so the maximum speedup is 2 in this case. We note that in [12] similar results are obtained using a four-stage Gauss method of order 8 (called Gauss81) as a corrector.

These results suggest that a parallel implementation based on a trivial predictor and a Runge–Kutta corrector with a large number of correctors would yield a meagre speedup and that a high-order multistep-type predictor is needed to reduce the number of corrections.

In an attempt to explain these numerical results, and with the view to designing effective numerical methods for the numerical solution of nonstiff problems by controlling the magnitude and the nature of the truncation coefficients, the author [2,3] has developed a comprehensive theory based on the use of Butcher series which allows the analysis of the local error of any method of the form (2). He [3] has applied this technique to an analysis of the local behaviour of predictor-corrector methods based on the trivial predictor and an implicit Runge–Kutta corrector and has compared the local error behaviour of the Gauss8 method with the local error behaviour of DORP7(8). He has shown that the principal error coefficients of both these methods have comparable magnitude (although this is a function of the large number of corrections that are required in the predictor-corrector implementation), and that the exact difference depends on the type of norm used to measure the error coefficients and the nature of the elementary differentials that appear in the local error expansion.

3. High-order prediction

With the view to constructing efficient predictor-corrector methods with only one or two corrections, the author [2] has extended the local error analysis used in [3] to obtain formulas for the local error expansion of the class of methods based on using an s-stage collocation Runge–Kutta method of order $p > s$ as a corrector, while the predictor is obtained by interpolating the derivative values. Since the stage order of the corrector is $s$, the predictor will also have order $s$. This is an example of using an Adams-type predictor where the prediction is
based only on the value of the approximation at the end of the step and the \( s \) intermediate derivative approximations.

As mentioned above, we will treat the updating stage as one of the \( s \) components, so that \( c_s = 1 \), and we can write our class of methods in the form of (2) as

\[
Y^{(0)} = e \otimes y_n + hA_p \otimes F(Y_n),
\]

\[
Y^{(k+1)} = e \otimes y_n + hA \otimes F(Y^{(k)}), \quad k = 0, \ldots, m - 1,
\]

\[
y_{n+1} = e_s^T Y^{(m)},
\]

where \( L_2 = A \), the Runge–Kutta matrix, and \( L_1 = A_p \), the prediction matrix.

It is well known that a Runge–Kutta method in which there is no update stage is a collocation method of order \( s \) if the following relationship holds:

\[
C(s): \quad Ac^{k-1} = k, \quad k = 1, \ldots, s.
\]

Let \( V \), \( W \) and \( \hat{V} \) be the \( s \times s \) matrices whose \( j \)th columns are, respectively, \( c^{j-1} \), \( c^j/j \) and \( (c-e)^{j-1} \); then a predictor based on the \( s \) past derivative values \( hf(Y_1), \ldots, hf(Y_s) \) gives

\[
A = WV^{-1}, \quad A_p = W\hat{V}^{-1}.
\]

If we apply this method with \( m \) corrections to the standard linear test problem \( y' = qy \) with \( z = qh \), we find that

\[
Y_{n+1} = M(z)Y_n,
\]

where

\[
M(z) = F + \sum_{j=1}^{m} A^j Ez^j + z^{m+1}A^m A_p, \quad E = [0 \ e].
\]

Furthermore, if the corrector is a collocation method and \( m \leq s \), then

\[
M(z) = E + \sum_{j=1}^{m} \frac{E_j z^j}{j!} + z^{m+1}A^m A_p,
\]

where \( E_j \) is the matrix with zeros in the first \( s - 1 \) columns and \( c^j \) in the last column.

<table>
<thead>
<tr>
<th>Method</th>
<th>Imag</th>
<th>Bisect</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lob5(1)</td>
<td>0.1</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Lob5(10)</td>
<td>4.3</td>
<td>4.3</td>
<td>4.3</td>
</tr>
<tr>
<td>Lob8(1)</td>
<td>0.1</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>Lob8(2)</td>
<td>0.3</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>NC8(1)</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
</tr>
</tbody>
</table>
As a brief introduction to the way in which the abscissae and the number of corrections can affect the stability of a method, we present stability information in Table 2. Here Lob5 and Lob8 represent collocation correctors based on the Lobatto abscissae with 5 and 8 stages (and with a multistep predictor as described in (4)), respectively, while NC8 represents a collocation corrector based on equidistant abscissae with 8 stages. The number in parentheses after the method in the above table represents the number of corrections; while the three values represent the size of the stability region along the imaginary axis, along the bisector of the imaginary axis and the negative real axis and along the negative real axis. These results and others elsewhere suggest that for a fixed number of corrections the choice of the free parameters (namely the abscissae for the methods being considered here) does not unduly affect the size of the stability region (although they can have some effect along the imaginary axis, as in NC8(1) and Lob8(1)). The most important factor in the size of the stability region appears to be the number of corrections. Thus the family of methods we are considering based on only one correction are unlikely to be efficient for problems which are mildly stiff because the stepsize will be constrained by stability considerations rather than accuracy.

Rather than give a detailed analysis of the local error expansion of the predictor-corrector methods in general (see [2], for example) we will attempt to summarise some of the significant results of that paper.

Let \( t = [t_1, \ldots, t_m] \) represent the tree formed by joining the subtrees \( t_1, \ldots, t_m \) by a single branch to a common root and if \( \phi \) and \( \tau \) represent the empty tree and the unique tree of order 1, respectively, then the elementary differentials associated with (1) are defined by

\[
\bar{F}(\phi)y = y, \quad \bar{F}(\tau)y = f(y), \quad \bar{F}(t)y = f^m(\bar{F}(t_1)y, \ldots, \bar{F}(t_m)y).
\]

We also denote by \( \rho(t) \) the order of \( t \), by \( h(t) \) the height of the tree (where \( h(\tau) = 0 \)), by \( b(t) \) the number of branches emanating from the root of \( t \) and denote by \( t_w \) an arbitrary tree of order \( w \).

In addition, to this notation we need the concept of a Butcher series which was introduced in [4]. Thus if \( T \) and \( T^* \) denote, respectively, the set of rooted trees and monotonically labelled trees (where the integer 1 is attached to the root and thereafter the integer labels increase in size as the scanning moves up the branches of the tree), then if \( q_i : T \rightarrow \mathbb{R} \) is an arbitrary map, then the series

\[
B(q_i, y_0) = \sum_{t \in T^*} q_i(t)\bar{F}(t)(y_0) \frac{h^\rho(t)}{\rho(t)!}
\]

is said to be a Butcher series. Using the B-series theory the author [2,3] has shown that for method (3) the local error vector \( d_{n+1} \) has an expansion \( B(\bar{e}, y(x_n)) \), as a Butcher series given by

\[
d_{n+1} = \sum_{t \in T^*} \bar{e}(t)\bar{F}(t)y(y_n) \frac{h^\rho(t)}{\rho(t)!},
\]

where, for \( t = [t_1, \ldots, t_v] \),

\[
\bar{e}(t) = c^{\rho(t)} - k_{m+1}(t),
\]
and

\[ k_j(\phi) = e, \quad k_j(t) = \rho(t) A_p \prod_{i=1}^{\nu} k_0(t_i), \]

\[ k_{j+1}(\phi) = e, \quad k_{j+1}(t) = \rho(t) A \prod_{i=1}^{\nu} k_j(t_i), \quad j = 1, \ldots, m. \] (8)

Here we have assumed that \( Y_0 \) has a Butcher series expansion as \( B(k_0, y(x_0)) \) and we observe that the measure of the local truncation error is given by \( e^T d_{n+1}, e^T = (0, \ldots, 0, 1) \).

In fact, the most obvious way to generate \( Y(\phi) \) from the initial value is to use the trivial predictor in conjunction with a Runge–Kutta method corrected a certain number of times. Using the results in [3], or modifying (8) with \( A_p = 0 \), it is easy to show that if the corrector has stage order \( p \) and \( p \) corrections are performed, as part of the starting procedure, then

\[ k_0(t) = (c - e)^{\rho(t)}, \quad k_0(t_{p+1}) = (p + 1)A(c - e)^{p}, \quad h(t_{p+1}) \neq p, \]

\[ k_0(t_{p+1}) = 0, \quad h(t_{p+1}) = p, \] (9)

while if \( p + 1 \) corrections are performed, then

\[ k_0(t) = (c - e)^{\rho(t)}, \quad \forall \rho(t) \leq p, \quad k_0(t_{p+1}) = (p + 1)A(c - e)^p, \]

\[ k_0(t_{p+1}) = 0, \quad h(t_{p+1}) = p, \] (10)

where \( t_{p+1} \) denotes an arbitrary tree of order \( p + 1 \).

We can now derive explicit expressions for the local error of the methods being considered here in terms of the local error of the Runge–Kutta collocating corrector method. Let the Runge–Kutta corrector have a local error given by

\[ \tilde{d}_{n+1} = \sum_{t \in T^*} \tilde{e}(t)[\tilde{F}(t)] y(x_n) \frac{h^{\rho(t)}}{\rho(t)!}, \] (11)

where, for \( t = [t_1, \ldots, t_v] \),

\[ \tilde{e}(t) = c^{\rho(t)} - k(t), \] (12)

with

\[ k(\phi) = e, \quad k(t) = \rho(t) A \prod_{i=1}^{\nu} k(t_i). \] (13)

Then it is easy to show by a simple induction argument that

\[ k_m(t) = c^{\rho(t)}, \quad \forall \rho(t) \leq p, \quad m - 1, 2, \ldots, \]

\[ k_m(t) = k(t), \quad \forall \rho(t) \leq p + m - 1. \] (14)

The author [2] has obtained formulas for \( k_m \) for trees of order greater than \( p + m - 1 \), but they become increasingly more complicated as the order of the tree increases. For the predictor-corrector format discussed in (2) in which the starting procedure and predictor are of order \( p \), the author [2] has shown for an arbitrary tree \( t_{p+m} \) of order \( p + m \),

\[ k_m(t_{p+m}) = k(t_{p+m}), \quad \forall t_{p+m} \neq [t_p]_m, \]

\[ k_m(t_{p+m}) = (p + m) \cdots (p + 1)A^{m-1}A_p(c - e)^p, \quad t_{p+m} = [t_p]_m. \] (15)
while for an arbitrary tree $t_{p+m+1}$ of order $p + m + 1$,

$$k_m(t_{p+m+1}) = k(t_{p+m+1}), \quad \forall t_{p+m+1} \neq [t_{p+2}]_{m-1}, \quad [\tau, [t_p]_{m-j}], \quad j = 1, \ldots, m - 1,$$

$$k_m([\tau, [t_p]_{m-j}]) = \frac{(p + m + 1) \cdots (p + 1)}{p + m + 1 - j} A^i C A^{m-j-1} A_p(c - e)^p,$$

$$j = 1, \ldots, m - 1,$$

$$k_m([t_{p+2}]_{m-1}) = (p + m + 1) \cdots (p + 2) A^{m-1} A_p(c - e)^p, \quad b(t_{p+2}) \neq 1,$$

$$k_m([t_{p+1}]_m) = (p + m + 1) \cdots (p + 2) A^{m-1} A_p k_0(t_{p+1}),$$

(16)

where $C = \text{diag}(c_1, \ldots, c_s)$.

Since the maximum stage order of an $s$-stage Runge–Kutta method is $s$, we will now assume that the predictor and the corrector have stage order $s$ and that the starting procedure is of order $s$ also and is based on the correction $s$ times by a collocating Runge–Kutta corrector of the trivial predictor. Hence we can use the results in (15) and (16) and write down the following expansion for the local truncation error after one correction. Here, for example, $\Sigma [\tilde{F}(t_{s+1})] y(x_n)$ represents the sum over all elementary differentials corresponding to all trees of order $s + 1$.

**Prediction:**

$$\text{LTE}_p = \frac{h^{s+1}}{s!} G_{p1} \sum [\tilde{F}(t_{s+1})] y(x_n)$$

$$+ \frac{h^{s+2}}{(s + 1)!} \left( G_{p2} \sum_{b(t_{s+2}) \geq 2} [\tilde{F}(t_{s+2})] y(x_n) + G_{p3} \sum_{b(t_{s+2}) = 1} [\tilde{F}(t_{s+2})] y(x_n) \right)$$

$$+ O(h^{s+3}),$$

(17)

where

$$G_{p1} = \frac{1}{s + 1} - e_s^T A_p(c - e)^s, \quad G_{p2} = \frac{1}{s + 2} - e_s^T A_p(c - e)^{s+1},$$

$$G_{p3} = \frac{1}{s + 2} - (s + 1) e_s^T A_p k_0(t_{s+1}), \quad e_s^T = (0, \ldots, 0, 1).$$

**Correction:**

$$\text{LTE}_c = \frac{h^{s+1}}{s!} G_{c1} \sum [\tilde{F}(t_{s+1})] y(x_n) + \frac{h^{s+2}}{(s + 1)!} G_{c2} \sum_{b(t_{s+2}) = 1} [\tilde{F}(t_{s+2})] y(x_n)$$

$$+ \frac{h^{s+2}}{(s + 1)!} G_{c3} \sum_{b(t_{s+2}) > 1} [\tilde{F}(t_{s+2})] y(x_n)$$

$$+ \frac{h^{s+3}}{(s + 2)!} G_{c4} \sum_{t_{s+3} \in \{\tau, t_{s+1}\}} [\tilde{F}(t_{s+3})] y(x_n)$$

(18)
\[ h^{s+3} \sum_{b(t_{s+2})=1}^{\infty} \left[ \bar{F}(\{t_{s+2}\}) \right] y(x_n) \]

\[ + \frac{h^{s+3}}{(s+2)!} G_{c6} \sum_{b(t_{s+2})>2} \left[ \bar{F}(\{t_{s+2}\}) \right] y(x_n) \]

\[ + \frac{h^{s+3}}{(s+2)!} G_{c7} \sum_{b(t_{s+2})=1}^{\infty} \left[ \bar{F}(\{t_{s+2}\}) \right] y(x_n) + O(h^{s+4}), \]

where

\[ G_{c1} = \frac{1}{s+1} - e_s^T A e^s, \]

\[ G_{c2} = \frac{1}{s+2} - e_s^T A e^{s+1}, \]

\[ G_{c3} = \frac{1}{s+2} - (s+1)e_s^T A A_p(c - e)^s, \]

\[ G_{c4} = \frac{1}{s+3} - e_s^T A e^{s+2}, \]

\[ G_{c5} = \frac{1}{s+3} - (s+1)e_s^T A A_p(c - e)^s, \]

\[ G_{c6} = \frac{1}{s+3} - (s+2)e_s^T A A_p(c - e)^{s+1}, \]

\[ G_{c7} = \frac{1}{s+3} - (s+2)(s+1)e_s^T A A_p k_0(t_{s+1}), \]

\[ C = \text{diag}(c_1, \ldots, c_s). \]

We note that if the corrector has order \( s + 1 \), then \( G_{c1} = 0 \) and henceforth we will make this assumption.

The values of \( k_0(t_{s+1}) \) in both (18) and (20) depend on the nature of the starting procedure and the number of initial corrections (we have assumed at least \( s \), see [2]). In an attempt to improve upon the efficiency of Lob8 (see Table 1) we attempt to minimize the local truncation coefficients of the corrector in (19) in some norm. We ignore the contribution of error terms higher than order \( s + 3 \) with the hope that they contribute significantly less to the local error than the error terms of orders \( s + 2 \) and \( s + 3 \). However, as noted in [3], the influence of the predictor error can take a long time to dissipate and to derive truly efficient methods we may have to look at higher-order terms.

But, the number of higher-order terms grows very quickly so that in this paper we attempt to construct efficient methods based only on controlling the sum of the magnitude of the principal error coefficients and those of the next order \( (s + 3) \). Initially we ignore the contribution of the term arising directly from the starting procedure (that is \( k_0(t_{s+1}) \)), so that the quantity we wish to normalise is \( N = \sum_{j=2}^{6} |G_{cj}| \).

The rationale behind controlling only the error coefficients after one correction and not those in the predictor is that the block predictor-corrector methods are implemented using local extrapolation. That is, we advance on the higher-order method and we use the difference between the predictor and the corrector as an estimate of the local truncation error to control the local error in the usual way. Consequently, we require the corrector to be order one higher than the predictor and we do not attempt to control the local truncation error of the predictor, but as will be seen from Table 3 this behaves satisfactorily.
Table 3

<table>
<thead>
<tr>
<th>Method</th>
<th>$G_{p1}$</th>
<th>$G_{p2}$</th>
<th>$N_1$</th>
<th>$N_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lob8(1)</td>
<td>2.76·10^{-6}</td>
<td>2.75·10^{-7}</td>
<td>1.06·10^{-5}</td>
<td>4.75·10^{-6}</td>
</tr>
<tr>
<td>NC8(1)</td>
<td>8.19·10^{-5}</td>
<td>2.86·10^{-5}</td>
<td>1.15·10^{-5}</td>
<td>5.16·10^{-6}</td>
</tr>
<tr>
<td>New8</td>
<td>2.76·10^{-6}</td>
<td>2.76·10^{-7}</td>
<td>2.17·10^{-12}</td>
<td>2.29·10^{-11}</td>
</tr>
</tbody>
</table>

We also denote by $N_1 = |G_{c2}| + |G_{c3}|$ the $L_1$ norm associated with the error terms of order $s + 2$ and by $N_2 = |G_{c4}| + |G_{c5}| + |G_{c6}|$ the $L_1$ norm of the error terms of order $s + 3$ (we exclude the $k_0(t_{s+1})$ term). We used MATLAB to minimise $N$ over a particular subclass of the parameter space, namely that the abscissae are the zeros of the collocating polynomial

$$Q(x) = P_s(x) - (k + m)P_{s-1}(x) - (1 + l)P_{s-2}(x) + kP_{s-3}(x) + lP_{s-4}(x) + mP_{s-5}(x).$$

Here the $P_j(x)$ are the Legendre polynomials of degree $j$ orthogonal on [0, 1]. We note that if $k = l = m = 0$, then we have Lobatto quadrature of order $2s - 2$. This minimization results in the method denoted by New8 in Table 3 in which $(k, l, m) = (8, -15.7933806, 0.109735)$, while Lob8(1) and NC8(1) represent one-correction methods in which the corrector has 8 stages and is based on either Lobatto abscissae or equidistant abscissae, respectively.

We implemented the three methods on the same two-body problem in a variable stepsize program using local extrapolation and a starting procedure which consists of $s$ corrections by the appropriate $s$-stage Runge–Kutta corrector and present the results in Table 4.

Table 4

<table>
<thead>
<tr>
<th>Tol</th>
<th>Steps</th>
<th>Functions</th>
<th>Time</th>
<th>Global error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lob8(1)</td>
<td>$10^{-2}$</td>
<td>18</td>
<td>329</td>
<td>4808</td>
</tr>
<tr>
<td></td>
<td>$10^{-4}$</td>
<td>26</td>
<td>476</td>
<td>6512</td>
</tr>
<tr>
<td></td>
<td>$10^{-6}$</td>
<td>44</td>
<td>812</td>
<td>9825</td>
</tr>
<tr>
<td></td>
<td>$10^{-8}$</td>
<td>73</td>
<td>1232</td>
<td>13212</td>
</tr>
<tr>
<td></td>
<td>$10^{-10}$</td>
<td>120</td>
<td>1904</td>
<td>18090</td>
</tr>
<tr>
<td></td>
<td>$10^{-12}$</td>
<td>1034</td>
<td>14651</td>
<td>101513</td>
</tr>
<tr>
<td>New8</td>
<td>$10^{-2}$</td>
<td>17</td>
<td>329</td>
<td>4804</td>
</tr>
<tr>
<td></td>
<td>$10^{-4}$</td>
<td>22</td>
<td>399</td>
<td>5428</td>
</tr>
<tr>
<td></td>
<td>$10^{-6}$</td>
<td>34</td>
<td>644</td>
<td>7858</td>
</tr>
<tr>
<td></td>
<td>$10^{-8}$</td>
<td>52</td>
<td>903</td>
<td>10280</td>
</tr>
<tr>
<td></td>
<td>$10^{-10}$</td>
<td>79</td>
<td>1302</td>
<td>13390</td>
</tr>
<tr>
<td></td>
<td>$10^{-12}$</td>
<td>127</td>
<td>2072</td>
<td>19684</td>
</tr>
<tr>
<td>NC8(1)</td>
<td>$10^{-2}$</td>
<td>17</td>
<td>315</td>
<td>4664</td>
</tr>
<tr>
<td></td>
<td>$10^{-4}$</td>
<td>27</td>
<td>490</td>
<td>6657</td>
</tr>
<tr>
<td></td>
<td>$10^{-6}$</td>
<td>44</td>
<td>805</td>
<td>9654</td>
</tr>
<tr>
<td></td>
<td>$10^{-8}$</td>
<td>74</td>
<td>1239</td>
<td>13136</td>
</tr>
<tr>
<td></td>
<td>$10^{-10}$</td>
<td>121</td>
<td>1918</td>
<td>18243</td>
</tr>
<tr>
<td></td>
<td>$10^{-12}$</td>
<td>2810</td>
<td>39557</td>
<td>270656</td>
</tr>
</tbody>
</table>
4. Conclusions and future work

Although we have given numerical results for only a single test problem, these results are typical of most small, nonstiff problems, and, consequently, we can draw the following conclusions.

- Methods based on one correction by either a Newton–Cotes collocation method or by a Lobatto collocation method give approximately the same performance. This suggests that there is little to choose in terms of efficiency between collocation methods which have their abscissae in \([0, 1]\). However, if two corrections are performed, the Lobatto methods are slightly more efficient at stringent tolerances (see [2]) because of their higher order.

- Comparing Lob8(1) with DORP8 and Lob8 (in Table 1) we see that in terms of function evaluations, Lob8 needs approximately 1.2 times as many function evaluations as Lob8(1) but that Lob8(1) takes approximately 1.5 times as long as Lob8 in ticks of the computer clock. The reason for this is due to the fact that Lob8 is purely a Runge–Kutta method while Lob8(1) is a multivalue method and a stepsize change means recalculation of the coefficients in this case. In the sequential implementation this recalculation is done for each component of the block on a single processor; but in a parallel implementation this recalculation of coefficients can be spread over a number of processors, so that if function evaluations are not cheap, Lob8(1) would still be slightly more efficient than Lob8 in a parallel environment.

- Comparing Lob8(1) and New8, we see that by a judicious choice of the three free parameters in the collocating polynomial \(Q\) we find that Lob8(1) is between 1.2 and 1.5 times as expensive (except at very lax tolerances) as New8, with similar results for computer time. Furthermore, when we compare New8 and Lob8 (the trivial predictor and 8 corrections), we find that Lob8 takes between 1.2 and 2 times as many function evaluations and, in addition, New8 has on average (at stringent tolerances) an extra digit of accuracy.

- Finally, comparing New8 and DORP8, we observe that New8 takes between 2.5 and 1.6 times as many function evaluations, so that there is a possible maximum speedup of \(7/1.6 \approx 4.4\) depending on the error tolerance.

- The approach adopted in this paper, of constructing a theoretical framework which allows the analysis of the local truncation error of a general class of predictor-corrector methods and then the construction of certain predictor-corrector methods based on the minimization of the norm of the principal error coefficients and the error coefficients for the next-order term, has enabled us to gain a significant improvement in efficiency over methods based on a standard Runge–Kutta collocating corrector with abscissae in the interval \([0, 1]\) — such as those based on Lobatto or Newton–Cotes quadrature. Apparently, greater efficiency is gained by allowing some of the abscissae to be in the interval \([1, 2]\), so that part of the prediction step is an interpolation, rather than purely extrapolation, as is the case if all of the abscissae lie in \([0, 1]\).

It seems entirely possible to eventually be able to construct methods with a block size of 8 or 9 with a sequential efficiency equal if not superior to DORP8. In order to accomplish this, we may need to do a number of things: weaken the stage order of the corrector to \(s - 1\), take the error behaviour of the starting procedure completely into account, include higher-order terms in the local error analysis and perhaps take an extra correction (which will also increase the stability region). This would give a maximum speedup across the method of about 8 in a parallel environment, which is not insignificant.
References


