Efficient parallel predictor-corrector methods

J.J.B. de Swart

Department of Numerical Mathematics

Report NM-R9417 September 1994

CWI is the National Research Institute for Mathematics and Computer Science. CWI is part of the Stichting Mathematisch Centrum (SMC), the Dutch foundation for promotion of mathematics and computer science and their applications.
SMC is sponsored by the Netherlands Organization for Scientific Research (NWO). CWI is a member of ERCIM, the European Research Consortium for Informatics and Mathematics.

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

Efficient Parallel Predictor-Corrector Methods

J.J.B. de Swart

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

Recently, the so-called Adams-Bashforth-Radau (ABR) methods were proposed in [4]. An ABR method is a high-order parallel predictor-corrector method for solving non-stiff initial value problems, based on a combination of Adams-Bashforth and Radau formulas. Comparison of ABR with the famous sequential 8(7) Runge-Kutta method of Dormand and Prince showed speed-up factors of about 2.7. In this paper we improve the ABR methods by making them more accurate without any additional costs. This improved version increases the speed-up factor on the average to 3.1.

CR Subject Classification (1991): G.1.7

Keywords & Phrases: numerical analysis, predictor-corrector iteration, Runge-Kutta methods, parallelism.

Note: The research reported in this paper was supported by STW (Netherlands Foundation for the Technical Sciences).

1. Introduction

We shall consider predictor-corrector methods (PC methods) for solving on parallel computers the (non-stiff) initial value problem

$$y'(t) = f(y(t)) \quad , \quad y(t_0) = y_0 \quad , \quad y, f \in \mathbb{R}^d. \tag{1.1}$$

In [4] a class of parallel PC methods has been proposed, including the Adams-Bashforth-Radau (ABR) methods. These methods showed a speed-up factor of about 2.7 compared to DOPRI8. The DOPRI8 code by Hairer-Nørsett-Wanner [2] is an implementation of the 13-stage, 8^{th}-order embedded Runge-Kutta method of Dormand and Prince, and is generally accepted as one of the best sequential codes. In this paper we improve the ABR methods by increasing the order by 1. The convergence and stability characteristics turn out to be even slightly better than those of ABR, while the sequential costs and the number of processors are (almost) the same.

The outline of the paper is as follows. In section 2 we specify a subclass of the large class of General Linear Methods, introduced by Butcher in 1966, and describe how methods that fall into this class can be compared by means of accuracy, stability and convergence. Section 3 briefly describes the ABR methods proposed in [4]. In section 4 we propose a more accurate variant of ABR. How this variant can be implemented without any additional costs compared to ABR is presented in section 5. Finally, in section 6, numerical experiments will show that this new variant indeed performs better than ABR.

2. A subclass of the class of General Linear Methods

In the following, the vector with unit entries is denoted by e, the i^{th} canonical basis vector by e_i, and the $d \times d$ identity matrix by I_{dd}. Furthermore, O_{mn} is the $m \times n$ zero matrix and

Report NM-R9417
ISSN 0169-0388
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

E_{mn} is the $m \times n$ matrix whose entries are zero, except for its n^{th} column which equals e. If v is a vector, v^j stands for the vector whose entries are the j^{th} powers of the entries of v.

To solve (1.1) we use methods of the form

$$Y_n = (A \otimes I_{dd})Y_{n-1} + h(B \otimes I_{dd})F(Y_{n-1}) + h(C \otimes I_{dd})F(Y_n), \tag{2.2}$$

$$y_n = y_{n-1} + h(c_s^T \otimes I_{dd})F(Y_n). \tag{2.3}$$

This type of methods falls into the class of General Linear Methods introduced by Butcher (see [1]).

Here the $s \times s$ matrices A, B, C and the s-dimensional vector c_s^T contain the method parameters, h denotes the step-size $t_n - t_{n-1}$ and $\otimes$ denotes the Kronecker product. Y_n is the so called *stage vector* which represents numerical approximations to the exact solution vectors $y(et_{n-1} + ah)$, where the s-dimensional vector a denotes the abscissa vector. Hence Y_n is an sd-dimensional vector. In this paper we restrict ourselves to the case where the components of a are the Radau IIA collocation points. For any vector $V = (V_i)$, $F(V)$ contains the derivative values $(f(V_i))$.

The formulas (2.2) and (2.3) are respectively called the *stage vector equation* and the *step point formula.*

Considering (2.2) as the correction equation we solve this equation by applying the PC scheme

$$Y_n^{(0)} = (A_0 \otimes I_{dd})Y_{n-1} + h(B_0 \otimes I_{dd})F(Y_{n-1}), \tag{2.4}$$

$$Y_n^{(j)} = (A \otimes I_{dd})Y_{n-1} + h(B \otimes I_{dd})F(Y_{n-1}) + h(C \otimes I_{dd})F(Y_n^{(j-1)}), \qquad j = 1, \ldots, m, \tag{2.5}$$

$$Y_n = Y_n^{(m)}.$$

Next we describe how accuracy, stability and convergence of the PC scheme can be defined in terms of A, B, C and c_s.

2.1 Accuracy

The conditions for p^{th}-order consistency of the stage vector equation (2.2) are given by (see, e.g. [3])

$$Ae = e\ ,\ AX_{sp} + BW_{sp} + CV_{sp} = U_{sp}, \tag{2.6}$$

where the $s \times p$ matrices U_{sp}, V_{sp}, W_{sp} and V_{sp} are defined by

$$U_{sp} := (\tfrac{1}{j}a^j) \quad , \quad V_{sp} := (a^{j-1})$$
$$W_{sp} := ((a-e)^{j-1}) \quad , \quad X_{sp} := (\tfrac{1}{j}(a-e)^j) \qquad \text{for } j = 1, \ldots, p.$$

If (2.6) is satisfied, then p will be called the *stage order.*

Note that, for $A = E_{ss}, B = O_{ss}$, and C fulfilling (2.6) with $p = s$, (2.2) reduces to the s-stage Radau IIA method.

In this paper we use a step point formula that coincides with the formula for the s^{th} stage of the Radau IIA method: $c_s^T = e_s^T U_{ss} V_{ss}^{-1}$. We will refer to this formula as the *Radau IIA step point formula.* It can be shown (see [4]) that for this case, the order of y_n (the so called *step point order*) equals $\min\{2s-1, p+1\}$, where p again denotes the stage order.

2.2 Stability

With respect to the scalar test equation $y' = \lambda y$, where λ runs through the spectrum of the Jacobian $\frac{\partial f(y)}{\partial y}$, we obtain for (2.2) the recursion

$$Y_n = M(z)Y_{n-1}\ ,\ M(z) := (I - zC)^{-1}(A + zB)\ ,\ z := \lambda h.$$

We define the *stability region* and the *real* and *imaginary stability intervals* according to

$$\begin{aligned} \mathbf{S} &:= \{z \in \mathbb{C} \mid \rho(M(z)) < 1\}, \\ (-\beta_{\text{re}}, 0) &:= \{z \in \mathbb{C} \mid z \in \mathbf{S} \wedge z < 0\}, \\ (-\beta_{\text{im}}, \beta_{\text{im}}) &:= \{z \in \mathbb{C} \mid z \in \mathbf{S} \wedge \text{Re}(z) = 0 \wedge z \neq 0\}, \end{aligned}$$

respectively, where $\rho(\cdot)$ denotes the spectral radius function. β_{re} and β_{im} are called the *real* and *imaginary stability boundary*, respectively.

For many methods that we consider in the next sections, it turns out that $\beta_{\text{im}} = 0$. To circumvent the numerical uncertainty we also computed the *practical imaginary stability interval* defined by $(-\beta^*_{\text{im}}, \beta^*_{\text{im}}) := \{z \in \mathbb{C} \mid \rho(M(z)) < 1 + 10^{-3} \wedge \text{Re}(z) = 0 \wedge z \neq 0\}$. In practical computations, β^*_{im} can be safely used as the imaginary stability boundary.

2.3 Convergence

For the convergence analysis of (2.5) we define the iteration error

$$\epsilon^{(j)} := Y_n^{(j)} - Y_n.$$

Application to the scalar test equation $y' = \lambda y$ and substitution in (2.5) yield

$$\epsilon^{(j)} := z^j C^j \epsilon^{(0)}\ ,\ z := \lambda h,$$

and consequently

$$\|\epsilon^{(m)}\|_\infty \leq |z|^m \|C^m\|_\infty \|\epsilon^{(0)}\|_\infty.$$

This leads us to defining the *region of convergence* by

$$\mathbf{C}_m := \{z \in \mathbb{C} |\ |z| < \gamma_m\}\ ,\ \gamma_m := \frac{1}{\sqrt[m]{\|C^m\|_\infty}},$$

where γ_m may be considered as the convergence boundary.

3. Adams-Bashforth-Radau Methods

Let us write the matrix C in the form

$$C = \begin{pmatrix} \overline{C}_1 & \overline{C}_2 \\ \underline{C}_1 & \underline{C}_2 \end{pmatrix},$$

where $\overline{C}_1$ and $\underline{C}_2$ are square matrices of size $q \times q$ and $r \times r$ respectively ($q + r = s$). From now on, upper and under bars refer to the first q and last r rows of an array.

Our first examination of methods of type (2.2) led to the observation that the convergence factors γ_m become larger as the order of consistency increases. In particular, we observed that the entries of $\underline{C}_2$ are relatively small. So ideally we would like to iterate solely with $\underline{C}_2$ and therefore we considered methods with $\overline{C}_1 = O_{qq}$ and $\overline{C}_2 = O_{qr}$. Thus the first q stages become explicit while the remaining r stages are solved by an iteration process that is determined by a 'small' matrix $\underline{C}_2$. The method can now be viewed as an r-processor method, since the iteration process is only invoked on r implicit stages, which can be evaluated in parallel.
If we choose $\underline{B} = O_{rs}$ and define the matrices $\overline{B}$, $\underline{C}_1$ and $\underline{C}_2$ by order conditions, while A is identified with the matrix E_{ss}, we see that both the q explicit and the r implicit stages are given order s. This method was called Adams-Bashforth-Radau (ABR) in [4].

In order to get reasonably large stability intervals, the number of implicit stages has to exceed the number of explicit stages ($r > q$). The characteristics of a few ABR methods are listed in Table 1.

For the predictor matrices A_0 and B_0 we can take $A_0 = E_{ss}$ and $B_0 = U_{ss}W_{ss}^{-1}$, i.e. B_0 is defined by order conditions. In [4] we referred to this predictor as the Adams-Bashforth (AB) predictor. Note that the first q rows of B_0 now coincide with $\overline{B}$. Hence for the first q stages we do not apply a corrector anymore after the prediction.

Here and in the following we assume that the costs of an algorithm are mainly determined by the number of right hand side evaluations (denoted shortly by f-evaluations).

Since f-evaluations of different stage vector components can be done in parallel, the costs of ABR on r processors per time step are m sequential f-evaluations for the corrector and, provided that $q \leq r$, 2 sequential f-evaluations for the predictor. If we apply an economization by replacing $F(Y_{n-1})$ in (2.4) and (2.5) by

$$F_{n-1}^{*} := \begin{pmatrix} F(\overline{Y}_{n-1}^{(0)}) \\ F(\underline{Y}_{n-1}^{(m-1)}) \end{pmatrix},$$

then the sequential costs are reduced to $m + 1$ f-evaluations per time step.

4. Improved Adams-Bashforth-Radau Methods

For ABR methods in every row $s + 1$ elements in the matrices A, B and C are determined by order conditions. Consequently, these methods have stage order s. In order to increase the stage order by 1 we have to impose an additional condition on each row of the parameter matrices. For the r implicit and q explicit stages this could be done by filling the s^{th} column in $\underline{B}$ and the $(s-1)^{th}$ column in $\overline{A}$, respectively. The drawback of this approach is that it leads to large elements in $\overline{A}$ (for instance, if $q = 2$, $s = 6$ and $A = (a_{ij})$, then $a_{26} \approx 105$). However, it turns out that this problem does not arise in the first row. Therefore we only use this strategy for the first stage. The order of the remaining $q-1$ explicit stages will be raised by 1 by using the first column of $\overline{C}$. Remark that, strictly spoken, the last $q-1$ explicit stages become implicit in this way. In the next paragraph we will see how to handle this aspect. This approach does not lead to large coefficients and, provided that some constraints are put on the size of q and r, the sequential costs of the resulting scheme will be the same as for the ABR methods. Since these methods are much alike ABR and have a higher stage order, we will refer to them as *improved* ABR.

Table 1: Characteristics for selected ABR correctors

s	=	$q+r$	stage order	order	β_{re}	β^*_{im}	γ_2	γ_3	γ_4	γ_{10}	...	γ_∞
5	=	2 + 3	5	6	1.97	1.89	2.45	3.08	3.47	5.80	...	7.03
6	=	2 + 4	6	7	3.35	2.86	2.04	2.61	3.15	5.80	...	7.74
7	=	2 + 5	7	8	5.23	4.57	1.84	2.36	2.85	5.40	...	8.39
8	=	3 + 5	8	9	0.40	0.43	2.19	2.80	3.38	6.35	...	10.33

Table 2: Characteristics for selected *improved* ABR correctors

s	=	$q+r$	stage order	order	β_{re}	β^*_{im}	γ_2	γ_3	γ_4	γ_{10}	...	γ_∞
5	=	2 + 3	6	7	2.60	3.27	2.47	3.17	3.66	6.45	...	7.85
6	=	2 + 4	7	8	3.84	5.85	2.05	2.63	3.20	6.04	...	8.73
7	=	2 + 5	8	9	5.24	8.08	1.85	2.38	2.89	5.66	...	9.55
8	=	3 + 5	9	10	0.97	1.39	2.20	2.82	3.42	6.56	...	11.34

As a consequence of the higher order of *improved* ABR, we expect the convergence characteristics to improve. Furthermore, the stability regions should become larger than those of ABR, since *improved* ABR is 'somewhat more implicit' by the $q-1$ additional elements in $\overline{C}_1$. Comparing the Tables 1 and 2 confirm these expectations.

If we add the Radau IIA step point formula, the step point order equals $\min\{2s-1, p+1\} = s+2$, provided that $s \geq 3$. For ABR this step point formula can be applied without any additional work, since the s^{th} stage already coincides with the last stage of the Radau IIA method. For *improved* ABR this is no longer true, since the last row of $\underline{B}$ contains a non-zero element. However, this element turns out to be very small (for example, if $s = 7$ and $B = (b_{ij})$, then $b_{77} \approx -1.3 \cdot 10^{-12}$). Therefore in practical applications, where $s \geq 3$, we observe step point order s+2 without a step point formula (see section 6).

5. The computation Scheme

In this section we show how *improved* ABR methods can be implemented on r processors without any additional costs compared to ABR. The idea is to take advantage from the observation that the number of implicit stages has to exceed the number of explicit stages in order to get reasonably large stability regions.

If the number of fixed point iterations is again denoted by m, the economized version of the ABR algorithm requires m sequential f-evaluations for the r implicit stages, plus 1 sequential f-evaluation for the q explicit stages per step. Since $r > q$, $r-q$ processors are idle during the evaluation of the explicit stages. In *improved* ABR we use these $r-q$ processors to improve the last $q-1$ explicit stages. To see in more detail how this can be done we present the following computation scheme.

matrix-vector computation	f-evaluations	#proc.
$\overline{Y}^{(0)} = (\overline{A}_0 \otimes I_{dd})Y_{n-1} + h(\overline{B}_0 \otimes I_{dd})F^*_{n-1}$ $\underline{Y}^{(0)} = (\underline{A}_0 \otimes I_{dd})Y_{n-1} + h(\underline{B}_0 \otimes I_{dd})F^*_{n-1}$		
	$F(\overline{Y}^{(0)})$ $F\begin{pmatrix} \underline{Y}^{(0)}_1 \\ \vdots \\ \underline{Y}^{(0)}_{r-q} \end{pmatrix}$	q $r-q$
$\overline{Y}^{(1)} = (\overline{A} \otimes I_{dd})Y_{n-1} + h(\overline{B} \otimes I_{dd})F^*_{n-1} +$ $h(\overline{C}_1 \otimes I_{dd})F(\overline{Y}^{(0)})$		
	$F\begin{pmatrix} \overline{Y}^{(1)}_{2q-r+1} \\ \vdots \\ \overline{Y}^{(1)}_{q} \end{pmatrix}$ $F\begin{pmatrix} \underline{Y}^{(0)}_{r-q+1} \\ \vdots \\ \underline{Y}^{(0)}_{r} \end{pmatrix}$	$r-q$ q
$\underline{Y}^{(1)} = (\underline{A} \otimes I_{dd})Y_{n-1} + h(\underline{B} \otimes I_{dd})F^*_{n-1} +$ $h(\underline{C}_1 \otimes I_{dd})F(\overline{Y}^{(1)}) + h(\underline{C}_2 \otimes I_{dd})F(\underline{Y}^{(0)})$		
	$F(\underline{Y}^{(1)})$	r
$\underline{Y}^{(2)} = (\underline{A} \otimes I_{dd})Y_{n-1} + h(\underline{B} \otimes I_{dd})F^*_{n-1} +$ $h(\underline{C}_1 \otimes I_{dd})F(\overline{Y}^{(1)}) + h(\underline{C}_2 \otimes I_{dd})F(\underline{Y}^{(1)})$		
	$F(\underline{Y}^{(2)})$	r
$\vdots$	$\vdots$	$\vdots$
$\underline{Y}^{(m-1)} = (\underline{A} \otimes I_{dd})Y_{n-1} + h(\underline{B} \otimes I_{dd})F^*_{n-1} +$ $h(\underline{C}_1 \otimes I_{dd})F(\overline{Y}^{(1)}) + h(\underline{C}_2 \otimes I_{dd})F(\underline{Y}^{(m-2)})$		
	$F(\underline{Y}^{(m-1)})$	r
$\underline{Y}^{(m)} = (\underline{A} \otimes I_{dd})Y_{n-1} + h(\underline{B} \otimes I_{dd})F^*_{n-1} +$ $h(\underline{C}_1 \otimes I_{dd})F(\overline{Y}^{(1)}) + h(\underline{C}_2 \otimes I_{dd})F(\underline{Y}^{(m-1)})$		
$Y_n = \begin{pmatrix} \overline{Y}^{(1)} \\ \underline{Y}^{(m)} \end{pmatrix}$ $F^*_n = \begin{pmatrix} F(\overline{Y}^{(1)}) \\ F(\underline{Y}^{(m-1)}) \end{pmatrix}$		
Total number of f-evaluations on r processors		$m+1$

The scheme shows which computations have to be done, categorized by matrix-vector computations (column 1) and f-evaluations (column 2). The third column denotes the number of processors that are involved in the corresponding f-evaluation.

The several symbols have the following meaning:

- $\overline{A}_0$ and $\overline{B}_0$ are $q \times s$ matrices defining a slightly modified AB predictor for the first q stages: the last $q-1$ rows are the same as in AB, but the first stage is given order $s+1$ by filling the $(s-1)^{th}$ element in the first row of $\overline{A}_0$ by order conditions as well.
- $\underline{A}_0$ and $\underline{B}_0$ (both $r \times s$ matrices) are the lower parts of the AB predictor.
- $\overline{A}$, $\overline{B}$ ($q \times s$ matrices) and $\overline{C}_1$ (a $q \times q$ matrix) define a correction formula of order $s+1$ for the first q stages: $\overline{A} = \overline{A}_0$, $\overline{B}$ and the last $q-1$ components of the first column of $\overline{C}_1$ are determined by order conditions. The remaining components in $\overline{C}_1$ are 0.
- $\underline{A}$, $\underline{B}$ ($r \times s$ matrices), $\underline{C}_1$ and $\underline{C}_2$ (an $r \times q$ and $r \times r$ matrix, respectively) correspond to a correction formula of order $s+1$ by defining the last column of $\underline{A}$ and $\underline{B}$ and the whole $\underline{C}_i$ ($i \in \{1,2\}$) by order conditions. The remaining parts of $\underline{A}$ and $\underline{B}$ are 0.
- $\overline{Y}_i^{(1)}$ and $\underline{Y}_i^{(0)}$ denote the i^{th} components of $\overline{Y}^{(1)}$ and $\underline{Y}^{(0)}$ respectively.

During the evaluation of the q components of $\overline{Y}^{(0)}$ we already evaluate the first $r-q$ components of the prediction $\underline{Y}^{(0)}$. Then we improve the last $q-1$ components of $\overline{Y}^{(0)}$ by means of the first column of $\overline{C}_1$, which results in $\overline{Y}^{(1)}$. Next we evaluate the remaining part of $\underline{Y}^{(0)}$. Now $r-q$ processors are available for the evaluation of $\overline{Y}^{(1)}$. Since we only benefit from the improvements in $\overline{Y}^{(1)}$ if we are able to evaluate all $q-1$ improved stages in $\overline{Y}^{(1)}$, we need to put a constraint on the size of q and r: $q-1 \leq r-q$. Remembering the first restriction for q and r (that is, $q < r$), we conclude that for *improved* ABR q and r have to satisfy

$$q \leq \min\{r-1, \frac{1}{2}(r+1)\}. \tag{5.7}$$

Note that (5.7) holds for all the correctors in Table 2. The rest of the scheme is analogous to the ABR case.

From the scheme it can be seen that the first q stages are solved in PEC-mode. Numerical experiments show that just one single correction is indeed enough to solve these implicit equations.

6. Numerical Experiments

The numerical experiments were performed using 15-digits arithmetic. The accuracies obtained are given by the number of correct digits Δ, defined by writing the maximum norm of the absolute error at the endpoint in the form $10^{-\Delta}$.

We took for $s = 7$ and $r = 5$, the 5-processor methods ABR (of order 8) and *improved* ABR (of order 9). We equipped both methods with the same dynamic iteration strategy (with a slightly more stringent stopping criterion) as in [4].

Two well-known test problems were taken from [2], namely the the Euler problem

$$\begin{array}{rcrcl} y_1' &=& y_2y_3 &,& y_1(0)=0, \\ y_2' &=& -y_1y_3 &,& y_2(0)=1, \quad 0 \leq t \leq 20, \\ y_3' &=& -0.51y_1y_2 &,& y_3(0)=1, \end{array} \tag{6.8}$$

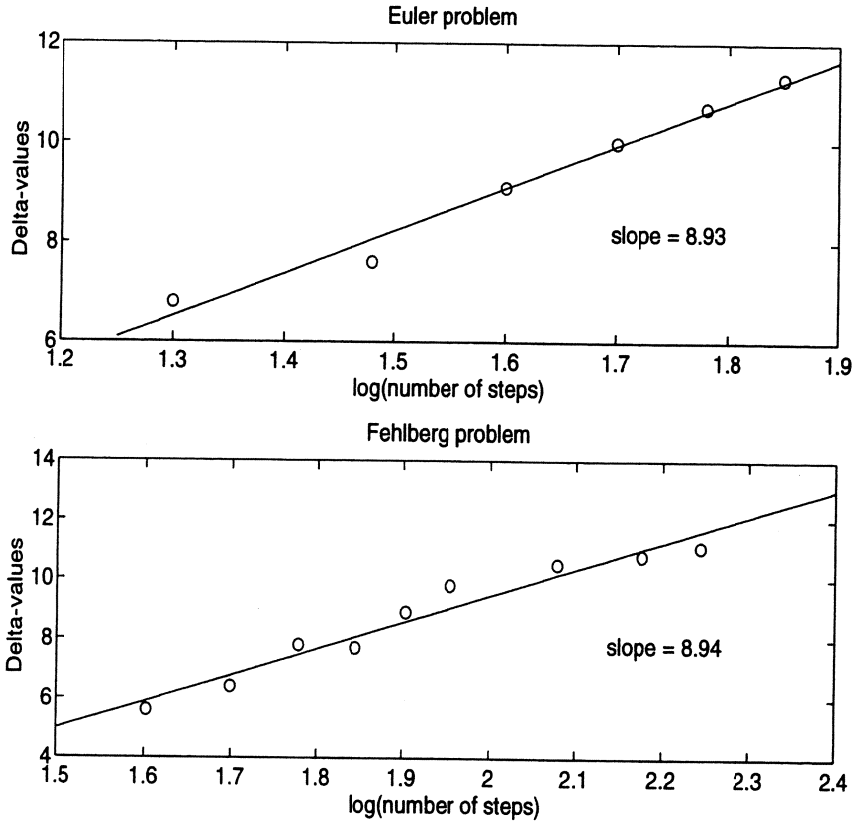

Figure 1: Observed order of *improved* ABR ($s = 2 + 5$)

and the Fehlberg problem

$$\begin{aligned} y_1' &= 2ty_1\log(\max\{y_2, 10^{-3}\}) \quad , \quad y_1(0) = 1, \\ y_2' &= -2ty_2\log(\max\{y_1, 10^{-3}\}) \quad , \quad y_2(0) = e, \end{aligned} \qquad 0 \le t \le 5. \tag{6.9}$$

First we investigate to what extent the omission of the step point formula and the PEC-mode for solving the first q stages affect the observed order of *improved* ABR. Therefore we plot the Δ-values against the $^{10}\log$(number of steps). These points should lie on a straight line whose slope equals the step point order. Figure 1 shows that the expected value 9 is fairly well approximated.

Next we compare the performance of *improved* ABR with that of ABR. For completeness, we also listed the performance of the DOPRI8 code with automatic step-size control by Hairer, Nørsett & Wanner [2]. Table 3 shows that *improved* ABR works about 20 % more efficiently than ABR, while the averaged speed-up factor of *improved* ABR compared to DOPRI8 (to be considered as one of the best sequential codes) is about 3.1.

7. Concluding Remarks

The attempt to improve the parallel Adams-Bashforth-Radau (ABR) methods proposed in [4] has resulted in a more efficient code. More particularly, on 5 processors, the speed-up of the improved version compared to the fully automatic code DOPRI8 is about 3.1. This speed-up could be further improved by including a step-size strategy.

Table 3: Comparison of *improved* ABR with ABR and DOPRI8

# f-evaluations for the Euler problem						
Δ-values	6	7	8	9	10	11
DOPRI8	415	576	728	898	1133	1422
ABR(s=2+5)	160	192	223	293	379	506
Improved ABR(s=2+5)	117	169	221	273	325	377

# f-evaluations for the Fehlberg problem						
Δ-values	6	7	8	9	10	11
DOPRI8	759	963	1227	1574	1990	2503
ABR(s=2+5)	335	430	532	689	846	1067
Improved ABR(s=2+5)	256	361	466	571	677	782

ACKNOWLEDGEMENTS

The author is very grateful to prof. dr. P.J. van der Houwen and dr. B.P. Sommeijer for their help during the preparation of this paper.

REFERENCES

[1] J.C. Butcher (1987): The numerical analysis of ordinary differential equations, Runge-Kutta and general linear methods, Wiley, New York.

[2] E. Hairer, S.P. Nørsett & G. Wanner (1987): Solving ordinary differential equations I. Non-stiff problems, Springer Series in Comp. Math., vol.8, Springer-Verlag, Berlin.

[3] P.J. van der Houwen & B.P. Sommeijer (1992): Block Runge-Kutta methods on parallel computers, Z. angew. Math. Mech. 72, 3-18.

[4] P.J. van der Houwen, B.P. Sommeijer & J.J.B. de Swart (1994): Parallel Predictor-Corrector Methods, CWI, Report NM-R9408, submitted for publication.