Procedia Computer Science

Volume 29, 2014, Pages 1014–1026

ICCS 2014. 14th International Conference on Computational Science

Implementation of an Adaptive BDF2 Formula and Comparison with the MATLAB Ode15s

E. Alberdi Celaya[1], J. J. Anza Aguirrezabala[2], and P. Chatzipantelidis[3]

[1] Department of Applied Mathematics, University of the Basque Country UPV/EHU, Bilbao, Spain
elisabete.alberdi@ehu.es
[2] Department of Applied Mathematics, University of the Basque Country UPV/EHU, Bilbao, Spain
juanjose.anza@ehu.es
[3] Department of Mathematics, University of Crete, Heraklion-Crete, Greece
chatzipa@math.uoc.gr

Abstract

After applying the Finite Element Method (FEM) to the diffusion-type and wave-type Partial Differential Equations (PDEs), a first order and a second order Ordinary Differential Equation (ODE) systems are obtained respectively. These ODE systems usually present high stiffness, so numerical methods with good stability properties are required in their resolution. MATLAB offers a set of open source adaptive step functions for solving ODEs. One of these functions is the *ode15s* recommended to solve stiff problems and which is based on the Backward Differentiation Formulae (BDF). We describe the error estimation and the step size control implemented in this function. The *ode15s* is a variable order algorithm, and even though it has an adaptive step size implementation, the advancing formula and the local error estimation that uses correspond to the constant step size formula. We have focused on the second order accurate and unconditionally stable BDF (BDF2) and we have implemented a real adaptive step size BDF2 algorithm using the same strategy as the BDF2 implemented in the *ode15s*, resulting the new algorithm more efficient than the one implemented in MATLAB.

Keywords: PDEs, stiff ODEs, Backward Differentiation Formula, adaptive BDF2

1 Introduction

Numerous phenomena of science and engineering are modelled mathematically using systems of Partial Differential Equations (PDEs). Mass, momentum and energy balances, with appropriate constitutive laws are the basis of a broad class of Boundary Condition (BC) problems from which the macroscopic movement of solids, fluids and gases with their corresponding forces can be deduced. Similarly, flow solutions for heat and mass transport problems can be obtained, and interaction problems between different media (mechanical, thermal, chemical, or electromagnetic) can be studied.

doi:10.1016/j.procs.2014.05.091

The mathematical modelling of the continuous media by means of differential equations shows the existing relationship between different applications which lead to similar boundary value problems. For example, the Laplace generalized PDE, which represents the behaviour of many stationary problems, is given by:

$$\boldsymbol{\nabla}\cdot\left[\underbrace{-C\boldsymbol{\nabla}u}_{Conductive\ flux}+\underbrace{\mathbf{D}u}_{Convective\ flux}\right]+\underbrace{Eu}_{Absortion}=\underbrace{f}_{Source} \tag{1}$$

where C and E are physical constants and $\mathbf{D}$ a vector that depends on the problem. This is the case of the heat transfer problem where the unknown function $u(\mathbf{x})$ represents the temperature. If the convective and absorbing terms are not considered, the Laplace-Poisson PDE is obtained:

$$\boldsymbol{\nabla}\cdot(-C\boldsymbol{\nabla}u)=f \tag{2}$$

Equation (2) governs the potential problem, which has many applications such as the flux of an incompressible and non-viscous fluid, the torsion of a profile of any section, the simple heat conduction or the mass diffusion without convection. All these examples are stationary but it is enough to add an additional source term related to the inertia to time change to extend any of these models to the general transient case, with Initial Conditions (IC) and Boundary Conditions (BC). The simplest representative cases are the diffusion and the wave equation. The latter is obtained by taking $f=-G\cdot u_{tt}(\mathbf{x},t)$:

$$\boldsymbol{\nabla}\cdot C\boldsymbol{\nabla}u(\mathbf{x},t)=G\cdot u_{tt}(\mathbf{x},t)\rightarrow\Delta u=\frac{G}{C}u_{tt} \tag{3}$$

If the unknown function $u(\mathbf{x},t)$ represents the transverse displacement of a stretched string with a force T $(C=T)$ and density ρ $(G=\rho)$, the PDE governs the transverse vibrations of the elastic string.

The analytical solution of PDEs in a general domain is not possible and it is necessary the use of numerical methods, being the Finite Element Method (FEM) the most capable in general to deal with any shape domains. After applying the Finite Element Method to the diffusion-type and wave-type PDEs with boundary conditions and initial conditions, a first order and a second order Ordinary Differential Equation (ODE) systems are obtained respectively. The process to obtain approximate solutions of these problems using the FEM, consists of discretizing the domain in elements and nodes. The solution approach is based on the elimination of the spatial derivatives of the PDE and this leads to a system of ODEs.

The ODE system that results after the FEM discretization presents high stiffness, which means that the greater the ratio of the eigenvalues of the Jacobian matrix, the more stiff the system of ODEs [5, 7, 11]. This means that a non-significant part of the solution requires very small step sizes to avoid instability of the whole solution. When we are solving systems of stiff ODEs by numerical integration, it is important to use an accurate algorithm with good stability properties. Since they were introduced, the Backward Differentiation Formulae (BDF) [4] have been widely used due to their good stability properties.

The software package MATLAB offers a set of codes to solve initial value problems [10, 12]:

$$y'(t)=f\left(t,y(t)\right),\quad y\left(t_0\right)=y_0 \tag{4}$$

on a finite time interval $T=[t_0,t_n]$. Some of these ode solvers are recommended to solve nonstiff problems. This is the case of the two explicit Runge-Kutta codes implemented in MATLAB: the *ode23* which is based on the Bogacki-Shampine 3(2) pair [1] and the *ode45* based on the

Dormand Prince (5,4) pair [3]. The *ode113* is another nonstiff ode solver, which is a PECE implementation of the Adams Bashforth-Moulton methods [14].

For stiff problems, the most popular code implemented in MATLAB is the *ode15s* which uses the Backward Differentiation Formulae (BDF) [4] and the Numerical Differentiation Formulae (NDF) [15]. This paper studies the algorithm that supports the *ode15s*: the underlying methods, the local error estimation and the adaptation of the step size to verify the specified error tolerance. The *ode15s* is a variable order and variable step size algorithm, although it uses an advancing formula and a local error estimation of the constant step size formula. We have focused on the second order accurate BDF (BDF2), which is A-stable. We have implemented a real adaptive step size BDF2 algorithm using the same strategy as the BDF2 implemented in the *ode15s*, resulting the one implemented by us more efficient.

The article is organized as follows: characteristics of the *ode15s* have been studied in Section 2. In Section 3 the implementation of the variable step size new BDF2 is explained. In Section 4 some examples are solved and the number of steps given and the errors obtained are analysed. Finally, some conclusions are presented in Section 5.

2 The Ode Solver Ode15s

The *ode15s* is based on the BDF methods [4], and it is possible to use the BDFs of orders $1-5$. The final "*s*" of the *ode15s* indicates that the algorithm is suggested to solve stiff differential equations [13]. By default *ode15s* uses NDF methods [15] which based on BDF methods, anticipate a backward difference of order $(k+1)$ when working in order k. This term has a positive effect on the local truncation error, making the NDFs more accurate than the BDFs and not much less stable, see Table 1. This implies that the NDF2 can achieve the same accuracy as the BDF2 with a step size 26% bigger. This modification was proposed for orders $k = 1, 2, 3, 4$, because it is inefficient for orders greater than 4.

The code always starts solving in order $k = 1$, and the maximum order to be reached can be given to the code as data. Assuming that the values y_{n+1}, y_{n+2}, ..., y_{n+k-1} are available, which are the approximations to the exact solution at the points $t_n + h$, $t_n + 2h$, ..., $t_n + (k-1)h$, the BDFs and NDFs compute the value $y_{n+k} \approx y(t_{n+k})$ using the following expressions:

- BDFs: $\sum_{j=1}^{k} \frac{1}{j} \nabla^j y_{n+k} = h f_{n+k}$.
- NDFs: $\sum_{j=1}^{k} \frac{1}{j} \nabla^j y_{n+k} = h f_{n+k} + \kappa \gamma_k \nabla^{k+1} y_{n+k}$.

being $f_{n+k} = f(t_{n+k}, y_{n+k})$ and $\gamma_k = \sum_{j=1}^{k} \frac{1}{j}$.

k	κ	%step size	A(α) of BDFs	A(α) of NDFs
1	-0.1850	26%	90°	90°
2	-1/9	26%	90°	90°
3	-0.0823	26%	86°	80°
4	-0.0415	12%	73°	66°

Table 1: NDFs of Klopfenstein and Shampine: efficiency and stability with respect to BDFs.

In [15] an alternative form to write the left-hand side of these two expressions is given, which is the same for both cases:

$$\sum_{j=1}^{k} \frac{1}{j} \nabla^j y_{n+k} = \gamma_k \left(y_{n+k} - y_{n+k}^{(0)} \right) + \sum_{j=1}^{k} \gamma_j \nabla^j y_{n+k-1} \tag{5}$$

where:

$$\begin{cases} \gamma_j = \sum_{l=1}^{j} \frac{1}{l} \\ y_{n+k}^{(0)} = \sum_{j=0}^{k} \nabla^j y_{n+k-1} = \nabla^0 y_{n+k-1} + \nabla y_{n+k-1} + ... + \nabla^k y_{n+k-1} \\ y_{n+k} - y_{n+k}^{(0)} = \nabla^{k+1} y_{n+k} \end{cases} \tag{6}$$

Taking into account expression (5) the NDFs can be written as follows:

$$(1-\kappa)\,\gamma_k \left(y_{n+k} - y_{n+k}^{(0)} \right) + \sum_{j=1}^{k} \gamma_j \nabla^j y_{n+k-1} = h f_{n+k} \tag{7}$$

where the values of κ are in the Table 1. BDF methods are obtained for $\kappa = 0$.

2.1 Error Estimation in the Ode15s

The *ode15s* uses the local truncation error as the error estimation:

$$est \approx LTE = C h^{k+1} y^{k+1}(t_n) + O\left(h^{k+2}\right) \tag{8}$$

being C the error constant of the method and k the order of the BDF or NDF formula in which the *ode15s* is solving.

Backward differences are used to calculate an approximation of $y^{k+1}(t_n)$. An approximation of $y(t)$ is obtained by using the backward interpolating polynomial of Newton that passes from the $(k+2)$ points $\{(t_{n+i}, y_{n+i})\}$ for $i = -1, 0, 1, 2, ..., k$:

$$y(t) \approx Q(t) = y_{n+k} + \sum_{j=1}^{k+1} \nabla^j y_{n+k} \frac{1}{j! h^j} \prod_{m=0}^{j-1} (t - t_{n+k-m}) \tag{9}$$

And the $(k+1)$th derivative of expression (9) is calculated:

$$Q^{(k+1)}(t) = \nabla^{k+1} y_{n+k} \frac{1}{(k+1)! h^{k+1}} (k+1)! = \nabla^{k+1} y_{n+k} \frac{1}{h^{k+1}} \tag{10}$$

Obtaining:

$$y^{(k+1)}(t) \approx Q^{(k+1)}(t) = \nabla^{k+1} y_{n+k} \frac{1}{h^{k+1}} \tag{11}$$

Substituting the approximation (11) in (8), the error estimation of the *ode15s* in terms of backward differences is obtained:

$$LTE \approx C \cdot \nabla^{k+1} y_{n+k} = est \tag{12}$$

2.2 Step Size Control in the Ode15s

The first step size can be given to the code as data [6]. Nevertheless, a failed step is easily repaired by the control of the step of the algorithm. Alternatively the *ode15s* can automatically set a trial first step size.

The *ode15s* is not allowed to change either the order or the step size until a minimum of $(k+2)$ consecutive steps are given using the same order formula and the same step size. If one step is unsuccessful, the order of the method or the step size are reduced. When the compulsory $(k+2)$ successful steps are given, it is possible to change the order and the step size. In this case, the step sizes which correspond to orders $(k-1)$ (for $k>1$), k and $(k+1)$ (whenever the maximum order defined has not been reached) are calculated.

After advancing from t_{n-1} to t_n with step size h using the k-order BDF/NDF formula, the local error estimation of the *ode15s* is:

$$est \approx LTE = h^{k+1}\phi(t_n) + O(h^{k+2}) \tag{13}$$

We will suppose that we have just given a step using a step size h and that we are interested in calculating the next step size which corresponds to order k. This step size could be a new step or the repetition of the previous step because it has not been verified the tolerance requirement. In both cases, the next step size is calculated by multiplying the previous step size by a constant σ. Hence, the next step size will be $(\sigma\cdot h)$ and the local error will be [13]:

$$(\sigma\cdot h)^{k+1}\phi(t_{n+1}) + O\left((\sigma h)^{k+2}\right) \tag{14}$$

By making use of Taylor's developments:

$$\phi(t_{n+1}) = \phi(t_n) + h\phi'(t_n) + O\left(h^2\right) \tag{15}$$

Expression (14) turns into:

$$(\sigma\cdot h)^{k+1}\phi(t_{n+1}) + O\left((\sigma h)^{k+2}\right) = \sigma^{k+1}h^{k+1}\phi(t_n) + O\left(h^{k+2}\right) = \sigma^{k+1}est + O\left(h^{k+2}\right) \tag{16}$$

Expression (16) gives the local error made with a step size $(\sigma\cdot h)$. The largest step size that will pass the error test is calculated by chosing σ so that $\sigma^{k+1}\left\|est\right\| \approx Rtol$ is verified, where $Rtol$ is the specified tolerance. This step size is:

$$\sigma\cdot h = \left(\frac{Rtol}{\|est\|}\right)^{1/k+1}\cdot h \tag{17}$$

Because a failed step is expensive, the codes use a fraction of the predicted step. The *ode15s* uses a safety factor $F_S = \frac{5}{6}$ and the new step size for order k is:

$$h_k = F\cdot h \tag{18}$$

where $F = F_S\cdot\sigma$. A new variable z is defined as follows:

$$z = 1.2\left(\frac{\|est\|}{Rtol}\right)^{1/(k+1)} \tag{19}$$

Taking into account (19) the factor $F = 1/z$ represents the hyperbola of Figure 1. Two regions can be clearly distinguished, separated by $\|est\| = Rtol$, where $z_S = 1.2$ is verified. In this case, the factor F takes the value of the safety factor: $F_S = \frac{5}{6}$. For a trial step h, the *ode15s*

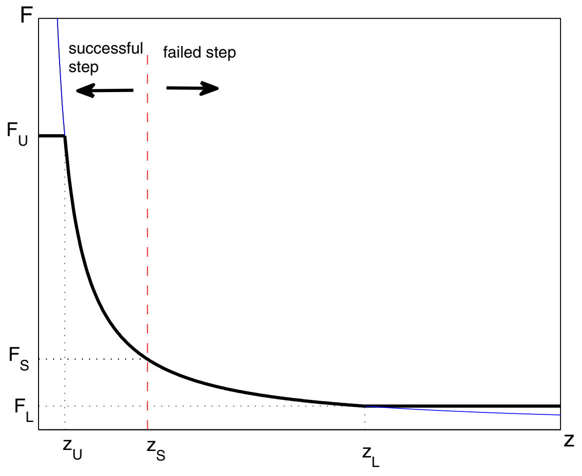

Figure 1: The factor F as function of the variable z.

computes the values of $\|est\|$ and z, and the step is considered successful if $z \leq z_S = 1.2$ is verified.

When the given step is successful, that is to say $z \in [0, 1.2]$, or equivalently $F \in \left[\frac{5}{6}, \infty\right)$, an upper threshold $F_U = 10$ is established and the step size of the k-order method is defined as:

$$h_k = \begin{cases} F_U \cdot h, & z \leq z_U \\ F \cdot h, & z_S < z \leq z_U \end{cases} \tag{20}$$

where $F_U = 10$, $z_U = \frac{1}{F_U} = 0.1$ and $z_S = 1.2$.

In a similar way, the step sizes h_{k-1} and h_{k+1} which correspond to the methods of orders $(k-1)$ and $(k+1)$ are calculated, being the safety factors $\frac{10}{13}$ and $\frac{10}{14}$ respectively. The error estimations of the methods of order $(k-1)$ and $(k+1)$ are calculated, $\|est_{k-1}\|$ and $\|est_{k+1}\|$. And the actual step size is multiplied by the factors $F_{k-1} = \frac{1}{z_{k-1}}$ and $F_{k+1} = \frac{1}{z_{k+1}}$ respectively:

$$z_{k-1} = 1.3 \cdot \left(\frac{\|est_{k-1}\|}{Rtol}\right)^{1/k}, \quad z_{k+1} = 1.4 \cdot \left(\frac{\|est_{k+1}\|}{Rtol}\right)^{1/(k+2)} \tag{21}$$

The step sizes h_{k-1} and h_{k+1} are defined as follows, being the upper thresholds $F_{k-1,U} = 10$ and $F_{k+1,U} = 10$:

$$h_{k-1} = \begin{cases} F_{k-1,U} \cdot h, & z_{k-1} \leq z_{k-1,U} \\ F_{k-1} \cdot h, & z_{k-1,S} < z_{k-1} \leq z_{k-1,U} \end{cases} \tag{22}$$

$$h_{k+1} = \begin{cases} F_{k+1,U} \cdot h, & z_{k+1} \leq z_{k+1,U} \\ F_{k+1} \cdot h, & z_{k+1,S} < z_{k+1} \leq z_{k+1,U} \end{cases} \tag{23}$$

where $z_{k-1,U} = 0.1$, $z_{k-1,S} = 1.3$, $z_{k+1,U} = 0.1$ and $z_{k+1,S} = 1.4$.

When the step sizes h_k, h_{k-1} and h_{k+1} are available, the process that the *ode15s* follows to set the next step size is this one:

1. If $h_{k-1} > h_k$ is verified, the following assignments are made: $h_{new} = h_{k-1}$ and $k_{new} = k-1$. If $h_{k-1} > h_k$ is not verified, $h_{new} = h_k$ and $k_{new} = k$ are considered.
2. Next, h_{k+1} and h_{new} are compared. If $h_{k+1} > h_{new}$, the value h_{k+1} is stored in h_{new} and one unit is added to the order k_{new}.
3. Finally, the value h_{new} is compared with the step size h used in the last step. If $h_{new} > h$, the next step size will be h_{new} and it will be given with order k_{new}. If not, the order and the step size of the last step are maintained.

When the given step is unsuccessful, the step is repeated. The step size of the k-order formula is calculated using a lower threshold $F_L = 0.1$ as follows:

$$h_k = \begin{cases} F \cdot h, & z_S < z \leq z_L \\ F_L \cdot h, & z > z_L \end{cases} \tag{24}$$

where $z_L = \frac{1}{F_L} = 10$ and $z_S = 1.2$ as before.

When $k > 1$, the step size for order $(k-1)$ is also calculated as:

$$h_{k-1} = \begin{cases} F_{k-1} \cdot h, & z_{k-1,S} < z_{k-1} \leq z_{k-1,L} \\ F_{k-1,L} \cdot h, & z_{k-1} > z_{k-1,L} \end{cases} \tag{25}$$

where $F_{k-1,L} = 0.1$, $z_{k-1,L} = \frac{1}{F_{k-1,L}} = 10$ and $z_{k-1,S} = 1.3$ as before.

If $h_{k-1} > h_k$, the next step will be given using the $(k-1)$-order formula and the step size will be the minimum value between the present step h and h_{k-1}.

In second or posterior trials after an unsuccessful step, the new step size is calulated by dividing by 2 the actual step.

3 Adaptive step size BDF2

Even though the *ode15s* changes the step size depending on the local error estimation, the advancing formula that uses for the BDFs or NDFs is the constant step size formula. We will focus on the second order accurate and unconditionally stable BDF2. The advancing formula used by the *ode15s* for this method is:

$$\frac{3}{2}y_{n+2} - 2y_{n+1} + \frac{1}{2}y_n = hf_{n+2} \tag{26}$$

which is the constant step size formula. And the same happens with the expression used to approximate the local truncation error (12).

Nevertheless, the advancing formula of the adaptive step size BDF2 is given by the next expression [6]:

$$y_{n+2} - \frac{(1+w_{n+1})^2}{1+2w_{n+1}}y_{n+1} + \frac{w_{n+1}^2}{1+2w_{n+1}}y_n = h_{n+2}\frac{1+w_{n+1}}{1+2w_{n+1}}f_{n+2} \tag{27}$$

being $w_{n+1} = h_{n+2}/h_{n+1}$, $h_{n+2} = t_{n+2} - t_{n+1}$ and $h_{n+1} = t_{n+1} - t_n$.

Taking into account the adaptive advancing formula of the BDF2, the local truncation error of the adaptive step size BDF2 becomes:

$$LTE = y(t_{n+2}) - y_{n+2} \approx \frac{h_{n+2}^2\,(h_{n+1} + h_{n+2})}{6}y^{(3)}(t_{n+2}) \tag{28}$$

Approximating the third derivative of (28) by:

$$y^{(3)}(t_{n+2}) \approx \frac{1}{h_{n+2}^2}\left[\frac{y_{n+2}-y_{n+1}}{h_{n+2}} - \left(1+\frac{h_{n+2}}{h_{n+1}}\right)\frac{y_{n+1}-y_n}{h_{n+1}} + \frac{h_{n+2}}{h_{n+1}h_n}(y_n - y_{n-1})\right] \tag{29}$$

and substituting (29) in (28), the expression of the local truncation error of the adaptive step size BDF2 is obtained:

$$LTE \approx \frac{(h_{n+1}+h_{n+2})}{6}\left[\frac{y_{n+2}-y_{n+1}}{h_{n+2}} - \left(1+\frac{h_{n+2}}{h_{n+1}}\right)\frac{y_{n+1}-y_n}{h_{n+1}} + \frac{h_{n+2}}{h_{n+1}h_n}(y_n - y_{n-1})\right] \tag{30}$$

being $h_n = t_n - t_{n-1}$.

4 Numerical Results

In this section some problems are solved using the BDF2 in two different ways:

- Case A: The advancing formula (26) and the local truncation error (12) of the constant step size BDF2 have been implemented. This is the way in which the *ode15s* operates but it also changes the order of the method. In this case the order is fixed: $k = 2$. The same criterion used in the *ode15s* to change the step size depending on the local error estimation has been implemented: the step size that follows a successful step is calculated using (20) and the failed steps are repeated by dividing the step size by 2.
- Case B: The adaptive advancing formula (27) and the adaptive local truncation error (30) of the BDF2 are implemented. The new step size is defined depending on the local error estimation, in the same way as in the Case A.

The initialization of both algorithms (Case A and B) has been done in the same way. The use of the local truncation error as the local error control requires three previous values, y_{n-1}, y_n, y_{n+1}. Hence, two additional values are required in the beginning, which have been calculated by the trapezoidal rule. In both cases, the first step size h_0 has been taken as:

$$h_0 = \frac{T}{steps_{ode15s}} \tag{31}$$

being T the time interval and $steps_{ode15s}$ the number of steps given by the *ode15s* when solving the same problem. The second step size h_1 has been given using the same value as h_0.

4.1 Example 1

Consider the problem [16]:

$$y'(t) = \lambda(y(t) - g(t)) + g'(t), \quad T = [0, 2.5], \quad y(0) = 1 \tag{32}$$

being $\lambda = -10^6$ and $g(t) = \sin(10t) + t$. Its exact solution is given by:

$$y(t) = e^{\lambda t} + g(t) \tag{33}$$

In Table 2 the number of steps given by each algorithm when different values of *Rtol* are considered are tabulated. The number of steps given by the *ode15s* is also included. We can observe that the algorithm that uses the variable step formula (Case B) gives less steps than the other. For $Rtol = 10^{-3}$ the total computational time of the algorithm of the Case B is four times smaller than the one of the Case A. And for $Rtol = 10^{-4}$, the algorithm of the Case B is 50 times quicker than the one of the Case A. In Figure 2 we can see the euclidean norm of the error (difference between the exact value and the numerical value) during the interval of integration. We can observe that the errors of the algorithm B are smaller than the ones obtained with the algorithm A. The *ode15s* is superior as it works in variable order $1-5$ and in this problem it gives many steps in order 4.

Rtol	Case A	Case B	*ode15s*
10^{-3}	8638 steps	874 steps	160 steps
10^{-4}	78175 steps	3024 steps	206 steps

Table 2: Number of steps given in example 1.

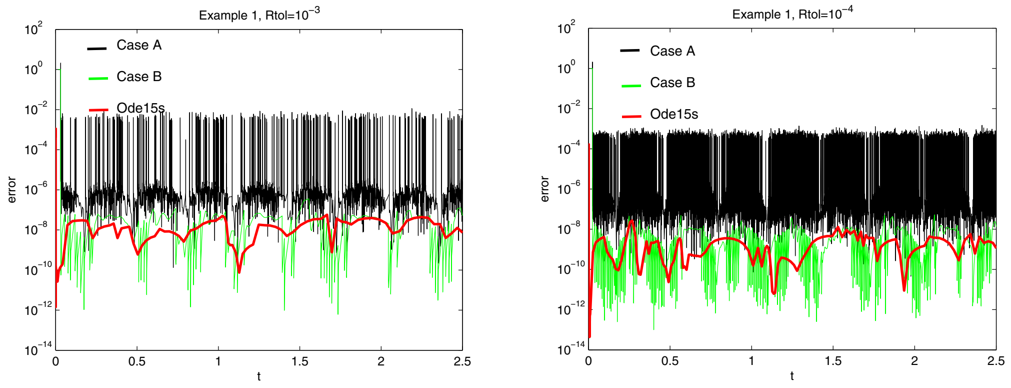

Figure 2: Euclidean norm of the error during the simulation in example 1.

4.2 Example 2

Consider the system of differential equations considered in [8]:

$$\begin{cases} y_1' = -20y_1 - 0.25y_2 - 19.75y_3 \\ y_2' = 20y_1 - 20.25y_2 + 0.25y_3 \\ y_3' = 20y_1 - 19.75y_2 - 0.25y_3 \end{cases}, \; T = [0, 10], \quad y(0) = (1, 0, -1)^T \tag{34}$$

The exact solution of this problem is: $\begin{cases} y_1(t) = \frac{1}{2}\left(e^{-0.5t} + e^{-20t}\left(\cos 20t + \sin 20t\right)\right) \\ y_2(t) = \frac{1}{2}\left(e^{-0.5t} - e^{-20t}\left(\cos 20t - \sin 20t\right)\right) \\ y_3(t) = -\frac{1}{2}\left(e^{-0.5t} + e^{-20t}\left(\cos 20t - \sin 20t\right)\right) \end{cases}$

The variable step size algorithm (Case B) requires less steps than the algorithm A, Table 3. The computational time of both algorithms is similar for $Rtol = 10^{-3}$; for $Rtol = 10^{-4}$ the algorithm B is 3 times quicker and it is 16 times quicker for $Rtol = 10^{-5}$. The errors obtained by the algorithm B are also smaller than the ones obtained by the algorithm A, see Figure 3. Again, the *ode15s* is superior as many steps are given in order 4.

Rtol	Case A	Case B	*ode15s*
10^{-3}	430 steps	126 steps	64 steps
10^{-4}	3385 steps	329 steps	89 steps
10^{-5}	28979 steps	1202 steps	122 steps

Table 3: Number of steps given in example 2.

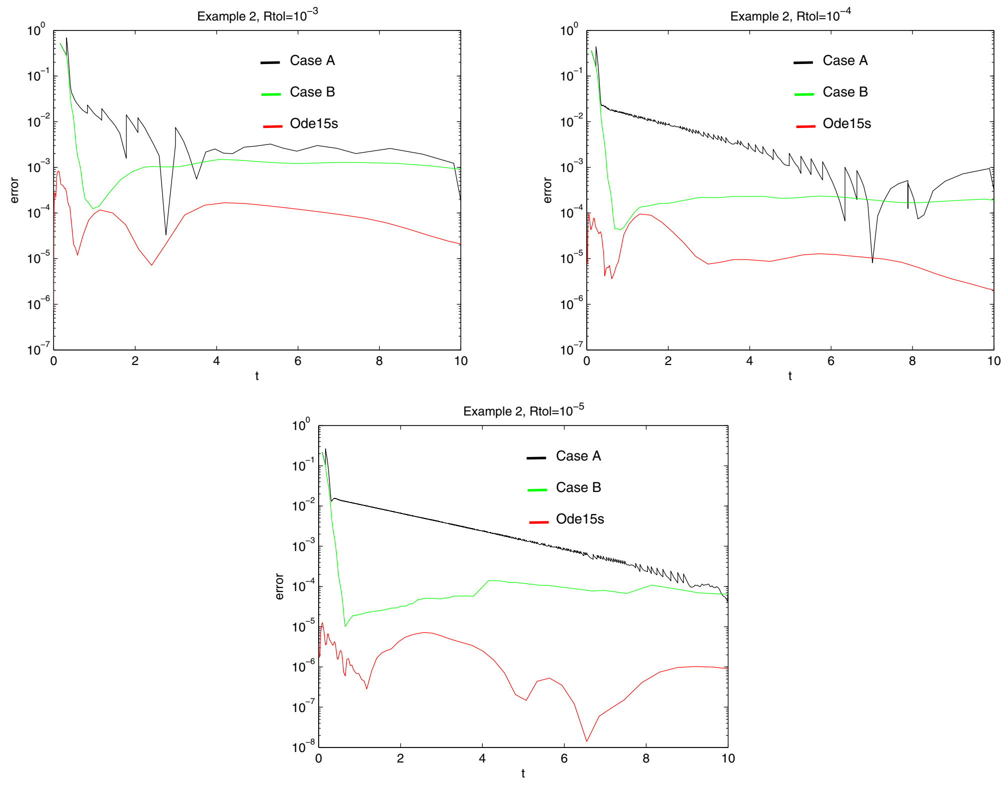

Figure 3: Euclidean norm of the error during the simulation in example 2.

4.3 Example 3

We consider the following stiff system considered by Hosseini and Hojjati in [9].

$$\begin{cases} y_1' = -0.1y_1 - 49.9y_2 \\ y_2' = -50y_2 \\ y_3' = 70y_2 - 120y_3 \end{cases}, \; T = [0,1], \quad y(0) = (2,1,2)^T \tag{35}$$

with stiffness ratio 1200 and exact solution: $\begin{cases} y_1(t) = e^{-50t} + e^{-0.1t} \\ y_2(t) = e^{-50t} \\ y_3(t) = e^{-50t} + e^{-120t} \end{cases}$

Again the algorithm of the case B requires less computational time, about 10 times less than the algorithm A for $Rtol = 10^{-5}$ and the results are superior, see Table 4 and Figure 4.

Rtol	Case A	Case B	*ode15s*
10^{-3}	75 steps	40 steps	68 steps
10^{-4}	702 steps	275 steps	87 steps
10^{-5}	13224 steps	727 steps	104 steps

Table 4: Number of steps given in example 3.

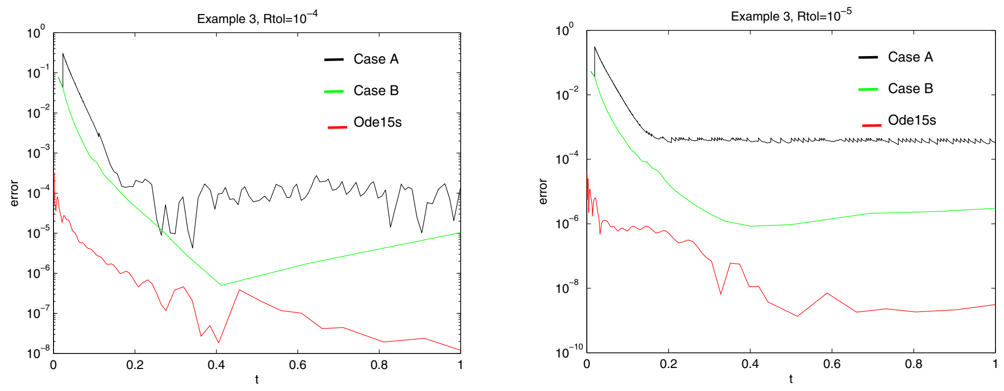

Figure 4: Euclidean norm of the error during the simulation in example 3.

4.4 Example 4

We consider the following stiff system as considered by Cash in [2]:

$$\begin{cases} y_1' = -\alpha y_1 - \beta y_2 + (\alpha+\beta-1)e^{-t} \\ y_2' = \beta y_1 - \alpha y_2 + (\alpha-\beta-1)e^{-t} \end{cases}, T = [0, 20], \quad y(0) = (1,1)^T \tag{36}$$

The eigenvalues of the Jacobian matrix are $-\alpha \pm \beta i$, and its exact solution is this one: $y_1(t) = y_2(t) = e^{-t}$. We have solved the problem for $\alpha = 1, \beta = 15$. In Table 5 the number of steps given in each case is tabulated. The variable step size algorithm (Case B) requires less steps than the algorithm A. In all the cases the computational time of the algorithm B is smaller than the one of the algorithm A: 10 times smaller for $Rtol = 10^{-3}$, 3 times smaller for $Rtol = 10^{-4}$ and 30 times smaller for $Rtol = 10^{-5}$. Again, the results obtained with the algorithm B are smaller than the ones obtained with the algorithm A, see Figure 5. This time the adaptive BDF2 algorithm is also more efficient than the *ode15s*. This is because the *ode15s* only reaches order 3 in this problem. In Figure 6 we can see the error estimations and the step sizes given by the *ode15s* in this problem. The order in which each step has been given has been represented too.

Rtol	Case A	Case B	*ode15s*
10^{-3}	403 steps	41 steps	414 steps
10^{-4}	3607 steps	353 steps	399 steps
10^{-5}	35311 steps	654 steps	387 steps

Table 5: Number of steps given in example 4.

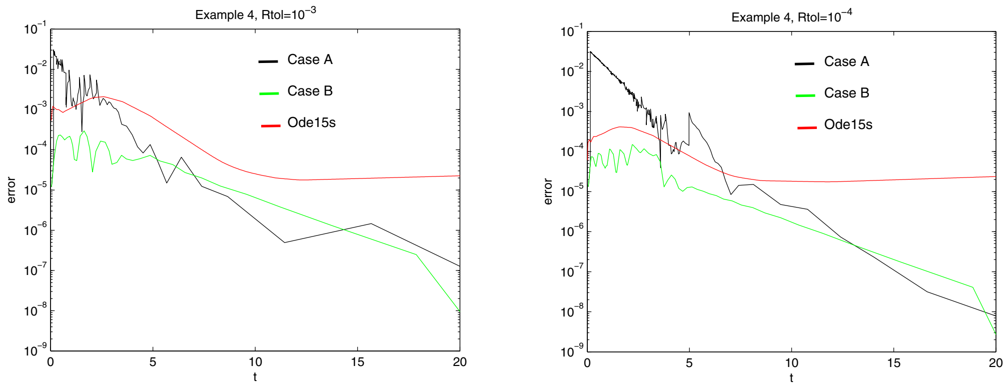

Figure 5: Euclidean norm of the error during the simulation in example 4.

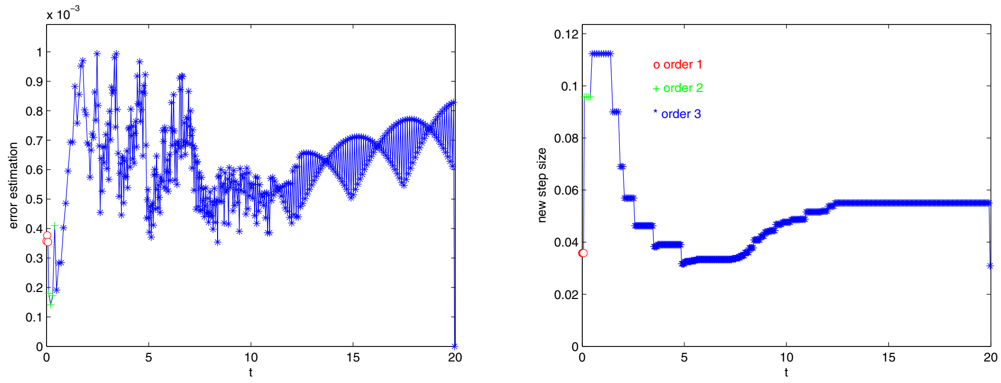

Figure 6: Error estimation and step sizes for example 4 in *ode15s*, $Rtol = 10^{-3}$.

5 Conclusions

When we are solving systems of stiff ODEs by numerical integration, it is crucial the use of algorithms with good accuracy and stability properties. It can be concluded from this analysis that in general, the adaptive BDF2 algorithm is more efficient than the one that uses the constant step size advancing formulae.

Acknowledgements. The first author was partially funded by the Basque Government Consolidated Research Group Grant IT649-13 on “Mathematical Modeling, Simulation, and Industrial Applications (M2SI)”.

References

[1] P. Bogacki and L. F. Shampine. A 3(2) pair of Runge-Kutta formulas. *Appl. Math. Lett.*, 2:1–9, 1989.

[2] J. R. Cash. On the integration of stiff systems of ODEs using extended backward differentiation formula. *Numer. Math.*, 34 (2):235–246, 1980.

[3] J. R. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.*, 6 (1):19–26, 1980.

[4] C. W. Gear. *Numerical initial value problems in Ordinary Differential Equations.* Prentice Hall, New Jersey, 1971.

[5] E. Hairer and G. Wanner. *Solving ordinary differential equations, II, Stiff and Differential Algebraic Problems.* Springer, Berlin, 1991.

[6] E. Hairer, G. Wanner, and S. P. N ørsett. *Solving ordinary differential equations, I, Nonstiff problems.* Springer, Berlin, 1993.

[7] M. T. Heath. *Scientific Computing. An introductory survey.* Mc Graw Hill, New York, 1997.

[8] G. Hojjati, M. Y. Rahimi Ardabili, and S. M. Hosseini. A-EBDF: an adaptative method for numerical solution of stiff systems of ODEs. *Math. Comput. Simul.*, 66:33–41, 2004.

[9] S. M. Hosseini and G. Hojjati. Matrix-free MEBDF method for numerical solution of systems of ODEs. *Math. Comput. Modell.*, 29:67–77, 1999.

[10] The Math Works Inc. http://www.mathworks.com.

[11] J. D. Lambert. *Computational Methods in Ordinary Differential Equations.* Wiley, London, 1973.

[12] L. F. Shampine and R. M. Corless. Initial value problems for ODEs in problem solving environments. *J. Comput. Appl. Math.*, 125:31–40, 2000.

[13] L. F. Shampine, I. Gladwell, and S. Thompson. *Solving ODEs with Matlab.* Cambridge University Press, New York, 2003.

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

[15] L. F. Shampine and M. W. Reichelt. The MATLAB ODE Suite. *SIAM J. Sci. Comput.*, 18 (1):1–22, 1997.

[16] J. Vigo-Aguiar, J. Martin-Vaquero, and R. Criado. On the stability of exponential fitting BDF algorithms. *Journal of Computational and Applied Mathematics*, 175:183–194, 2005.