Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2010, Article ID 765620, 19 pages
doi:10.1155/2010/765620

Research Article

Implementing Adams Methods with Preassigned Stepsize Ratios

David J. López[1] and José G. Romay[2]

[1] *Departamento Matemática Aplicada y Estadística, Universidad Politécnica de Cartagena, 30202 Cartagena, Spain*
[2] *Departamento de Matemática y Computación, Universidad del Zulia, Maracaibo 4001, Venezuela*

Correspondence should be addressed to David J. López, david.lopez@upct.es

Received 1 October 2009; Accepted 13 January 2010

Academic Editor: Angelo Luongo

Runge-Kutta and Adams methods are the most popular codes to solve numerically nonstiff ODEs. The Adams methods are useful to reduce the number of function calls, but they usually require more CPU time than the Runge-Kutta methods. In this work we develop a numerical study of a variable step length Adams implementation, which can only take preassigned step-size ratios. Our aim is the reduction of the CPU time of the code by means of the precalculation of some coefficients. We present several numerical tests that show the behaviour of the proposed implementation.

1. The Variable-Stepsize Adams Method

The Adams methods (Bashforth and Adams [1], Moulton [2]) are the most popular multistep formulas to solve a non-stiff initial value problem

$$y'(x) = f\big(x, y(x)\big), \qquad y(x_0) = y_0. \tag{1.1}$$

These codes, and other interesting multistep methods, are based on the replacement of $f(x, y(x))$ by an interpolating polynomial $p(x)$, and the approximation

$$y(x+h) = y(x) + \int_x^{x+h} f\big(t, y(t)\big)dt \simeq y(x) + \int_x^{x+h} p(t)dt. \tag{1.2}$$

When the stepsize is constant, the explicit k-step Adams-Bashforth method can be written in terms of the backward differences of f,

$$y_{n+1} = y_n + h\sum_{j=0}^{k-1}\gamma_j^*\nabla^j f_n, \tag{1.3}$$

where the coefficients γ_j^* are obtained by the recurrence

$$\gamma_0^* = 1, \qquad \gamma_j^* = 1 - \sum_{i=1}^{j}\frac{\gamma_{j-i}^*}{i+1}. \tag{1.4}$$

The formulation of the implicit k-step Adams-Moulton is straightforward, as can be seen, for example, by Lambert in [3, Section 3.9]. The implicit equation is usually solved using a *PECE* fixed-point implementation [3, page 144], starting from an Adams-Bashforth formula. The order of the k-step ABM pair is $k + 1$ (an exhaustive study of the order of a predictor-corrector pair can be seen in [3, Section 4.2]), and it is equivalent to the k-order pair based on k-step Adams-Bashforth and $(k - 1)$-step Adams-Moulton with local extrapolation [3, pages 114-115].

The γ_j^* coefficients do not depend on the step length, so they remain constant in the whole integration and need only be computed once. However, they do vary when the stepsize $h_n = x_{n+1} - x_n$ is not constant, so they must be recalculated in each step of the integration.

There are lots of different ways to implement a variable-stepsize variable-order Adams code (actually one different for each developer). For example, codes as DEABM [4], VODE [5] and LSODE [6, 7] include some controls to diminish the computational cost in some situations by keeping the stepsize constant, or by avoiding the growth of the order. Nowadays, symbolic programs such as Matlab and Mathematica as well as their numerical solvers of ODEs `ode113` [8] and `NDSolve` [9, pages 961–972,1068,1215-1216], are very popular. We do not want to restrict our study to a particular code, so we will use in the tests a *purer* implementation of the variable-stepsize Adams code, developed by ourselves, based on the Krogh divided differences formulation [10].

Krogh developed a way to compute the Adams formulas in terms of modified divided differences. Following the notation of Hairer et al. [11, pages 397–400], the variable-stepsize explicit k-step Adams-Bashforth predictor is

$$p_{n+1} = y_n + h_n\sum_{j=0}^{k-1}g_j(n)\beta_j(n)\phi_j(n), \tag{1.5}$$

and the k-step Adams-Moulton corrector can be expressed as

$$y_{n+1} = p_{n+1} + h_n g_k(n)\phi_k(n+1). \tag{1.6}$$

The coefficients can be calculated easily by the recurrences

$$\beta_0(n) = 1, \qquad \beta_j(n) = \beta_{j-1}(n)\frac{x_{n+1} - x_{n-j+1}}{x_n - x_{n-j}}, \tag{1.7}$$

$$\phi_0(n) = f_n, \qquad \phi_j(n) = \phi_{j-1}(n) - \beta_{j-1}(n-1)\phi_{j-1}(n-1). \tag{1.8}$$

The calculation of $g_j(n)$ requires a double loop in terms of j and q:

$$\begin{aligned} c_{0,q}(x_{n+1}) &= \frac{1}{q}, \\ c_{j,q}(x_{n+1}) &= c_{j-1,q}(x_{n+1}) - c_{j-1,q+1}(x_{n+1})\frac{h_n}{x_{n+1} - x_{n-j+1}}, \\ g_j(n) &= c_{j,1}(x_{n+1}). \end{aligned} \tag{1.9}$$

The computational cost of the recalculation of $g_j(n)$ in each step is the major disadvantage of the variable-stepsize Adams code when the cost is measured in CPU time.

2. The Stepsize Ratios

The coefficients "$\beta(n)$" and "$g(n)$" are scalar and they are independent of the function f. Let us denote the ratios

$$r_i = \frac{h_{i+1}}{h_i}. \tag{2.1}$$

Then it is well known that dependence on the points x_i in (1.7) and (1.9) is actually a dependence only on the ratios r_i.

2.1. The Coefficients in Terms of the Stepsize Ratios

Let us study the number of ratios needed for each coefficient.

Proposition 2.1. *For $j \geq 1$ the coefficient $\beta_j(n)$ only depends on the j ratios $r_{n-1}, r_{n-2}, \ldots, r_{n-j}$, and for $j \geq 2$ the coefficient $g_j(n)$ only depends on the $j-1$ ratios $r_{n-1}, r_{n-2}, \ldots, r_{n-j+1}$.*

Proof. We only need to apply the relation (with $p > m$)

$$\begin{aligned} x_p - x_m &= \sum_{j=0}^{p-m-1} h_{m+j} = \sum_{j=0}^{p-m-1} h_m \prod_{i=0}^{j-1} r_{m+i} \\ &= h_m \cdot \varphi_{p-m}(r_{p-2}, r_{p-3}, \ldots, r_m) \end{aligned} \tag{2.2}$$

to deduce that the fraction in (1.7) can be rewritten as

$$\begin{aligned}\frac{x_{n+1}-x_{n-j+1}}{x_n-x_{n-j}} &= \frac{h_{n-j+1}\cdot\varphi_j\left(r_{n-1},r_{n-2},\ldots,r_{n-j+1}\right)}{h_{n-j}\cdot\varphi_j\left(r_{n-2},r_{n-3},\ldots,r_{n-j}\right)}\\ &= \frac{h_{n-j}\cdot r_{n-j}\cdot\varphi_j\left(r_{n-1},r_{n-2},\ldots,r_{n-j+1}\right)}{h_{n-j}\cdot\varphi_j\left(r_{n-2},r_{n-3},\ldots,r_{n-j}\right)}\\ &= \psi_j\left(r_{n-1},r_{n-2},\ldots,r_{n-j}\right)\end{aligned} \tag{2.3}$$

and only depends on the j ratios $r_{n-1},r_{n-2},\ldots,r_{n-j}$. Using an inductive reasoning, starting with the constant value $\beta_0(n)=1$, it follows that if $j\geq 1$, then

$$\beta_j(n)=\Psi_j\left(r_{n-1},r_{n-2},\ldots,r_{n-j}\right). \tag{2.4}$$

A similar proof can be developed for $g_j(n)$. Applying (2.2) to the fraction in (1.9),

$$\begin{aligned}\frac{h_n}{x_{n+1}-x_{n-j+1}} &= \frac{h_{n-j+1}\cdot\prod_{i=0}^{j-2}r_{n-j+i+1}}{h_{n-j+1}\cdot\varphi_j\left(r_{n-1},r_{n-2},\ldots,r_{n-j+1}\right)}\\ &= \frac{\prod_{i=0}^{j-2}r_{n-j+i+1}}{\varphi_j\left(r_{n-1},r_{n-2},\ldots,r_{n-j+1}\right)}\\ &= \theta_j\left(r_{n-1},r_{n-2},\ldots,r_{n-j+1}\right).\end{aligned} \tag{2.5}$$

From this equation and the starting values

$$c_{0,q}(x_{n+1})=\frac{1}{q},\qquad c_{1,q}(x_{n+1})=\frac{1}{q(q+1)}, \tag{2.6}$$

the inductive reasoning shows that, if $j\geq 2$,

$$c_{j,q}(x_{n+1})=\Theta_{j,q}\left(r_{n-1},r_{n-2},\ldots,r_{n-j+1}\right), \tag{2.7}$$

and so

$$g_j(n)=c_{j,1}(x_{n+1})=\Theta_{j,1}\left(r_{n-1},r_{n-2},\ldots,r_{n-j+1}\right), \tag{2.8}$$

and the proof is complete. □

2.2. The Number of Coefficients in Terms of the Number of Ratios

Let us fix only $\lambda\geq 2$ options to choose the ratios, that is,

$$r_i\in\{\omega_1,\omega_2,\ldots,\omega_\lambda\},\quad \forall i\geq 0. \tag{2.9}$$

As the ratios r_i belong to a finite subset, then the coefficients $\beta_j(n)$ and $g_j(n)$ belong to finite spaces of possible events. As there are only a finite number of "$\beta(n)$" and "$g(n)$" coefficients, they can be precalculated, and read at the start of the numerical integration. In particular, the major computational disadvantage of the variable-stepsize Adams code (the double loop in the computation of (1.9) in each integration step) is removed.

The importance of Proposition 2.1 is not the calculation of the functions Ψ_j and $\Theta_{j,1}$, but the existence of those functions and the number of the ratios for each one. Let us count the number of different possible coefficients in terms of λ and the order $k+1$. For simplicity we ignore the constants $\beta_0(n) = 1$, $g_0(n) = 1$, and $g_1(n) = 1/2$.

For the predictor (1.5) we require from $\beta_1(n)$ to $\beta_{k-1}(n)$. According to (1.8), the value $\beta_k(n)$ is not necessary for $\phi_k(n+1)$ in the corrector (1.6). From Proposition 2.1 it follows that there are only λ^j different values for $\beta_j(n)$, so the total length is

$$\lambda + \lambda^2 + \cdots + \lambda^{k-1} = \frac{\lambda^k - \lambda}{\lambda - 1}. \tag{2.10}$$

Again from Proposition 2.1, if $j \geq 2$, the coefficient $g_j(n)$ can only take λ^{j-1} different values. But in this case $g_k(n)$ is used in the implicit corrector (1.6), so it is necessary to compute $g_2(n), \ldots, g_k(n)$ and (2.10) is also valid for the "$g(n)$" coefficients. We can formulate the first lemma.

Lemma 2.2. *The "$\beta(n)$" and "$g(n)$" coefficients needed in the k-step pair* (1.5) *and* (1.6) *can be stored in two arrays of length* (2.10).

A variable-stepsize algorithm needs an estimator of the error to change the stepsize. This estimator can be computed in different ways. We follow Lambert's [3, section 4.4] and extend them to variable-stepsizes.

As the k-step pair is actually the k-order pair with local extrapolation, we can choose the same local error estimator for both pairs, which leads to

$$\mathrm{LE}_0(n) = h_n\big(g_k(n) - g_{k-1}(n)\big)\phi_k(n+1). \tag{2.11}$$

In practice the variable-stepsize Adams method is also endowed with the capability of changing its order, so it is necessary to obtain some local error estimators for decreasing or increasing the order. By modifying the previous equation we obtain the estimators

$$\begin{aligned} \mathrm{LE}_{-1}(n) &= h_n\big(g_{k-1}(n) - g_{k-2}(n)\big)\phi_{k-1}(n+1), \\ \mathrm{LE}_{+1}(n) &= h_n\big(g_{k+1}(n) - g_k(n)\big)\phi_{k+1}(n+1), \end{aligned} \tag{2.12}$$

for decreasing or increasing the order, respectively. The value of $\beta_k(n)$ is necessary for $\phi_{k+1}(n+1)$ in $\mathrm{LE}_{+1}(n)$, and also $g_{k+1}(n)$, so Lemma 2.2 can be updated to the following lemma.

Lemma 2.3. *The "$\beta(n)$" and "$g(n)$" coefficients needed for the predictor* (1.5)*, the corrector* (1.6)*, and the estimators* (2.11) *and* (2.12) *can be stored in two arrays of length*

$$\lambda + \lambda^2 + \cdots + \lambda^{k-1} + \lambda^k = \frac{\lambda^{k+1} - \lambda}{\lambda - 1}. \tag{2.13}$$

The CPU time used by the code will increase drastically if we let the order grow without control. For this reason it is necessary to impose a maximum order. In double precision (8 bytes per data, 15-16 decimal digits) the maximum order is usually $k + 1 = 13$ (Lambert [3, page 144]). For our purposes we let this maximum as a parameter MO (Maximum Order). We must emphasize that, when the maximum order $k + 1 = \mathrm{MO}$ is reached, the estimator $\mathrm{LE}_{+1}(n)$ is not necessary, and then we do not need to calculate (and store) $\beta_{\mathrm{MO}-1}(n)$ and $g_{\mathrm{MO}}(n)$. This fact produces the final lemma.

Lemma 2.4. *The "$\beta(n)$" and "$g(n)$" coefficients needed for a Variable-Step Variable-Order Adams k-step code, with order up to $k + 1 = MO$, can be stored in two arrays of length*

$$\lambda + \lambda^2 + \cdots + \lambda^{MO-2} = \frac{\lambda^{MO-1} - \lambda}{\lambda - 1}. \tag{2.14}$$

The amount of RAM needed to store and read the coefficients can be an important disadvantage of a *fixed ratios* implementation. In the following section we search some useful choices of λ and ω_m in (2.9).

3. The Fixed Ratios Strategy

Usually, the implementation of the variable-stepsize Adams method includes a restriction in the ratios of the kind of

$$\omega \le r_i \le \Omega \tag{3.1}$$

that appears in the stability of generic multistep variable-stepsize methods (Hairer et al. [11, page 403]) and in the convergence of the variable-stepsize variable-order Adams method (Shampine [12]). Then, the stepsizes ratio is obtained by adapting equation (391a) by Butcher in [13, page 293]:

$$r = \max\left(\omega, \min\left(\Omega, 0.9\left(\frac{\mathrm{tol}}{\mathrm{est}}\right)^{1/(k+1)}\right)\right). \tag{3.2}$$

The fixed ratios implementation that we propose selects the ratio as the largest ω_m in (2.9) lower to r in (3.2):

$$r_i = \max\{\omega_m \le r\}. \tag{3.3}$$

The pathologic case $\omega_m > r$ for $1 \le m \le \lambda$ can be easily avoided by including $\omega_1 = \omega$ as the minimum of the prefixed ratios.

For the numerical study we programmed a code, denoted as VSVOABM, with a classical implementation. VSVOABM uses (3.2) for the stepsize selection, and lets the order vary freely up to $k+1 = 13$. In addition, we denote by FRABM (fixed ratios Adams-Bashforth-Moulton) a code based on VSVOABM, changing (3.2) by (3.3) to select the new stepsize. As the prefixed ratio r_i in (3.3) is lower than r in (3.2), the stepsize selected in FRABM will be

smaller than the one that VSVOABM would choose from the same estimator, so the error of FRABM is expected to be lower (of course using more steps). As the coefficients "$\beta(n)$" and "$g(n)$" do not depend on the problem, they were precalculated only once, stored in a file and put in RAM at the start of the integration in FRABM. Then, in each step, they are read from memory instead of being calculated. As FRABM was programmed from VSVOABM, the comparison of the two will show the behaviour of the classical strategy *let the ratios free and calculate the coefficients in each step* versus the new *fix a finite number of ratios and read the coefficients from RAM in each step*.

The authors presented in [14] a large number of experiments. In this paper we summarize them. First of all we tested the following problems. In all of them the variation of the stepsize is crucial.

(1) Five revolutions of the classical *Two-Body* problem used in many relevant books, such as Butcher [13, pages 4–7], Dormand [15, pages 86–90], or Shampine [16, pages 90–95]. The variation of the stepsize grows with the eccentricity of the ellipsis. We test a pair of different problems, one with middle eccentricity $e = 0.6$ and another one with high eccentricity $e = 0.9$.

(2) An *Arenstorf* problem [17] with the popular orbit printed by Butcher in [13, page 9]. of Hairer et al. [11, page 130, Figure 0.1] shows that a variable-stepsize strategy is very important in this problem.

(3) The *Lorenz* problem in the interval $[0, 16]$, extremely sensitive to errors in the first steps [11, pages 120, 245].

(4) A fictitious seven-body problem, denoted *Pleiades* [11, pages 245-246]. In this problem the function $f(x, y)$ is expensive to evaluate.

In the figures we compare the decimal logarithm of the *CPU time* versus the decimal logarithm of the error, with tolerances from 10^{-3} to 10^{-13}, except in the Lorenz problem in which tolerances were restricted from 10^{-7} to 10^{-13} (see Hairer et al. [11, page 245] for an explanation). Nowadays most of these problems can be solved quickly in a standard computer, so CPU time was measured with extreme precision in "uclocks" (about 840 nanoseconds, less than 1 microsecond).

The main conclusions were the following.

(i) Ratios ω_m must be close to 1, even if the stepsize may vary a lot in the integration (Section 3.1).

(ii) Coefficients "$\beta(n)$" must be calculated from (1.7) in each step and coefficients "$g(n)$" must be precalculated once and read in each step (Section 3.2).

(iii) FRABM methods with $\lambda = 5$ ratios, $r_i \in \{0.5, 0.9, 1, 1.1, 2\}$, and maximum orders MO = 10 and MO = 12 are two good methods with very different RAM requirements (Section 3.3).

3.1. The Ratios in the Classical Implementation

We present some results of five revolutions of a highly eccentric orbit $e = 0.9$ of the Two-Body problem with VSVOABM. In Figure 1 we see significant changes in the stepsize. In the beginning of a revolution (perigee) it is about one hundred times less than in the middle of a revolution (apogee). However, the ratios in Figure 2 remain near to 1 everywhere, even those less than 0.9 from the rejected steps.

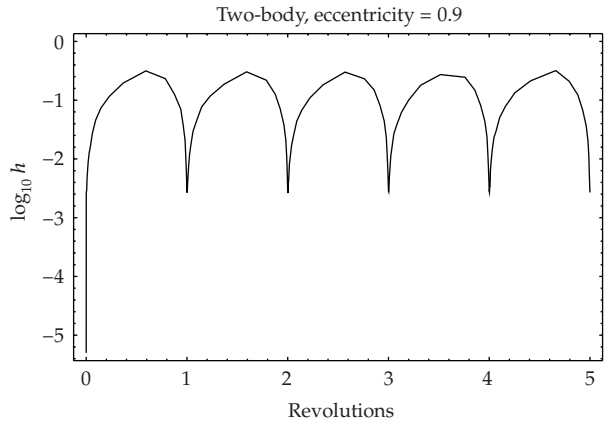

Figure 1: Decimal logarithm of the stepsizes in a highly eccentric two-body problem.

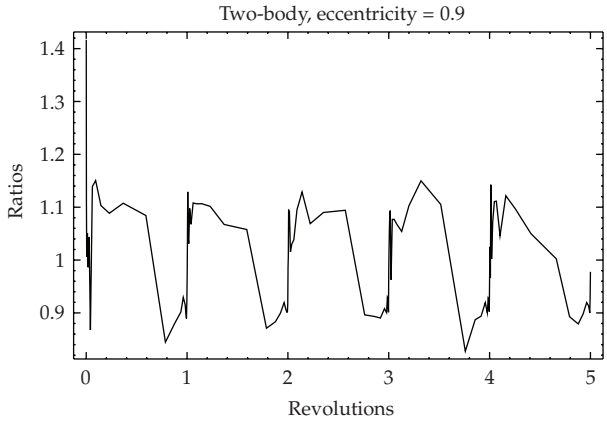

Figure 2: Ratios in a highly eccentric two-body problem.

We have tested some other eccentricities in the Two-Body problem. As the eccentricity reduces to 0, the changes in the stepsizes in Figure 1 diminish. In any case, the structure of Figure 2 shows little change.

In Table 1 we order the ratios and select the percentiles 25, 50 (the median), and 75 of the accepted steps for different tolerances. The percentiles approach to unity as the tolerance reduces, probably because the selected orders increase with small tolerances (see in Shampine [16, page 262, Figure 5.3]). In Table 2 we show the behaviour of the ratios in a global sample obtained with tolerances from 10^{-3} to 10^{-13} for all the proposed problems.

We must conclude that a good choice of a finite number of prefixed ratios must select them near to 1, even for problems with big changes in the stepsize.

3.2. The Difference between the "$\beta(n)$" and the "$g(n)$" Coefficients

The coefficients "$\beta(n)$" and "$g(n)$" are archived in two arrays of the same size, so the CPU time required when we *search* for a particular $\beta_j(n)$ or $g_j(n)$ must be the same. However, there

Table 1: Percentiles 25, 50, and 75 of the ratios for the highly eccentric $e = 0.9$ two-body problem.

Tolerance	$P25$	$P50$	$P75$
10^{-3}	0.94	1.04	1.17
10^{-4}	0.96	1.11	1.14
10^{-5}	0.95	1.07	1.11
10^{-6}	0.93	1.04	1.09
10^{-7}	0.93	1.03	1.07
10^{-8}	0.94	1.02	1.06
10^{-9}	0.95	1.02	1.04
10^{-10}	0.95	1.02	1.04
10^{-11}	0.96	1.01	1.03
10^{-12}	0.97	1.01	1.03
10^{-13}	0.97	1.01	1.02

Table 2: Relevant percentiles of the ratios of accepted steps in a global sample.

$P10$	$P20$	$P25$	$P30$	$P33$	$P40$	$P50$	$P60$	$P66$	$P70$	$P75$	$P80$	$P90$
0.94	0.96	0.97	0.98	0.98	1.00	1.01	1.02	1.03	1.03	1.04	1.04	1.09

is an important difference when we *compute* them from their algorithms, because while $\beta_j(n)$ is calculated by means of the single loop (1.7), a double loop is required in (1.9) for $g_j(n)$.

We present different FRABM integrations for the Arenstorf problem, all of them with $\lambda = 4$ prefixed ratios, $r_i \in \{0.5, 0.9, 1.1, 2\}$, and maximum order MO = 13. We change the way to evaluate (*computing* or *reading*) the ("$\beta(n)$"/"$g(n)$") coefficients, which produces four different implementation modes. In all of them the coefficients must be the same, and also the numerical results and errors (round-off errors can produce in (3.2) slightly different estimators est and different selections of the ratio r_i in (3.3), but it does not happen often). As the CPU time can be different for the four FRABM methods, the comparison must provide parallel graphs.

In all the tests we made, both *reading* "$g(n)$" strategies (diamonds and squares in Figure 3) won to both *computing* "$g(n)$" strategies (stars and triangles). In Figure 3 the best implementation mode resulted to be *reading* "$\beta(n)$" and "$g(n)$". In other problems we tested *computing* "$\beta(n)$" and *reading* "$g(n)$" was slightly superior. Taking into account all the tests, both strategies were virtually tied, and they were doubtlessly more efficient than both *computing* "$g(n)$" modes. But as storing "$\beta(n)$" requires an important amount of RAM, we discard the *reading* mode for "$\beta(n)$", so from now on the FRABM implementation will compute "$\beta(n)$" from (1.7) in each step, and only precalculate, store, and read in each step "$g(n)$". In this way RAM requirements are halved.

3.3. Proposing Some FRABM Algorithms

Now we will develop some concrete FRABM algorithms. It is impossible to reproduce all of the experiments we made in [14] comparing sets of ratios obtained by statistical and intuitive processes, so in this paper we only present the final conclusions.

First of all, we looked for the minimum and maximum ratios ω and Ω. These ratios will not be used often, but they are indispensable. The minimum $\omega_1 = \omega < 0.9$ is required for the rejected steps, and the maximum $\omega_\lambda = \Omega$ will be used specially in the first steps,

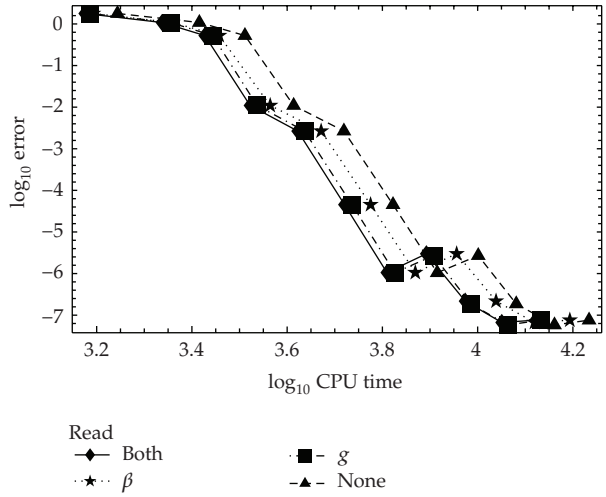

Figure 3: FRABM, computing versus reading "$\beta(n)$" and "$g(n)$" in the Arenstorf problem.

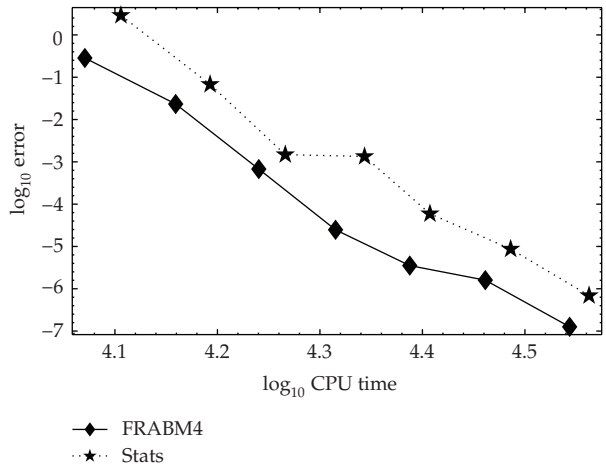

Figure 4: Two different $\lambda = 4$, MO = 13 FRABM methods in the Lorenz problem.

as the first stepsize h_0 must be very small because the code starts with the minimum order $k + 1 = 2$. Computing (and not reading) the "$g(n)$" coefficients directly from (1.9) only in these special cases looks like an interesting idea, because it sets two preassigned ratios free. However, we must note that using one nonpreassigned ratio $r_i \notin \{\omega_1, \omega_2, \ldots, \omega_\lambda\}$ forces the code to compute the next k "$g(n)$" coefficients from (1.9) to eliminate r_i from (2.8), and the computational cost will be increased.

In our experiments there were no clear winners, but maybe the pair $\omega = 0.5$ and $\Omega = 2$ was more *robust* than other compared ratios, so we assumed the criterion of Dormand [15, page 73] and Butcher [13, page 293] and selected this pair.

With the addition of the ratio $\omega_2 = 1$ we obtained the basic doubling and halving FRABM3 method with $\lambda = 3$ prefixed ratios, $r_i \in \{0.5, 1, 2\}$. Lemma 2.4 shows that for

Table 3: RAM cost in terms of the number of prefixed ratios λ for maximum order $MO = 11$.

λ	Number of coefficients	RAM cost
3	29523	0.23 MB
4	349524	2.67 MB
5	2441405	18.63 MB
6	12093234	92.26 MB

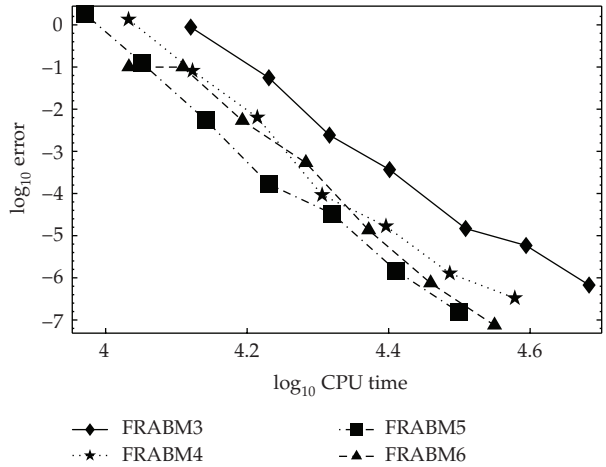

Figure 5: FRABMλ with $3 \leq \lambda \leq 6$, MO = 11 in the Lorenz problem.

maximum order MO = 13 this method only requires 265719 "$g(n)$" coefficients and 2.03 MB of RAM to store them in double precision.

In the vast majority of occasions the stepsize is accepted, not rejected. That is why we decided to keep $\omega_1 = 0.5$ as the unique prefixed ratio for rejections, and looked for $\omega_m \geq 0.9$ if $m \geq 2$. In fact, taking into account the numerical results of our tests, for $\lambda > 3$ we decided to fix $\omega_2 = 0.9$, and $\omega_{\lambda-1} = 1.1$ as its companion. In particular, the method with $\lambda = 4$ ratios, $r_i \in \{0.5, 0.9, 1.1, 2\}$ (from now on denoted FRABM4, diamonds in Figure 4), performed significantly better than the one with $r_i \in \{0.5, 0.98, 1.03, 2\}$ obtained with percentiles 33 and 66 in Table 2 (stars in Figure 4).

We tested several different strategies for $\lambda = 5$ ratios, including the one with percentiles 25, 50, and 75 from Table 2. However, the best strategy turned out to be the simplest: adding $\omega_3 = 1$ to the ratios in FRABM4. Then for FRABM5 we selected $r_i \in \{0.5, 0.9, 1, 1.1, 2\}$. The methods with ratios obtained by means of a *statistical* criterion from Table 2 performed clearly worse than others selected *intuitively* in all the tests we made.

For no specific reason, we add the ratio 1.05 to develop a $\lambda = 6$ method, with $r_i \in \{0.5, 0.9, 1, 1.05, 1.1, 2\}$. In Figure 5 we compared the behaviour of FRABM methods with $3 \leq \lambda \leq 6$ ratios, all of them with maximum order MO = 11 (the RAM requirements of the methods are in Table 3).

In Figure 5, and in all the problems we tested, FRABM5 was the best of the compared methods. We cannot assure that $\lambda = 5$ ratios are a better choice than $\lambda = 6$; we assure that in all of our tests the combination $\lambda = 5$, $r_i \in \{0.5, 0.9, 1, 1.1, 2\}$ performed better than $\lambda = 6$, $r_i \in \{0.5, 0.9, 1, 1.05, 1.1, 2\}$. This situation did not change when we raised λ. We compared methods with $3 \leq \lambda \leq 10$ ratios, all of them with the same MO = 9 (reduced through the fault

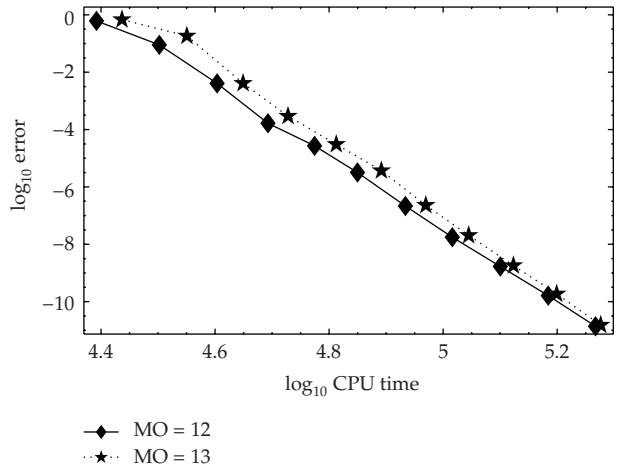

Figure 6: FRABM5, maximum orders MO = 12 and MO = 13 in the Pleiades problem.

Table 4: RAM cost in terms of the number of prefixed ratios λ and the maximum order MO.

λ	MO	RAM	Category
3	13	2.03 MB	Low cost
4	11	2.67 MB	Low cost
5	10	3.73 MB	Low cost
4	12	10.67 MB	Middle cost
5	11	18.63 MB	Middle cost
4	13	42.67 MB	High cost
5	12	93.13 MB	High cost

of the RAM cost for $\lambda = 10$), and FRABM5 was always the winner. It is surprising because FRABM5, MO = 9 only requires an amount of 0.75 MB of RAM.

We cannot conclude yet that the best method is FRABM5, because when $\lambda < 5$ bigger maximum orders MO are attainable with lower RAM cost. However, it appears that when $\lambda > 5$ the improvement of the error is less than the growth of the CPU time (and of course the RAM requirements), so we restrict our study to $3 \leq \lambda \leq 5$.

We divided the methods into three categories in terms of their RAM requirements (see Table 4), with the usual restriction MO ≤ 13 for $\lambda = 3, 4$, and MO = 12 for FRABM5. As the case $\lambda = 3$ reaches the maximum order MO = 13 with low-cost, FRABM3 was considered in the three categories.

We rejected the case MO = 13 for FRABM5 because it needs the enormous amount of 465.66 MB of RAM. Besides, the RAM requirements worked against the efficacy of the code. Figure 6 shows the comparative of FRABM5 methods with maximum orders MO = 12 and MO = 13 in the Pleiades problem. In all the problems under consideration the case MO = 13 was not competitive with MO = 12, specially for large tolerances.

The behaviour of the methods shown in Figure 6 is an exception. The efficacy of FRABMλ methods in Table 4 was consistent with its maximum order, that is, it was better with higher MO. However the *middle cost* methods were between *low-cost* and *high-cost* methods for all the problems in almost every tolerance, so we discarded them because they were not the winners in any circumstance.

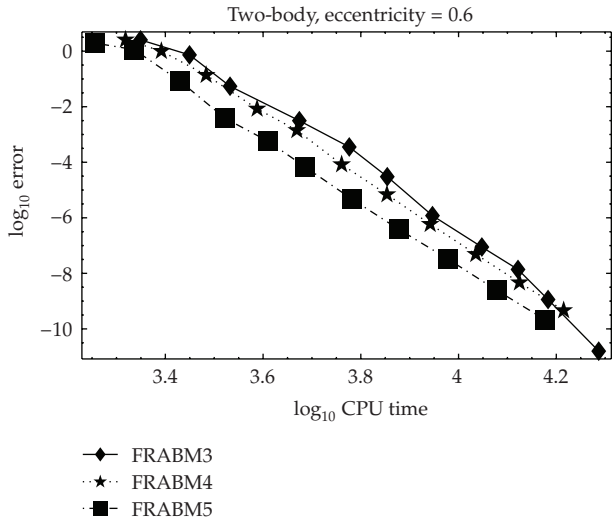

Figure 7: Low cost, FRABM3 with MO = 13, FRABM4 with MO = 11 and FRABM5 with MO = 10 in a middle eccentric Two-Body problem.

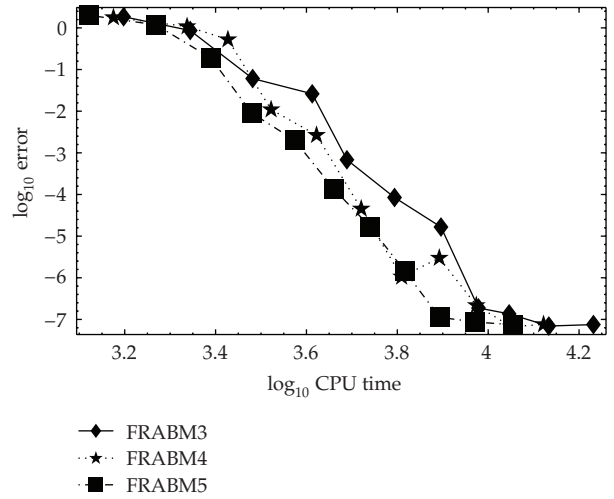

Figure 8: High cost, FRABM3 with MO = 13, FRABM4 with MO = 13, and FRABM5 with MO = 12 in the Arenstorf problem.

Figures 7 and 8 show representative examples of the performance of the methods in all the problems under consideration. FRABM3 with MO = 13 was clearly the poorest of the compared methods in both categories. For low- and high-cost FRABM5, MO = 10, 12 won FRABM4, MO = 11, 13, respectively. Thus, combinations of $\lambda = 5$ prefixed ratios $\{0.5, 0.9, 1, 1.1, 2\}$ with maximum orders MO = 10 and MO = 12 are *officially* chosen as our two FRABM candidates to solve general non-stiff ODEs.

4. The Behaviour of FRABM5

In this section we compare the pair of selected FRABM5 methods (green line with stars for MO = 10, red line with diamonds for MO = 12) and the classical implementation VSVOABM with maximum order MO = 13 (blue line with squares). As the most popular representative of Runge-Kutta methods, we include in the tests the version of the Dormand and Prince DOPRI853 code [18] downloaded from E. Hairer's website (black line with triangles). Figure 9 shows the efficacy of these four methods when the computational cost is measured in CPU time. We also present in Figure 10 the efficacy graphics with the computational cost measured in number of function calls.

Let us interpret the figures. At a point where a pair of methods has the same y-coordinate (*error*) the method with graph α units to the left in the x-coordinate gains $100 \cdot (1 - 10^{-\alpha})\%$ in the cost.

Taking into account only the Adams methods and *CPU time*, both FRABM5 codes clearly beat VSVOABM in both Two-Body, Arenstorf, Lorenz and Pleiades problems. The difference fluctuates between 0.1–0.15 units in the x-coordinate (a gain of 20%–30% in CPU time). Let us remark that FRABM5, MO = 10 requires the insignificant amount of 3.73 MB of RAM and is the most efficient multistep method for large tolerances. The case MO = 12 is more efficient for small tolerances, but its RAM cost grows to 93.13 MB.

When comparing the *number of function calls*, VSVOABM does not start very well in large tolerances, but it can take advantage of its superior maximum order MO = 13 to win FRABM5, MO = 10 in small tolerances. In the Two-Body problem with $e = 0.9$ the case MO = 12 is slightly better than VSVOABM even in these small tolerances. In the rest of the tests VSVOABM is the winner. We made some experiments (not included in this paper) to compare the efficacy in number of function calls of VSVOABM with FRABM4 and FRABM5, all of them with MO = 13. There was not a clear winner in our tests, which allows us to assert that the *main variable* that produces differences among the Adams methods is not the implementation mode, but the maximum order MO. This can be seen in the figures of Two-Body with $e = 0.6$ and Pleiades.

It is well known that Runge-Kutta methods, like DOPRI853, can hardly compete with multistep methods when comparing the number of evaluations. This can clearly be seen in all of the experiments. However, as Runge-Kutta methods do not need to recalculate their coefficients in each step, they are very superior in CPU time unless the function $f(x, y)$ is *expensive to evaluate*. DOPRI853 shows its potential in both Two-Body, Arenstorf,and Lorenz problems, all of them with a *cheap* function $f(x, y)$. The difference between DOPRI853 and VSVOABM is about 0.4–0.6 units in the x-axis, so the gain of DOPRI853 oscillates between 60%–75%, which means that it is about 2.5–4 times faster than VSVOABM. It is true that the *fixed ratios* strategy improves the efficacy of the Adams method, but the difference between DOPRI853 and VSVOABM was so great that both FRABM5 methods are clearly worse than DOPRI853 in these problems.

The function $f(x, y)$ in the Pleiades is more complicated, and its evaluation requires a significant amount of CPU time. DOPRI853 beats ABMVSVO in all tolerances, but the difference is small, which allows both FRABM5 methods to win in this problem. For small tolerances FRABM5, MO = 12 gains about 20% CPU time with respect to DOPRI853.

The Pleiades problem is expensive to evaluate because it is formed by 28 *long* first-order differential equations. Hairer et al. [11, page 427] indicated that the efficacy of Adams methods gets worse as the number of differential equations grows. To study this question in our fixed ratios implementation we add two new expensive to evaluate ODEs extracted from [11].

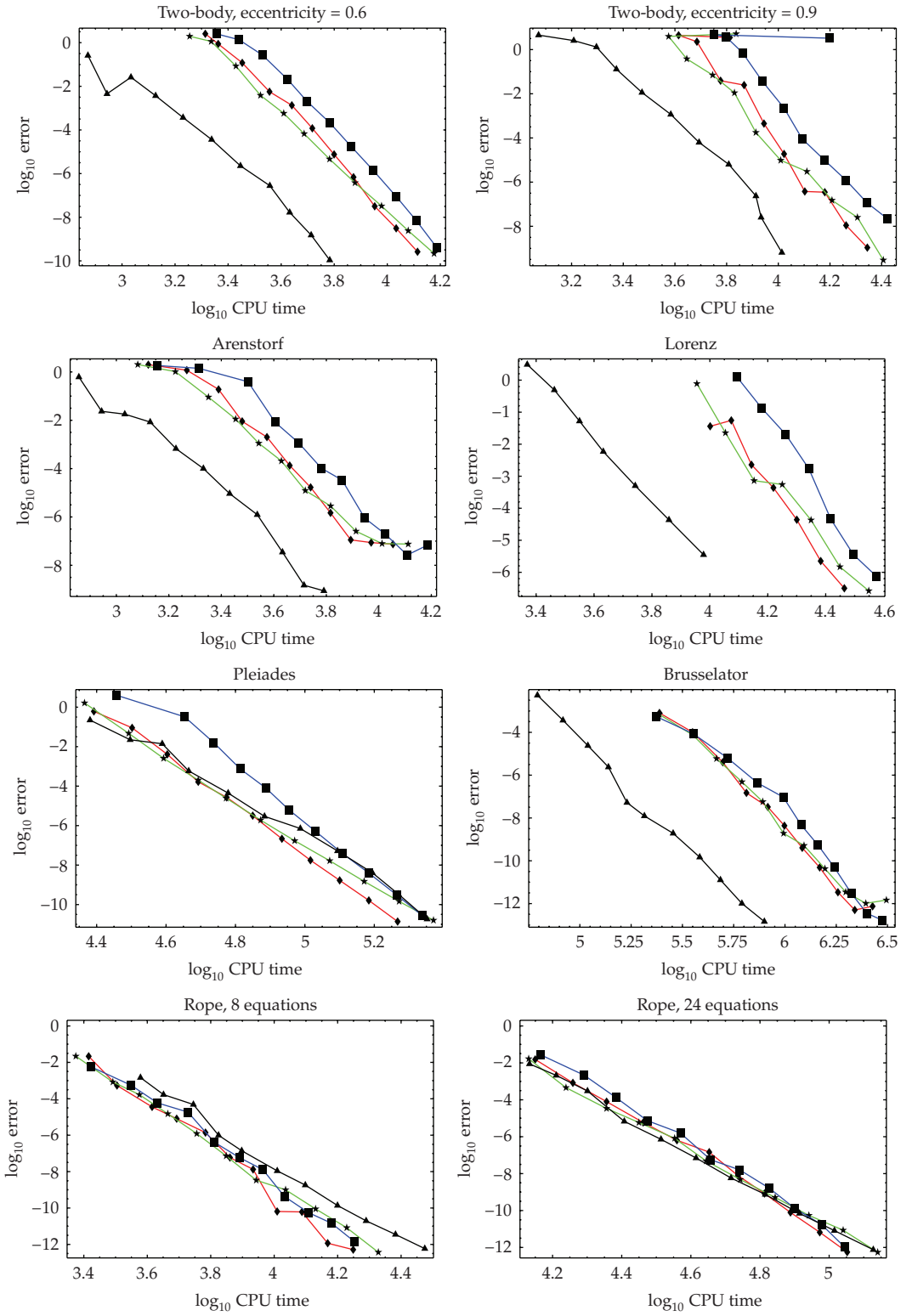

Figure 9: Continued.

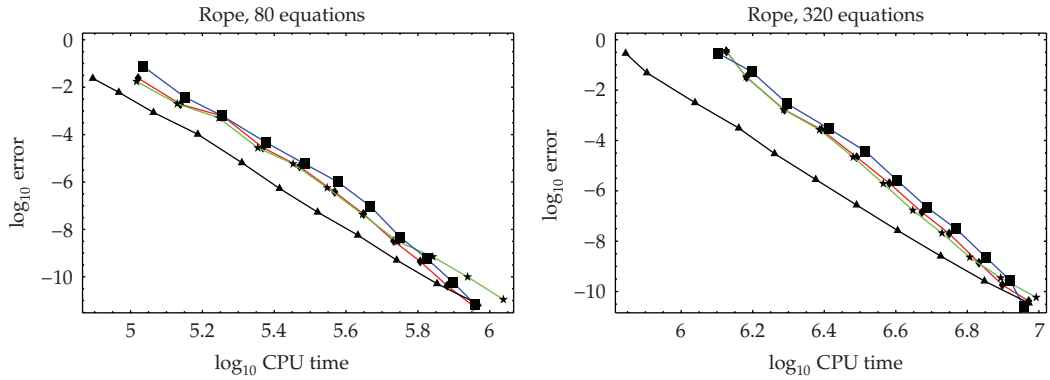

Figure 9: Error versus CPU time. FRABM5 MO = 10: green line with stars, FRABM5 MO = 12: red line with diamonds, VSVOABM MO = 13: blue line with squares, DOPRI853: black line with triangles.

(5) A discretized *Brusselator* with diffusion [11, pages 248-249], constituted by 882 *short* first-order differential equations.

(6) A discretized hanging *Rope* problem. In this problem we let the dimension n in equation (10.8) of [11, pages 247-248] as a free parameter. Then this problem consists of $2n$*long* first-order differential equations.

The comparison in number of function calls in the Brusselator problem does not show any new significant conclusion. However, in the Rope examples FRAMB5, MO = 12 beats ABMVSVO even for small tolerances.

In the Brusselator problem, DOPRI853 clearly beats all FRABM5 methods in CPU time, which corresponds to the indication of Hairer, Nørsett, and Wanner. In the Rope problem we can also see the negative influence of the dimension in the behaviour of the multistep methods. With 8 first-order equations FRABM5 methods are more efficient than DOPRI853, specially with MO = 12, which gains up to 40% in small tolerances. For 24 equations FRABM5, MO = 12 still remains in the first position in these tolerances, but now all of the methods have similar results. However, DOPRI853 takes the lead for 80 equations, and the difference grows for 320 equations. We must conclude that the statement in Hairer et al. [11, page 427] about dimension and efficacy of the Adams code is also valid for the fixed ratios implementation.

VSVOABM is not far from FRABM5 methods in the Brusselator and Rope problems, but it only beats MO = 10 in small tolerances. FRABM5, MO = 12 is superior in all tolerances.

We sum up the conclusions of this efficacy section.

(1) When the CPU time is used to measure the efficacy, DOPRI853 is the best choice when $f(x, y)$ is *short* or it has a high number of components. However, the Pleiades and Rope problems show that FRABM5 can beat DOPRI853 in CPU time when $f(x, y)$ has a moderate number of *long* components. For these kinds of ODEs we propose our fixed ratios implementation, particularly FRABM5 with maximum order MO = 10 for large tolerances and MO = 12 for small ones. VSVOABM was not the best method in any test.

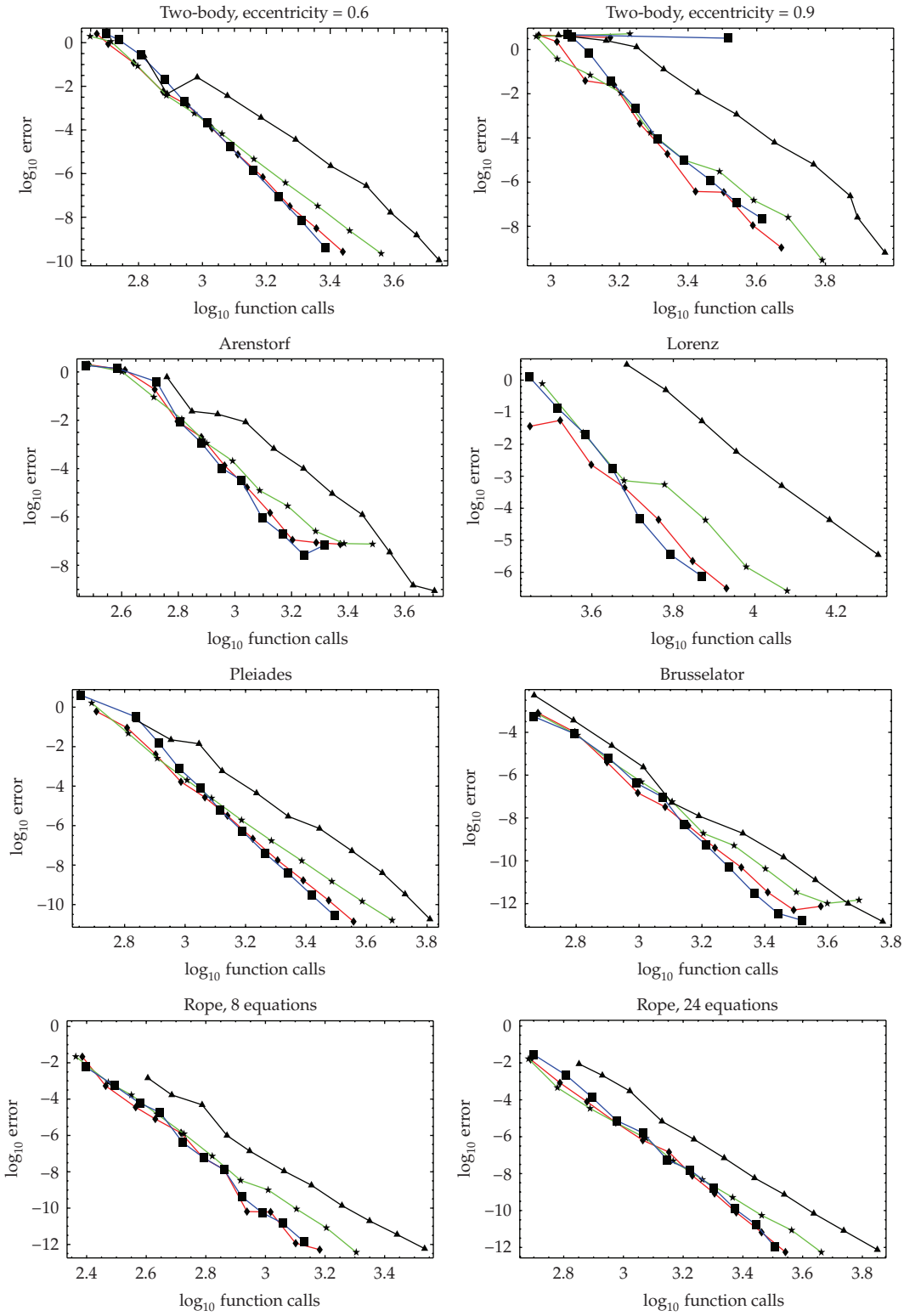

Figure 10: Continued.

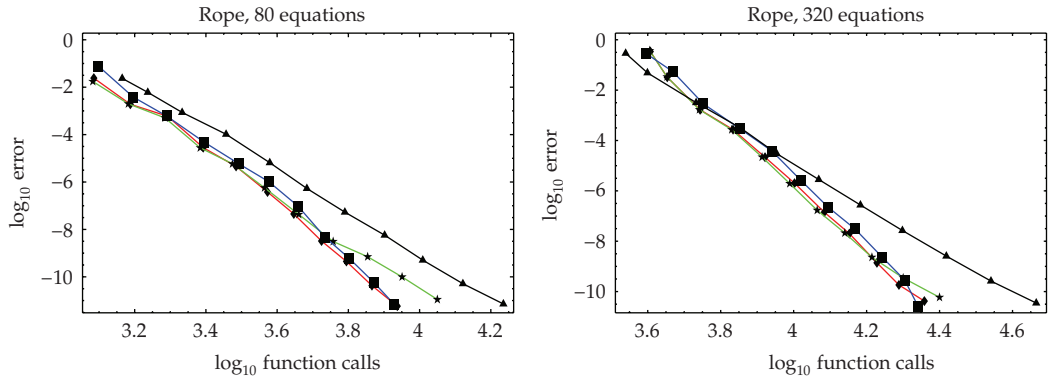

Figure 10: Error versus evaluations. FRABM5 MO = 10: green line with stars, FRABM5 MO = 12: red line with diamonds, VSVOABM MO = 13: blue line with squares, and DOPRI853: black line with triangles.

(2) When comparing the number of evaluations, *fixed ratios* and *VSVO* strategies are comparable; only the maximum order MO produces differences between large and small tolerances. In this chapter, specially relevant if the evaluation of $f(x, y)$ has an associated monetary cost, all the multistep methods won DOPRI853. In particular, both FRABM5 were slightly better than VSVOABM in large tolerances. In small tolerances FRABM5, MO = 12 was the leader in some problems and VSVOABM in others, while the case MO = 10 did not win in any test. FRABM4, MO = 13 is a choice with an acceptable RAM cost that performed slightly better than FRABM5, MO = 12 in these small tolerances, and was virtually tied to ABMVSVO.

References

[1] F. Bashforth and J. C. Adams, *An Attempt to Test the Theories of Capillary Action by Comparing the Theoretical and Measured Forms of Drops of Fluid*, Cambridge University Press, Cambridge, UK, 1883, With an Explanation of the Method of Integration Employed in Constructing the Tables which Give the Theoretical Form of Such Drops, by J. C. Adams.

[2] F. R. Moulton, *New Methods in Exterior Ballistics*, University Chicago Press, Chicago, Ill, USA, 1926.

[3] J. D. Lambert, *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*, John Wiley & Sons, Chichester, UK, 1991.

[4] L. F. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations, The Initial Value Problem*, Freeman and Company, San Francisco, Calif, USA, 1975.

[5] P. N. Brown, G. D. Byrne, and A. C. Hindmarsh, "VODE: a variable-coefficient ODE solver," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, no. 5, pp. 1038–1051, 1989.

[6] A. C. Hindmarsh, "LSODE and LSODI, two new initial value ordinary differential equations solvers," *ACM Signum Newsletterm*, vol. 15, no. 4, pp. 10–11, 1980.

[7] A. C. Hindmarsh, "ODEPACK, a systematized collection of ODE solvers," in *Scientific Computing (Montreal, Que., 1982)*, R. S. Stepleman, et al., Ed., pp. 55–64, IMACS, New Brunswick, NJ, USA, 1983.

[8] L. F. Shampine and M. W. Reichelt, "The MATLAB ODE suite," *SIAM Journal on Scientific Computing*, vol. 18, no. 1, pp. 1–22, 1997.

[9] S. Wolfram, *The Mathematica Book*, Wolfram Media, Champaign, Ill, USA, 5th edition, 2003.

[10] F. T. Krogh, "Changing stepsize in the integration of differential equations using modified divided differences," in *Proceedings of the Conference on the Numerical Solution of Ordinary Differential Equations (Univ. Texas, Austin, Tex., 1972)*, vol. 362 of *Lecture Notes in Mathmatics*, Springer, Berlin, Germany, 1974.

[11] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I. Nonstiff Problem*, vol. 8 of *Springer Series in Computational Mathematics*, Springer, Berlin, Germany, 2nd edition, 1993.

[12] L. F. Shampine, "Variable order Adams codes," *Computers & Mathematics with Applications*, vol. 44, no. 5-6, pp. 749–761, 2002.
[13] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*, John Wiley & Sons, Chichester, UK, 2003.
[14] J. G. Romay, *Mejora de los métodos de Adams de paso y orden variable mediante el cálculo previo de coeficientes para cocientes de amplitudes de paso prefijados*, Ph.D. thesis, Universidad Politécnica de Cartagena, 2004.
[15] J. R. Dormand, *Numerical Methods for Differential Equations: A Computational Approach*, Library of Engineering Mathematics, CRC Press, Boca Raton, Fla, USA, 1996.
[16] L. F. Shampine, *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, New York, NY, USA, 1994.
[17] R. F. Arenstorf, "Periodic solutions of the restricted three-body problem representing analytic continuations of Keplerian elliptic motions," *American Journal of Mathematics*, vol. 85, pp. 27–35, 1963.
[18] J. R. Dormand and P. J. Prince, "Practical Runge-Kutta processes," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, no. 5, pp. 977–989, 1989.