Solving Initial Value Problems for Ordinary Differential Equations by two approaches: BDF and Piecewise-linearized Methods

J. Ibáñez [a], V. Hernández [a], E. Arias [b], P. A. Ruiz [a,*]

[a] *Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas, Technical University of Valencia, Camino de Vera s/n, 46022-Valencia (Spain)*

[b] *Departamento de Sistemas Informáticos, University of Castilla-La Mancha, Avda. España s/n, 02071-Albacete (Spain)*

Abstract

Many scientific and engineering problems are described using Ordinary Differential Equations (ODEs), where the analytic solution is unknown. Much research has been done by the scientific community on developing numerical methods which can provide an approximate solution of the original ODE. In this work, two approaches have been considered based on BDF and Piecewise-linearized Methods. The approach based on BDF methods uses a Chord-Shamanskii iteration for computing the nonlinear system which is obtained when the BDF schema is used. Two approaches based on piecewise-linearized methods have also been considered. These approaches are based on a theorem proved in this paper which allows to compute the approximate solution at each time step by means of a block-oriented method based on diagonal Padé approximations. The difference between these implementations is in using or not using the scale and squaring technique.

Five algorithms based on these approaches have been developed. MATLAB and Fortran versions of the above algorithms have been developed, comparing both precision and computational costs. BLAS and LAPACK libraries have been used in Fortran implementations. In order to compare in equality of conditions all implementations, algorithms with fixed step have been considered. Four of the five case studies analyzed come from biology and chemical kinetics stiff problems. Experimental results show the advantages of the proposed algorithms, especially when they are integrating stiff problems.

Key words: Ordinary Differential Equation (ODE), Initial Value Problem (IVP), Backward Differentiation Formula (BDF) Method, Piecewise-linearized Method, Diagonal Padé Approximation, BLAS, LAPACK
PACS: 87.64.Aa

1 Introduction

Much research has been done by the scientific community on developing numerical methods which permit an approximate solution to ODEs. In recent years many review articles and books have appeared on numerical methods for integrating ODEs, in particular in stiff cases [1]. Stiff problems are very common problems in many fields of the applied sciences: control theory, biology, chemical kinetics, electronic circuit theory, fluids, etc. One of the most popular multistep method families for solving IVPs for stiff ODEs is formed by the BDF methods [2–5]. Another approach used in this paper is based on piecewise-linearized methods. These methods solve an IVP by approximating the right-hand side of the ODE by means of a Taylor polynomial of degree one. The resulting approximation can be integrated analytically to obtain the solution in each subinterval and yields the exact solution for linear problems. In [6,7] an exhaustive study of this approach is introduced. The developed piecewise-linearized algorithms use Theorem 1 and Corollary 1 (Section 3) which allow to compute the approximate solution at each time step by means of a block-oriented method based on diagonal Padé approximations.

The paper is structured as follows. In Section 2 a BDF algorithm is presented. The proposed approaches for solving IVPs by a piecewise-linearized method based on diagonal Padé approximations are presented in Section 3. The experimental results are shown in Section 4. Finally, some conclusions and future work are outlined in Section 5.

2 A BDF algorithm

Let the IVP

$$\dot{x}(t) = f(t, x(t)),\ t \in [t_0, t_f], x(t_0) = x_0, \tag{1}$$

where $f(t, x(t)) \in \mathbb{R}^n$, $t \in [t_0, t_f]$, satisfies the necessary conditions under which the problem has a unique solution.

In this section an algorithm is presented which solves IVPs for ODEs by means of a BDF approach [8, Chapter 5] which uses a Chord-Shamanskii method [9, Chapter 5] to solve the implicit equations that appear in BDF methods. In a BDF scheme, the integration interval $[t_0, t_f]$ is divided so that the approximate solution at t_i, x_i, is obtained by solving an implicit nonlinear system obtained

* Corresponding author. Tel: +34-96-3877007 Ext:73538. Fax: +34-96-3877359
Email addresses: jjibanez@dsic.upv.es (J. Ibáñez), vhernand@dsic.upv.es (V. Hernández), earias@dsi.uclm.es (E. Arias), pruiz@dsic.upv.es (P. A. Ruiz).

by differentiating the polynomial which interpolates past values of x_i, and setting the derivative at t_i to $f(t_i, x_i)$. Several methods have been implemented for solving that implicit nonlinear system; however, in the context of stiff ODEs, one of the better choices is to apply implicit schemes based on Newton's or quasi-Newton methods.

If $\{t_0, t_1, \cdots t_f\}$ is a partition of interval $[t_0, t_f]$ and a BDF scheme is applied, the approximate solution x_i is obtained by solving the following equation:

$$x_i - \sum_{j=1}^{r} \alpha_j x_{i-j} - \Delta t_{i-1} \beta f(t_i, x_i) = 0, \tag{2}$$

where $\Delta t_{i-1} = t_i - t_{i-1}$, and α_j $(j = 1, 2, \cdots, r)$, β are parameters which appear in Table 1, where r is the order of BDF method

Usually a Newton method is used to solve (2) at each time step. In this way, x_i is obtained from the Newton iteration

$$\begin{gathered} x_i^0 = x_{i-1}, \\ \left(I_n - h\beta J_i^{l-1}\right) \Delta x = -F(x_i^l), x_i^l = x_i^{l-1} + \Delta x, l \geq 1, \end{gathered} \tag{3}$$

where J_i^{l-1} is the Jacobian matrix evaluated at (x_i^{l-1}, t_i).

In this paper an inexact Newton technique for solving (3) has been used which significantly speeds up the implicit BDF integrator without loss of accuracy. This approach [9, pp. 86] uses a combination of the Chord and Shamanskii methods, based on the reduction in the nonlinear residual. This latter option decides if the Jacobian should be recomputed based on the ratio of successive residuals.

If two consecutive approaches x_i^{l-1} and x_i^l verify that $\left\|F(x_i^l)\right\| / \left\|F(x_i^{l-1})\right\|$ is below a given threshold, the Jacobian is not recomputed. If the ratio is too large, the Jacobian matrix at x_i^l is computed for use in the subsequent chord steps. In addition, a threshold for the ratio of successive residuals is also input. The Jacobian matrix is recomputed and factored if either the ratio of successive residuals exceeds the threshold $0 < \rho < 1$ or the number of iterations without an update exceeds a parameter $m \in \mathbb{N}$. Algorithm 1 (*iodbcs*) solves the IVP for ODEs (1) by means of the above BDF Chord-Shamanskii method.

Algorithm 1 $[\{x_i\}, e] = iodbcs(ff, Jf, x_0, t_0, t_f, \Delta t, r, tol, m, \rho)$
Inputs: *functions ff and Jf compute $f(\tau, x) \in \mathbb{R}^n$ and the Jacobian matrix $J(\tau, x) \in \mathbb{R}^n$ ($\tau \in \mathbb{R}$, $x \in \mathbb{R}^n$); vector of initial conditions $x_0 \in \mathbb{R}^n$; initial time $t_0 \in \mathbb{R}$; final time $t_f \in \mathbb{R}$; step size Δt; order $r \in \mathbb{N}$ of the BDF method; tolerance vector $tol \in \mathbb{R}^2$ that contains the relative error tolerance (tol_1) and the absolute error tolerance (tol_2); maximum number $m \in \mathbb{N}$ of iterations*

	β	α_1	α_2	α_3	α_4	α_5
$r=1$	1	1				
$r=2$	2/3	4/3	-1/3			
$r=3$	6/11	18/11	-9/11	2/11		
$r=4$	12/25	48/25	-36/25	16/25	-3/25	
$r=5$	60/137	300/137	-300/137	200/137	-75/137	12/137

Table 1
BDF method parameters (order r=1, 2, 3, 4 and 5).

without computing the Jacobian matrix; threshold ρ
Outputs: *Solutions* $\{x_i\}$ $(x_i \in \mathbb{R}^n)$ *at* $t_0, t_0+\Delta t, t_0+2\Delta t, \ldots$; $e \in \mathbb{Z}$ *indicates the convergence of the method (if* $e=-1$ *the method does not converge)*

1 Initialize α *and* β *according to the values given in Table 1*
2 $i=0$
3 $m = \lceil (t_f - t_0)/\Delta t \rceil$;
4 For $i = 1:m$
4.1 $t_i = t_{i-1} + \Delta t$
4.2 $p = \min(r, i)$
4.3 $f_0 = \sum_{j=1}^{r} \alpha_j x_{i-j}$
4.4 $x_i = x_{i-1}$
4.5 $f = ff(t_i, x_i)$
4.6 $r_c = \|f\|_\infty$; $r_c = r_0$
4.7 While $r_c \geq t_1 r_0 + tol_2$
4.7.1 $J = Jf(t_i, x_i)$
4.7.2 $[L, U] = lu(I - \Delta t \beta_p J)$ *(LU decomposition)*
4.7.3 $i_s = 0$; $\rho = 0$
4.7.4 While $i_s < m$ *and* $\rho \leq m$
4.7.4.1 $i_s = i_s + 1$
4.7.4.2 Solve the lower linear system $Ly = f_0 - x_i + f$ *for* y
4.7.4.3 Solve the upper linear system $U\Delta x = y$ *for* Δx
4.7.4.4 $x_i = x_i + \Delta x$
4.7.4.5 $f = ff(t_i, x_i)$
4.7.4.6 $r_+ = \|f\|_\infty$
4.7.4.7 $\sigma = r_+ / r_m$; $r_c = r_+$
4.7.4.8 If $\|\Delta x\| < tol_1 r_0 + r_1$ *Leave While loop 4.7.4*
4.8 If $\sigma > 1$
4.8.1 $e = -1$ *(The method does not converge)*
4.8.2 Return

3 A piecewise-linearized approach for solving IVPs for ODEs

Given a partition $t_0 < t_1 < \cdots < t_{l-1} < t_l = t_f$ of interval $[t_0, t_f]$, IVP (1) can be approximated by means of a set of IVPs obtained as a result of the linear approximation of $f(t, x(t))$ at each subinterval [6,10]

$$\dot{y}(t) = f_i + J_i(y(t) - y_i) + g_i(t - t_i),\ t \in [t_i, t_{i+1}],$$
$$y(t_i) = y_i,\ i = 0, 1, \cdots, l-1,$$

where

$$f_i = f(t_i, y_i) \in \mathbb{R}^n,$$
$$J_i = \frac{\partial f}{\partial x}(t_i, y_i) \in \mathbb{R}^{n \times n} (\text{Jacobian matrix}),$$
$$g_i = \frac{\partial f}{\partial t}(t_i, y_i) \in \mathbb{R}^n (\text{gradient vector}).$$

The IVP associated to the first subinterval is

$$\dot{y}(t) = f_0 + J_0(y(t) - y_0) + g_0(t - t_0), t \in [t_0, t_1],$$
$$y(t_0) = y_0 = x_0.$$

Its analytic solution is given by

$$y(t) = y_0 + \int_{t_0}^{t} e^{J_0(t-\tau)}[f_0 + g_0(\tau - t_0)]d\tau\,, t \in [t_0, t_1],$$

therefore it is possible to compute $y_1 = y(t_1)$.

By proceeding in the same way, the analytic solution of IVP associated to subinterval i, $i = 1, \cdots, l-1$,

$$\dot{y}(t) = f_i + J_i(y(t) - y_i) + g_i(t - t_i),\ t \in [t_i, t_{i+1}],\ \ y(t_i) = y_i$$

is given by

$$y(t) = y_i + \int_{t_i}^{t} e^{J_i(t-\tau)}[f_i + g_i(\tau - t_i)]d\tau,\ \ t \in [t_i, t_{i+1}]\,. \tag{4}$$

If second order partial derivatives of $f(t, x)$ are bounded on $[t_0, t_f] \times \mathbb{R}^n$, then the above piecewise-linearized method converges [6]. If a (1,1) Padé approximation is used for computing $e^{J_i(t-t_i)}$, the above method is consistent of order 1 for autonomous ODEs and 2 for non-autonomous ODEs and linearly stable [7, pp. 26].

The piecewise-linearized approaches for solving IVPs presented in this paper are based on the following theorem and corollary.

Theorem 1 *The solution of IVP*

$$\begin{gathered}\dot{y}(t) = f_i + J_i(y(t) - y_i) + g_i(t - t_i),\ t \in [t_i, t_{i+1}], \\ y(t_i) = y_i \in \mathbb{R}^n, \\ f_i \in \mathbb{R}^n, J_i \in \mathbb{R}^{n\times n}, g_i \in \mathbb{R}^n,\end{gathered} \tag{5}$$

is

$$y(t) = y_i + F_{12}^{(i)}(t - t_i)f_i + F_{13}^{(i)}(t - t_i)g_i, \tag{6}$$

where $F_{12}^{(i)}(t - t_i)$ *and* $F_{13}^{(i)}(t - t_i)$ *are the blocks* $(1, 2)$ *and* $(1, 3)$ *of* $e^{C_i(t-t_i)}$, *and*

$$C_i = \begin{bmatrix} J_i & I_n & 0_n \\ 0_n & 0_n & I_n \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Proof. Defining $s = \tau - t_i$ and $\theta = t - t_i$, the integral which appears in (4) can be expressed as

$$\int_{t_i}^{t} e^{J_i(t-\tau)}[f_i + g_i(\tau - t_i)]d\tau = \left[\int_0^\theta e^{J_i(\theta-s)}ds\right] f_i + \left[\int_0^\theta e^{J_i(\theta-s)}sds\right] g_i. \tag{7}$$

Because C_i is an upper-triangular block matrix, the exponential of $C_i\theta$ has the same structure,

$$e^{C_i\theta} = \begin{bmatrix} F_{11}^{(i)}(\theta) & F_{12}^{(i)}(\theta) & F_{13}^{(i)}(\theta) \\ 0_n & F_{22}^{(i)}(\theta) & F_{23}^{(i)}(\theta) \\ 0_n & 0_n & F_{33}^{(i)}(\theta) \end{bmatrix},$$

where $F_{jk}^{(i)}(\theta)$, $1 \le j \le k \le 3$, are square matrices of order n. Since

$$\tfrac{de^{C_i\theta}}{d\theta} = C_i e^{C_i\theta}\,, e^{C_i\theta}|_{\theta=0} = I_{3n},$$

the following IVPs are obtained

$$\frac{dF_{11}^{(i)}(\theta)}{d\theta} = J_i F_{11}^{(i)}(\theta),\ F_{11}^{(i)}(0) = I_n, \tag{8}$$

$$\frac{dF_{22}^{(i)}(\theta)}{d\theta} = 0,\ F_{22}^{(i)}(0) = I_n, \tag{9}$$

$$\frac{dF_{33}^{(i)}(\theta)}{d\theta} = 0,\ F_{33}^{(i)}(0) = I_n, \tag{10}$$

$$\frac{dF_{23}^{(i)}(\theta)}{d\theta} = F_{33}^{(i)}(\theta),\ F_{23}^{(i)}(0) = 0_n, \tag{11}$$

$$\frac{dF_{12}^{(i)}(\theta)}{d\theta} = J_i F_{12}^{(i)}(\theta) + F_{22}^{(i)}(\theta), F_{12}^{(i)}(0) = 0_n, \tag{12}$$

$$\frac{dF_{13}^{(i)}(\theta)}{d\theta} = J_i F_{13}^{(i)}(\theta) + F_{23}^{(i)}(\theta),\ F_{13}^{(i)}(0) = 0_n. \tag{13}$$

The solutions of (8), (9) and (10) are given by

$$F_{11}^{(i)}(\theta) = e^{J_i\theta},$$
$$F_{22}^{(i)}(\theta) = I_n,$$
$$F_{33}^{(i)}(\theta) = I_n.$$

Therefore the solutions of (11), (12) and (13) are

$$F_{23}^{(i)}(\theta) = \theta I_n, \tag{14}$$

$$F_{12}^{(i)}(\theta) = \int_0^\theta e^{J_i(\theta-s)} ds, \tag{15}$$

$$F_{13}^{(i)}(\theta) = \int_0^\theta e^{J_i(\theta-s)} s ds. \tag{16}$$

Note that the integrals involved in (7) can be computed by means of (15) and (16). In conclusion, once the Jacobian matrix J_i and the vectors g_i and f_i have been computed, the solution of IVP (5) is given by

$$y(t) = y_i + F_{12}^{(i)}(\theta) f_i + F_{13}^{(i)}(\theta) g_i.$$

As $\theta = t - t_i$, then

$$y(t) = y_i + F_{12}^{(i)}(t - t_i) f_i + F_{13}^{(i)}(t - t_i) g_i, \tag{17}$$

so the theorem is proved. □

According to Theorem 1, the approximate solution of IVP (5) at t_{i+1} is ob-

tained from the approximate solution at t_i by the following expression

$$y_{i+1} = y_i + F_{12}^{(i)}(\Delta t_i) f_i + F_{13}^{(i)}(\Delta t_i) g_i, \tag{18}$$

where $\Delta t_i = t_{i+1} - t_i$.

For autonomous ODEs the following result is obtained.

Corollary 1 *The solution of IVP*

$$\dot{y}(t) = f_i + J_i(y(t) - y_i),\ y(t_i) = y_i, \tag{19}$$

is

$$y(t) = y_i + F_{12}^{(i)}(t - t_i) f_i,$$

where $F_{12}^{(i)}(t - t_i)$ *is the block (1,2) of matrix* $e^{C_i(t-t_i)}$*, and*

$$C_i = \begin{bmatrix} J_i & I_n \\ 0_n & 0_n \end{bmatrix}.$$

Proof. It is enough to apply Theorem 1 for $g_i = 0_{n\times 1} \in \mathbb{R}^n$. □

According to Corollary 1, the approximate solution of IVP (19) at t_{i+1} is obtained from the approximate solution at t_i by the following expression

$$y_{i+1} = y_i + F_{12}^{(i)}(\Delta t_i) f_i. \tag{20}$$

Algorithm 2 is consequence of Theorem 1. This algorithm (*inolex*) computes the approximate solution of the IVP for non-autonomous ODEs (1) by means of a piecewise-linearized method based on the exponential of matrix $C_i \Delta t_i$.

Algorithm 2 $\{x_i\} = inolex(data, x_0, t_0, t_f, \Delta t)$
Inputs: *Data is a function that computes Jacobian matrix* $J(\tau, x) \in \mathbb{R}^{n\times n}$ *and function vector* $f(\tau, x) \in \mathbb{R}^n$ *(*$\tau \in \mathbb{R}$*,* $x \in \mathbb{R}^n$*); vector* $x_0 \in \mathbb{R}^n$ *of initial conditions ; initial time* $t_0 \in \mathbb{R}$*; final time* $t_f \in \mathbb{R}$*; step size* $\Delta t \in \mathbb{R}$
Output: *Vector of solutions* $\{x_i\}$ $(x_i \in \mathbb{R}^n)$ *at* $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots$

1 $m = \lceil (t_f - t_0)/\Delta t \rceil$
2 For $i = 0 : m - 1$
2.1 $[J, f, g] = data(t_i, x_i)$
2.2 $C = \begin{bmatrix} J & I_n & 0_n \\ 0_n & 0_n & I_n \\ 0_n & 0_n & 0_n \end{bmatrix}$
2.3 $F = e^{C\Delta t}$

2.4 $t_{i+1} = t_i + \Delta t$
2.5 $x_{i+1} = x_i + F_{12}f + F_{13}g$

3.1 Algorithms based on the scaling and squaring technique

The (p, q) Padé approximation to e^A is defined by

$$R_{pq} = [D_{pq}(A)]^{-1} N_{pq}(A),$$

where

$$N_{pq}(A) = \sum_{k=0}^{p} \frac{(p+q-k)!p!}{(p+k)!k!(p-k)!} A^k$$

and

$$D_{pq}(A) = \sum_{k=0}^{p} \frac{(p+q-k)!p!}{(p+k)!k!(p-k)!} (-A)^k.$$

Non-singularity of $D_{pq}(A)$ is assured if p and q are large enough or if the eigenvalues of A are negative.

The problem with this method is that it only provides good approaches near the origin [11, p.573]. This problem can be avoided by using the widely used scaling and squaring method for computing the matrix exponential [12,13] by exploiting the equality

$$e^A = \left(e^{A/m}\right)^m.$$

The idea is to choose m to be a power of two ($m = 2^j$) for which $e^{A/m}$ can be reliably and efficiently computed, and then to form the matrix $\left(e^{A/m}\right)^m$ by repeated squaring. One commonly used criterion for choosing m is to make it the smallest power of two for which $||A||/m \leq 1$.

Diagonal Padé approximants ($p = q$) are preferred, since R_{pq} ($p \neq q$) is less accurate than R_{ll}, where $l = max(p, q)$, but R_{ll} can be computed at same cost.

Algorithm 3 (*exmdpa*) computes the exponential of a matrix by means of a scaling-squaring diagonal Padé approximation method with variable order q.

Algorithm 3 $F = exmdpa(A, q)$
Inputs: *Matrix* $A \in \mathbb{R}^{n\times n}$*; order* $q \in \mathbb{N}$ *of diagonal Padé approximation of the exponential function*
Output: *Matrix* $F = e^A \in \mathbb{R}^{n\times n}$

1 $nor = ||A||_\infty$
2 $j_A = \max(0, 1 + int(\log_2(nor)))$

3 $s = \frac{1}{2^{j_A}}$
4 $A = sA$
5 $X = A$
6 $N = I_n + c_1(1)$
7 $D = I_n + c_2(1)$
8 *For* $k = 2 : q$
  8.1 $X = XA$
  8.2 $N = N + c_1(k)X$
  8.3 $D = D + c_2(k)X$
9 *Solve* $DF = N$ *for* F *using Gaussian elimination*
10 *For* $k = 1 : j_A$
  10.1 $F = F^2$

In order to reduce the high computational and storage costs of Algorithm 2, a block oriented version of Algorithm 3 has been developed. In this way, y_{i+1} can be computed without explicitly computing the exponential of matrix $C\Delta t$ (step 2.3 of Algorithm 2). This algorithm only computes blocks (1,2) and (1,3) of the exponential of matrix

$$A = \begin{bmatrix} J_i & I_n & 0_n \\ 0_n & 0_n & I_n \\ 0_n & 0_n & 0_n \end{bmatrix}. \tag{21}$$

With this goal, some steps of Algorithm 3 are adapted to matrices

$$X = \begin{bmatrix} X_{11} & X_{12} & X_{13} \\ 0_n & X_{22} & X_{23} \\ 0_n & 0_n & X_{33} \end{bmatrix},$$

$$N = \begin{bmatrix} N_{11} & N_{12} & N_{13} \\ 0_n & N_{22} & N_{23} \\ 0_n & 0_n & N_{33} \end{bmatrix},$$

$$D = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ 0_n & D_{22} & D_{23} \\ 0_n & 0_n & D_{33} \end{bmatrix},$$

$$F = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0_n & F_{22} & F_{23} \\ 0_n & 0_n & F_{33} \end{bmatrix}.$$

Step 4: $A = sA$.

$$A = sA = \begin{bmatrix} sJ_i & sI_n & 0_n \\ 0_n & 0_n & sI_n \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Let $J_i = sJ_i$, then A can be expressed as

$$A = \begin{bmatrix} J_i & sI_n & 0_n \\ 0_n & 0_n & sI_n \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Step 5: $X = A$.

$$X = \begin{bmatrix} J_i & sI_n & 0_n \\ 0_n & 0_n & sI_n \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Step 6: $N = I_n + c_1(1)A$.

$$\begin{bmatrix} N_{11} & N_{12} & N_{13} \\ 0_n & N_{22} & N_{23} \\ 0_n & 0_n & N_{33} \end{bmatrix} = \begin{bmatrix} I_n + c_1(1)J_i & c_1(1)sI_n & 0_n \\ 0_n & I_n & c_1(1)sI_n \\ 0_n & 0_n & I_n \end{bmatrix},$$

therefore

$$\begin{gathered} N_{11} = I_n + c_1(1)J_i, \\ N_{12} = c_1(1)sI_n, \\ N_{13} = 0_n, \\ N_{22} = I_n, \\ N_{23} = c_1(1)sI_n, \\ N_{33} = I_n. \end{gathered}$$

Step 7: $D = I_n + c_2(1)A$.

$$\begin{bmatrix} D_{11} & D_{12} & D_{13} \\ 0_n & D_{22} & D_{23} \\ 0_n & 0_n & D_{33} \end{bmatrix} = \begin{bmatrix} I_n + c_2(1)J_i & c_2(1)sI_n & 0_n \\ 0_n & I_n & c_2(1)sI_n \\ 0_n & 0_n & I_n \end{bmatrix},$$

therefore

$$\begin{gathered} D_{11} = I_n + c_2(1)J_i, \\ D_{12} = c_2(1)sI_n, \\ D_{13} = 0_n, \\ D_{22} = I_n, \\ D_{23} = c_2(1)sI_n, \\ D_{33} = I_n. \end{gathered}$$

Substep 8.1: $X = XA$.

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ 0_n & 0_n & X_{23} \\ 0_n & 0_n & 0_n \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} & X_{13} \\ 0_n & 0_n & X_{23} \\ 0_n & 0_n & 0_n \end{bmatrix} \begin{bmatrix} J_i & sI_n & 0_n \\ 0_n & 0_n & sI_n \\ 0_n & 0_n & 0_n \end{bmatrix} = \begin{bmatrix} X_{11}J_i & sX_{11} & sX_{12} \\ 0_n & 0_n & 0_n \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Bearing in mind data dependencies, X_{ij}, $1 \leq i \leq j \leq 3$, can be computed as follows

$$\begin{gathered} X_{13} = sX_{12}, \\ X_{12} = sX_{11}, \\ X_{11} = X_{11}J_i, \\ X_{22} = 0_n, \\ X_{23} = 0_n, \\ X_{33} = 0_n. \end{gathered}$$

Substep 8.2: $N = N + c_1(k)X$.

$$\begin{bmatrix} N_{11} & N_{12} & N_{13} \\ 0_n & N_{22} & N_{23} \\ 0_n & 0_n & N_{33} \end{bmatrix} = \begin{bmatrix} N_{11} + c_1(k)X_{11} & N_{12} + c_1(k)X_{12} & N_{13} + c_1(k)X_{13} \\ 0_n & N_{22} & N_{23} \\ 0_n & 0_n & N_{33} \end{bmatrix}.$$

As matrices N_{22}, N_{23} and N_{33} do not vary inside loop 8, then

$$\begin{gathered} N_{22} = I_n, \\ N_{23} = c_1(1)sI_n, \\ N_{33} = I_n. \end{gathered}$$

Substep 8.3: $D = D + c_2(k)X$.

$$\begin{bmatrix} D_{11} & D_{12} & D_{13} \\ 0_n & D_{22} & D_{23} \\ 0_n & 0_n & D_{33} \end{bmatrix} = \begin{bmatrix} D_{11} + c_2(k)X_{11} & D_{12} + c_2(k)X_{12} & D_{13} + c_2(k)X_{13} \\ 0_n & D_{22} & D_{23} \\ 0_n & 0_n & D_{33} \end{bmatrix}.$$

As matrices D_{22}, D_{23} and D_{33} do not vary inside loop 8, then

$$\begin{gathered} D_{22} = I_n, \\ D_{23} = c_2(1)sI_n, \\ D_{33} = I_n. \end{gathered}$$

Step 9: To compute F by solving $DF = N$.

$$\begin{bmatrix} D_{11} & D_{12} & D_{13} \\ 0_n & I_n & c_2(1)sI_n \\ 0_n & 0_n & I_n \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0_n & F_{22} & F_{23} \\ 0_n & 0_n & F_{33} \end{bmatrix} = \begin{bmatrix} N_{11} & N_{12} & N_{13} \\ 0_n & I_n & c_1(1)sI_n \\ 0_n & 0_n & I_n \end{bmatrix},$$

therefore

$$\begin{gathered} D_{11}F_{11} = N_{11}, \\ D_{11}F_{12} + D_{12}F_{22} = N_{12}, \\ D_{11}F_{13} + D_{12}F_{23} + D_{13}F_{33} = N_{13}, \\ F_{22} = I_n, \\ F_{23} + c_2(1)sF_{33} = c_1(1)sI_n, \\ F_{33} = I_n. \end{gathered}$$

Since $c_1(1) = 0.5$ and $c_2(1) = -0.5$, then $F_{23} = sI_n$ and matrices F_{11}, F_{12} and F_{13} can be computed solving the equations

$$\begin{gathered} D_{11}F_{11} = N_{11}, \\ D_{11}F_{12} = N_{12} - D_{12}, \\ D_{11}F_{13} = N_{13} - sD_{12} - D_{13}. \end{gathered}$$

It is only necessary to know N_{11}, N_{12}, N_{13}, D_{11}, D_{12} and D_{13} to compute F_{11}, F_{12} and F_{13} in step 9.

Substep 10.1: $F = F^2$. Making the product

$$\begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0_n & F_{22} & F_{23} \\ 0_n & 0_n & F_{33} \end{bmatrix} = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0_n & F_{22} & F_{23} \\ 0_n & 0_n & F_{33} \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0_n & F_{22} & F_{23} \\ 0_n & 0_n & F_{33} \end{bmatrix},$$

and equaling the corresponding blocks, then

$$\begin{gathered} F_{11} = F_{11}^2, \\ F_{12} = F_{11}F_{12} + F_{12}F_{22}, \\ F_{13} = F_{11}F_{13} + F_{12}F_{23} + F_{13}F_{33}, \\ F_{22} = F_{22}^2, \\ F_{23} = F_{22}F_{23} + F_{23}F_{33}, \\ F_{33} = F_{33}^2. \end{gathered}$$

Bearing in mind that before entering in loop 10 $F_{22} = I_n$ and $F_{33} = I_n$, then inside the above loop $F_{22} = I_n$ and $F_{33} = I_n$, therefore

$$F_{23} = F_{23} + F_{23} = 2F_{23}.$$

Because before entering loop 10 $F_{23} = sI_n$, then

$$F_{23} = 2^k sI_n$$

in the k iteration. According to data dependence, F_{11}, F_{12} and F_{13} can be computed as

$$\begin{gathered} F_{13} = F_{11}F_{13} + F_{12}F_{23} + F_{13}, \\ F_{12} = F_{11}F_{12} + F_{12}, \\ F_{11} = F_{11}^2. \end{gathered}$$

The complete algorithm that solves IVP (1) corresponds to Algorithm 4 (*inolsp*). This algorithm solves IVPs for non-autonomous ODEs by a piecewise-linearized approach with scaling-squaring of the diagonal Padé approximants. This algorithm uses the following auxiliary algorithms:

- Algorithm 5 (*coedpa*) computes the coefficients of the polynomials of degree greater than zero in the diagonal Padé approximation of the exponential function.
- Algorithm 6 (*inlbsp*) computes the approximate solution at t_{i+1} of IVP for non-autonomous ODEs (1), obtained after the piecewise-linearized process, by a block-oriented version of the scaling-squaring diagonal Padé method.

This is the block oriented version of Algorithm 2. The approximate computational cost of Algorithm 6 is $2\left(q+3j_{J\Delta t}+\frac{7}{3}\right)n^3$ flops, where $j_{J\Delta t} = \max(0, 1+int(\log_2(\|J\Delta t\|_\infty)))$.

Algorithm 4 $\{x_i\} = inolsp(data, x_0, t_0, t_f, \Delta t, q)$
Inputs: *Data is a function that computes function vector $f(\tau, x) \in \mathbb{R}^n$, Jacobian matrix $J(\tau, x) \in \mathbb{R}^{n\times n}$ and gradient vector $g(\tau, x) \in \mathbb{R}^n$ ($\tau \in \mathbb{R}$, $x \in \mathbb{R}^n$); vector $x_0 \in \mathbb{R}^n$ of initial conditions; initial time $t_0 \in \mathbb{R}$; final time $t_f \in \mathbb{R}$; step size $\Delta t \in \mathbb{R}$; order $q \in \mathbb{N}$ of the diagonal Padé approximation of the exponential function*
Outputs: *Vectors of solutions $\{x_i\}$ ($x_i \in \mathbb{R}^n$) at $t_0, t_0+\Delta t, t_0+2\Delta t, \ldots$*

1 $[c_1, c_2] = coedpa(q)$ *(Algorithm 5)*
2 $m = \lceil (t_f - t_0)/\Delta t \rceil$
3 For $i = 0 : m-1$
  3.1 $[f, J, g] = data(t_i, x_i)$
  3.2 $x_{i+1} = inlbsp(J, f, g, x_i, \Delta t, c_1, c_2)$ *(Algorithm 6)*
  3.3 $t_{i+1} = t_i + \Delta t$

Algorithm 5 $[c_1, c_2] = coedpa(q)$
Inputs: *Order $q \in \mathbb{N}$ of the diagonal Padé approximation of the exponential function*
Outputs: *Vectors $c_1, c_2 \in \mathbb{R}^q$ with the coefficients of terms greater than 0 in the (q,q) diagonal Padé approximation of the exponential function*

1 $c_1(1) = 0.5$
2 $c_2(1) = -0.5$
3 For $k = 2 : q$
  3.1 $c_1(k) = \frac{q-k+1}{(2q-k+1)k} c_1(k-1)$
  3.2 $c_2(k) = (-1)^k c_1(k)$

Algorithm 6 $y_{i+1} = inlbsp(J, f, g, y_i, \Delta t, c_1, c_2)$
Inputs: *Jacobian matrix $J \in \mathbb{R}^{n\times n}$; function vector $f \in \mathbb{R}^n$; gradient vector $g \in \mathbb{R}^n$; vector $y_i \in \mathbb{R}^n$; step size $\Delta t \in \mathbb{R}$; vectors $c_1, c_2 \in \mathbb{R}^q$ with the coefficients of terms of degree greater than 0 in the (q,q) diagonal Padé approximation of the exponential function*
Output: *Vector $y_{i+1} \in \mathbb{R}^n$ given by expression* (18)

1 $nor = \|J\|_\infty \Delta t$
2 $j_{J\Delta t} = \max(0, 1+int(\log_2(nor)))$; $s = \frac{\Delta t}{2^{j_{J\Delta t}}}$; $J = sJ$
3 $X_{11} = J$; $X_{12} = sI_n$; $X_{13} = 0_n$
4 $N_{11} = I_n + c_1(1)J$;$N_{12} = c_1(1)sI_n$; $N_{13} = 0_n$
5 $D_{11} = I_n + c_2(1)J$;$D_{12} = c_2(2)sI_n$; $D_{13} = 0_n$
6 For $k = 2 : q$
  6.1 $X_{13} = sX_{12}$; $X_{12} = sX_{11}$; $X_{11} = X_{11}J$

6.2 $N_{11} = N_{11} + c_1(k)X_{11}$; $N_{12} = N_{12} + c_1(k)X_{12}$; $N_{13} = N_{13} + c_1(k)X_{13}$
6.3 $D_{11} = D_{11} + c_2(k)X_{11}$; $D_{12} = D_{12} + c_2(k)X_{12}$; $D_{13} = D_{13} + c_2(k)X_{13}$
7 *Solve* $D_{11}F_{11} = N_{11}$ *for* F_{11}
8 *Solve* $D_{11}F_{12} = N_{12} - D_{12}$ *for* F_{12}
9 *Solve* $D_{11}F_{13} = N_{13} - sD_{12} - D_{13}$ *for* F_{13}
10 *For* $k = 1 : j_{J\Delta t}$
10.1 $F_{13} = F_{11}F_{13} + sF_{12} + F_{13}$; $F_{12} = F_{11}F_{12} + F_{12}$; $F_{11} = F_{11}^2$
10.2 $s = 2s$
11 $y_{i+1} = y_i + F_{12}f + F_{13}g$

For autonomous ODEs the computational and storage costs can be reduced if Corollary 1 is applied. Hence, another algorithm (*iaolsp*) can be developed to solve IVPs for autonomous ODEs by a piecewise-linearized approach with scaling-squaring of the diagonal Padé approximants.

This algorithm uses the auxiliary Algorithms 5 (*coedpa*) and 8. Algorithm 8 (*ialbsp*) computes the approximate solution at t_{i+1} of IVP for autonomous ODEs (1), obtained after the piecewise-linearized process, by a block-oriented version of the scaling-squaring diagonal Padé method. The approximate computational cost of Algorithm 8 is $2\left(q + 2j_{J\Delta t} + \frac{4}{3}\right)n^3$ flops, where $j_{J\Delta t} = \max(0, 1 + int(\log_2(\|J\Delta t\|_\infty)))$.

Algorithm 7 $\{x_i\} = iaolsp(data, x_0, t_0, t_f, \Delta t, q)$
Inputs: *Data is a function that computes function vector* $f(\tau, x) \in \mathbb{R}^n$ *and the Jacobian matrix* $J(\tau, x) \in \mathbb{R}^{n\times n}$; *vector* $x_0 \in \mathbb{R}^n$ *of initial conditions; initial time* $t_0 \in \mathbb{R}$; *final time* $t_f \in \mathbb{R}$; *step size* $\Delta t \in \mathbb{R}$; *order* $q \in \mathbb{N}$ *of the diagonal Padé approximation of the exponential function*
Outputs: *Vectors of solutions* $\{x_i\}$ $(x_i \in \mathbb{R}^n)$ *at* $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots$

1 $[c_1, c_2] = coedpa(q)$ *(Algorithm 5)*
2 $m = \lceil (t_f - t_0)/\Delta t \rceil$
3 *For* $i = 0 : m - 1$
3.1 $[f, J, g] = data(t_i, x_i)$
3.2 $x_{i+1} = ialbsp(J, f, x_i, \Delta t, c_1, c_2)$ *(Algorithm 8)*
3.3 $t_{i+1} = t_i + \Delta t$

Algorithm 8 $y_{i+1} = ialbsp(J, f, y_i, \Delta t, c_1, c_2)$
Inputs: *Jacobian matrix* $J \in \mathbb{R}^{n\times n}$; *function vector* $f \in \mathbb{R}^n$; *vector* $y_i \in \mathbb{R}^n$; *step size* $\Delta t \in \mathbb{R}$; *vectors* $c_1, c_2 \in \mathbb{R}^q$ *with the coefficients of terms of degree greater than 0 in the (q,q) diagonal Padé approximation of the exponential function*
Output: *Vector* $y_{i+1} \in \mathbb{R}^n$ *given by expression* (20)

1 $nor = \|J\|_\infty \Delta t$
2 $j_{J\Delta t} = \max(0, 1 + int(\log_2(nor)))$; $s = \frac{\Delta t}{2^{j_{J\Delta t}}}$; $J = sJ$

3 $X_{11} = J$; $X_{12} = sI_n$
4 $N_{11} = I_n + c_1(1)J$; $N_{12} = c_1(1)sI_n$
5 $D_{11} = I_n + c_2(1)J$; $D_{12} = c_2(1)sI_n$
6 *For* $k = 2 : q$
  6.1 $X_{12} = sX_{11}$; $X_{11} = X_{11}J$
  6.2 $N_{11} = N_{11} + c_1(k)X_{11}$; $N_{12} = N_{12} + c_1(k)X_{12}$
  6.3 $D_{11} = D_{11} + c_2(k)X_{11}$; $D_{12} = D_{12} + c_2(k)X_{12}$
7 *Solve* $D_{11}F_{11} = N_{11}$ *for* F_{11}
8 *Solve* $D_{11}F_{12} = N_{12} - D_{12}$ *for* F_{12}
9 *For* $k = 1 : j_{J\Delta t}$
  9.1 $F_{12} = F_{11}F_{12} + F_{12}$
  9.2 $F_{11} = F_{11}^2$
  9.3 $s = 2s$
10 $y_{i+1} = y_i + F_{12}f$

3.2 Algorithms not based on scale-squaring technique

Since matrix C of Algorithm 2 is multiplied by Δt, it is possible compute the approximate solution x_{i+1} without using the scaling-squaring technique. Therefore, the computational costs are reduced without loss of accuracy (see Section 4). Note that in this case it not is necessary to compute block N_{11}. The following algorithms solve IVPs for ODEs by that method:

- Algorithm 9 (*inolwp*) solves the IVP for non-autonomous ODEs (1) by a piecewise-linearized approach without scaling-square of the diagonal Padé approximants.
- Algorithm 10 (*iaolwp*) solves the IVP for autonomous ODEs (1) by a piecewise-linearized approach without scaling-square of the diagonal Padé approximants.

These algorithms use the following auxiliary algorithms:

- Algorithm 11 (*inlbwp*) computes the approximate solution at t_{i+1} of IVP for non-autonomous ODEs (1), obtained after the piecewise-linearized process, by a block-oriented implementation without scaling-squaring of diagonal Padé method. The approximate computational cost of Algorithm 11 is $2\left(q + \frac{4}{3}\right)n^3$ flops.
- Algorithm 12 (*ialbwp*) computes the approximate solution at t_{i+1} of IVP for autonomous ODEs (1), obtained after the piecewise-linearized process, by a block-oriented version without scaling-squaring implementation of the diagonal Padé method. The approximate computational cost of this algorithm is $2\left(q + \frac{1}{3}\right)n^3$ flops.

Figure 1 shows a scheme with the developed piecewise-linearized algorithms.

Algorithm 9 $\{x_i\} = inolwp(data, x_0, t_0, t_f, \Delta t, q)$
Inputs: *Data is a function that computes* $f(\tau, x) \in \mathbb{R}^n$, $J(\tau, x) \in \mathbb{R}^{n\times n}$ *and* $g(\tau, x) \in \mathbb{R}^n$ ($\tau \in \mathbb{R}$, $x \in \mathbb{R}^n$); *vector* $x_0 \in \mathbb{R}^n$ *of initial conditions; initial time* $t_0 \in \mathbb{R}$; *final time* $t_f \in \mathbb{R}$; *step size* $\Delta t \in \mathbb{R}$; *order* $q \in \mathbb{N}$ *of the diagonal Padé approximation of the exponential function*
Outputs: *Vector of solutions* $\{x_i\}$ $(x_i \in \mathbb{R}^n)$ *at* $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots$

1 $[c_1, c_2] = coedpa(q)$ *(Algorithm 5)*
2 $m = \lceil (t_f - t_0)/\Delta t \rceil$
3 *For* $i = 0 : m - 1$
  3.1 $[f, J, g] = data(t_i, x_i)$
  3.2 $x_{i+1} = inlbwp(J, f, g, x_i, \Delta t, c_1, c_2)$ *(Algorithm 11)*
  3.3 $t_{i+1} = t_i + \Delta t$

Algorithm 10 $\{x_i\} = iaolwp(data, x_0, t_0, t_f, \Delta t, q)$
Inputs: *data is a function that computes* $f(\tau, x) \in \mathbb{R}^n$ *and* $J(\tau, x) \in \mathbb{R}^{n\times n}$ ($\tau \in \mathbb{R}$, $x \in \mathbb{R}^n$); *vector* $x_0 \in \mathbb{R}^n$ *of initial conditions; initial time* $t_0 \in \mathbb{R}$; *final time* $t_f \in \mathbb{R}$; *step size* $\Delta t \in \mathbb{R}$; *order* $q \in \mathbb{N}$ *of the diagonal Padé approximation of the exponential function*
Outputs: *Vector of solutions* $\{x_i\}$ $(x_i \in \mathbb{R}^n)$ *at* $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots$

1 $[c_1, c_2] = coedpa(q)$ *(Algorithm 5)*
2 $m = \lceil (t_f - t_0)/\Delta t \rceil$
3 *For* $i = 0 : m - 1$
  3.1 $[f, J, g] = data(t_i, x_i)$
  3.2 $x_{i+1} = ialbwp(J, f, x_i, \Delta t, c_1, c_2)$ *(Algorithm 12)*
  3.3 $t_{i+1} = t_i + \Delta t$

Algorithm 11 $y_{i+1} = inlbwp(J, f, g, y_i, \Delta t, c_1, c_2)$
Inputs: *Jacobian matrix* $J \in \mathbb{R}^{n\times n}$; *function vector* $f \in \mathbb{R}^n$; *gradient vector* $g \in \mathbb{R}^n$; *vector* $y_i \in \mathbb{R}^n$; *step size* $\Delta t \in \mathbb{R}$; *vectors* $c_1, c_2 \in \mathbb{R}^q$ *with the coefficients of terms of degree greater than 0 in the (q,q) diagonal Padé approximation of the exponential function*
Output: *Vector* $y_{i+1} \in \mathbb{R}^n$ *given by expression* (18)

1 $J = \Delta t J$
2 $X_{11} = J$; $X_{12} = \Delta t I_n$; $X_{13} = 0_n$
3 $N_{12} = c_1(1) I_n$; $N_{13} = 0_n$
4 $D_{11} = I_n + c_2(1) J$;$D_{12} = c_2(2) s I_n$; $D_{13} = 0_n$
5 *For* $k = 2 : q$
  5.1 $X_{13} = sX_{12}$; $X_{12} = sX_{11}$; $X_{11} = X_{11} J$
  5.2 $N_{12} = N_{12} + c_1(k) X_{12}$; $N_{13} = N_{13} + c_1(k) X_{13}$
  5.3 $D_{11} = D_{11} + c_2(k) X_{11}$; $D_{12} = D_{12} + c_2(k) X_{12}$; $D_{13} = D_{13} + c_2(k) X_{13}$
6 *Solve* $D_{11} F_{12} = N_{12} - D_{12}$ *for* F_{12}
7 *Solve* $D_{11} F_{13} = N_{13} - sD_{12} - D_{13}$ *for* F_{13}

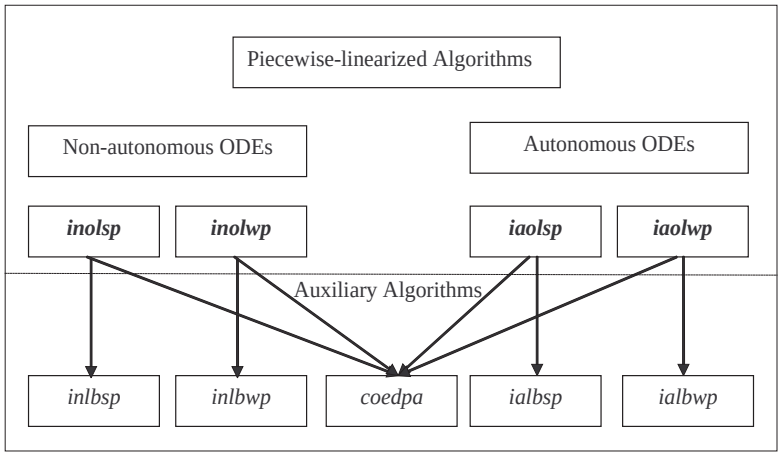

Fig. 1. Schema of piecewise-linearized Algorithms 4($inolsp$), 9($inolwp$), 7($iaolsp$) and 10($iaolwp$)

8 $y_{i+1} = y_i + F_{12}f + F_{13}g$

Algorithm 12 $y_{i+1} = ialbwp(J, f, y_i, \Delta t, c_1, c_2)$
Inputs: *Jacobian matrix* $J \in \mathbb{R}^{n\times n}$*; function vector* $f \in \mathbb{R}^n$*; vector* $y_i \in \mathbb{R}^n$*; step size* $\Delta t \in \mathbb{R}$*; vectors* $c_1, c_2 \in \mathbb{R}^q$ *with the coefficients of terms of degree greater than 0 in the (q,q) diagonal Padé approximation of the exponential function*
Output: *Vector* $y_{i+1} \in \mathbb{R}^n$ *given by expression* (20)

1 $J = \Delta t J$
2 $X_{11} = J$*;* $X_{12} = \Delta t I_n$
3 $D_{11} = I_n + c_2(1)J$
4 $D_{12} = c_2(1)sI_n$
5 For $k = 2 : q$
  5.1 $X_{12} = sX_{11}$*;* $X_{11} = X_{11}J$
  5.2 $N_{12} = N_{12} + c_1(k)X_{12}$
  5.3 $D_{11} = D_{11} + c_2(k)X_{11}$*;* $D_{12} = D_{12} + c_2(k)X_{12}$
6 Solve $D_{11}F_{12} = N_{12} - D_{12}$ *for* F_{12}
7 $y_{i+1} = y_i + F_{12}f$

4 Experimental results

The main objective of this section is to compare the algorithms developed in Sections 2 and 3. What follows is a short description of the characteristic parameters for the implemented algorithms:

- *iaolsp* and *inolsp* solve IVPs for ODEs by means of a piecewise-linearized approach and a block-oriented version with scaling-squaring implementation

of the diagonal Padé approximation method:
  · Order (q) of the diagonal Padé approximation of the exponential function.
- *iaolwp* and *inolwp* solve IVPs for ODEs by means of a piecewise-linearized approach and a block-oriented version without scaling-squaring implementation of the diagonal Padé approximation method:
  · Order (q) of the diagonal Padé approximation of the exponential function.
- *iodbcs* solves IVPs for ODEs by means of a BDF method based on a Chord-Shamanskii iteration:
  · Order (r) of BDF method.
  · Relative error tolerance (tol_1) and absolute error tolerance (tol_2).
  · Maximum number of iterations without computing the Jacobian matrix (m).
  · threshold (ρ).

As test battery five case studies were considered. The criteria to select these cases studies were:

- To solve physics and physical chemistry problems by the developed algorithms (case studies 1, 2, 3 and 5). Case study 4 has been selected because it has a known analytic solution.
- To prove the implementations on autonomous and non-autonomous ODEs: three IVPs for autonomous ODEs and two IVPs for non-autonomous ODEs have been selected.
- To compare the implementations when the ODE is stiff (cases studies 1, 2, 3 and 5) or non-stiff (case study 4).

Numerous tests were made (for each case study the characteristic parameters were varied, although only the parameters which offered better accuracy and lower computational cost for each algorithm are presented). Three kinds of tests are shown:

- Variable step size.
- Variable final time.

For each test, the following results are shown:

- Tables which contain the relative error

$$E_r = \frac{\|x - x^*\|_\infty}{\|x\|_\infty},$$

  where x^* is the computed solution and x is the analytic solution (case study 4) or the solution computed by the MATLAB function ode15s with a vector of relative error tolerances $rtol = 10^{-13}$ and a vector of absolute error tolerances $atol = 10^{-13}$ [14] (case studies 1, 2, 3 and 5).
- Tables with the execution time.

All algorithms were implemented in MATLAB and Fortran. The MATLAB implementations were tested on an Intel Core 2 Duo processor at 1.83 GHz with 2 GB main memory, using MATLAB version 7.5. The tests for the larger dimension problem (case study 5) were carried out on a SGI Altix 3700 node [15], with 1.3 GHz Intel Itanium II with 3 MB cache. In this case the algorithms were implemented in FORTRAN using the mathematical libraries BLAS [16] and LAPACK [17]. The implementations were compiled with Intel FORTRAN compiler (release 8.1) and SGI SCSL (Scientific Computing Software Library) mathematical library (release 1.5.1) was used. The SCSL is an optimized version of BLAS and LAPACK for SGI systems. The implemented algorithms are available online at [18]. The case studies considered in this work are presented below.

4.1 Case study 1 (Chemical Akzo Nobel problem)

This case study corresponds to the stiff autonomous ODE [19] defined by

$$\begin{aligned} \dot{x} = f(x), x = x(t) \in \mathbb{R}^6, 0 \leq t \leq 180, \\ x(0) = [0.444,\ 0.00123,\ 0,\ 0,\ 0.007,\ 0.35999964]^T, \end{aligned}$$

where

$$f(x) = \begin{bmatrix} -2r_1 + r_2 - r_3 - r_4 \\ -0.5r_1 - r_4 - 0.5r_5 + F_{in} \\ r_1 - r_2 + r_3 \\ -r_2 - r_3 - 2r_4 \\ r_2 - r_3 + 2r_5 \\ -r_5 \end{bmatrix},$$

and r_i, $i = 1, 2, 3, 4, 5$, and F_{in} are defined as

$$\begin{aligned} r_1 &= k_1 x_1^4 x_2^{0.5}, \\ r_2 &= k_2 x_3 x_4, \\ r_3 &= \frac{k_2}{K} x_1 x_5, \\ r_4 &= k_3 x_1 x_4^2, \\ r_5 &= k_4 x_6^2 x_2^{0.5}, \\ F_{in} &= klA \left(\frac{p(\mathrm{CO}_2)}{H} - x_2 \right). \end{aligned}$$

Er	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	2.576e-5	1.223e-5	8.658e-7	2.323e-7	9.823e-9
iaolsp	2.080e-5	5.238e-6	1.485e-7	3.851e-8	1.588e-9
iaolwp	8.100e-6	2.824e-6	1.485e-7	3.851e-8	1.588e-9

Table 2
Relative errors considering t_f=60 and varying Δt (case study 1)

The tests were carried out considering:

$$k_1 = 18.7, \quad K = 34.4,$$
$$k_2 = 0.58, \quad klA = 3.3,$$
$$k_3 = 0.09, \; p(\mathrm{CO}_2) = 0.9$$
$$k_4 = 0.42, \quad H = 737.$$

This problem originates from Akzo Nobel Central Research in Arnhem. It describes a chemical process in which two species, FLB and ZHU, are mixed, while carbon dioxide is continuously added. The variables x_i correspond to the following concentrations: x_1 = [FLB], x_2 = [CO_2], x_3 = [FLBT], x_4 = [ZHU], x_5 = [ZLA] and x_6 = [ZLA.ZHU], where FLBT, ZLA and ZLA.ZHU are other species that appear in the chemical process.

The optimal values of characteristic parameters were:

- *iodbcs*: r=3, tol_1=tol_2=10^{-14}, m=2, $\rho = 0.5$.
- *iaolsp*: q=1.
- *iaolwp*: q=1.

The following tests were done:

- First test (Tables 2 and 3): t_f=60 and Δt variable.
- Second test (Table 4 and Figure 2): Δt=0.01 and t_f variable.

Conclusions for this case study:

- Considering the same step size, the implementations based on the piecewise-linearized approach have lower relative error and lower execution time than the implementation based on BDF approach.
- The implementation based on the piecewise-linearized method and on the diagonal Padé approach without scaling-squaring (*iaolwp*) has the shorter execution time.

Execution time (seconds)	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	0.196	0.253	1.275	2.545	11.43
iaolsp	0.087	0.153	0.745	1.493	7.462
iaolwp	0.058	0.114	0.570	1.138	5.692

Table 3
Execution time of the MATLAB implementations considering t_f=60 and varying Δt (case study 1)

Er	t_f=60	t_f=90	t_f=120	t_f=150	t_f=180
iodbcs	8.658e-7	5.388e-7	4.183e-7	3.607e-7	3.303e-7
iaolsp	1.485e-7	9.687e-8	7.838e-8	6.980e-8	6.546e-8
iaolwp	1.485e-7	9.687e-8	7.838e-8	6.980e-8	6.546e-8

Table 4
Relative errors considering Δt=0.01 and varying t_f (case study 1)

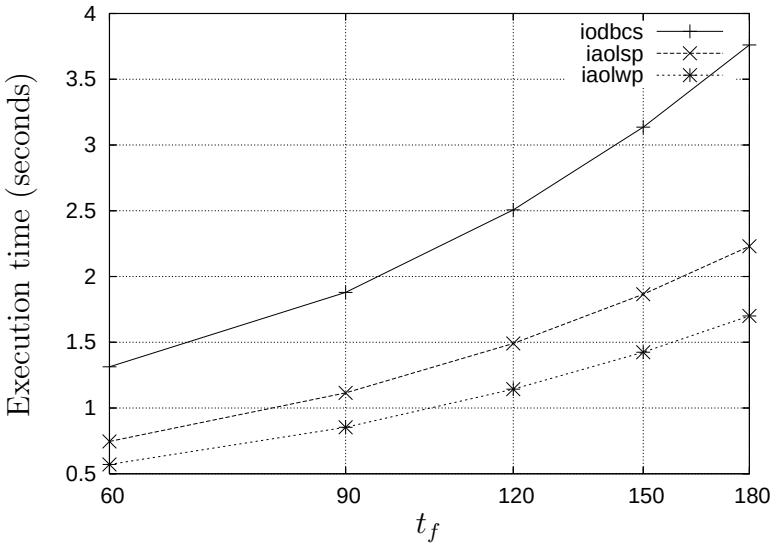

Fig. 2. Execution time of the MATLAB implementations considering $\Delta t = 0.01$ and varying t_f (case study 1)

4.2 Case study 2 (HIRES problem)

This case study is presented in [19] and it corresponds to the stiff IVP for ODEs defined by

$$\dot{x} = f(x), x = x(t) \in \mathbb{R}^8, 0 \leq t \leq 321.8122,$$
$$x(0) = [1, 0, 0, 0, 0, 0, 0, 0.0057]^T,$$

Er	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	2.136e-4	5.279e-5	1.933e-6	4.767e-7	1.885e-8
iaolsp	4.185e-5	1.147e-5	4.8495e-7	1.219e-7	4.899e-9
iaolwp	4.183e-5	1.147e-5	4.8495e-7	1.219e-7	4.899e-9

Table 5
Relative errors considering t_f=50 and varying Δt (case study 2)

where

$$f(x) = \begin{pmatrix} -1.71x_1 + 0.43x_2 + 8.32x_3 + 0.0007 \\ 1.71x_1 - 8.75x_2 \\ -10.03x_3 + 0.43x_4 + 0.035x_5 \\ 8.32x_2 + 1.71x_3 - 1.12x_4 \\ -1.745x_5 + 0.43x_6 + 0.43x_7 \\ -280x_6x_8 + 0.69x_4 + 1.71x_5 - 0.43x_6 + 0.69x_7 \\ 280x_6x_8 - 1.81x_7 \\ -280x_6x_8 + 1.81x_7 \end{pmatrix}.$$

The name HIRES was given by Hairer and Wanner [1]. The HIRES problem explains the "High Irradiance Responses" (HIRES) of phytochrome-mediated photomorphogenesis by means of a chemical reaction involving eight reactants. The variables x_i are the concentrations of the eight reactants.

The optimal values of characteristic parameters were:

- *iodbcs*: r=3, tol_1=tol_2=10^{-14}, m=2, $\rho = 0.5$.
- *iaolsp*: $q = 2$.
- *iaolwp*: q=2.

The following tests were done:

- First test (Tables 5 and 6): t_f=50 and Δt variable.
- Second test (Table 7 and Figure 3): Δt=0.01 and t_f variable.

The conclusions obtained in this case are the same as for case study 1.

Execution time (seconds)	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	0.199	0.300	1.521	2.471	10.258
iaolsp	0.137	0.250	0.991	1.819	9.098
iaolwp	0.069	0.139	0.709	1.316	6.578

Table 6
Execution time of the MATLAB implementations considering t_f=50 and varying Δt (case study 2)

Er	t_f=100	t_f=150	t_f=200	t_f=250	t_f=300
iodbcs	2.294e-6	2.989e-6	4.276e-6	7.425e-6	2.406e-5
iaolsp	5.753e-7	7.496e-7	1.072e-6	1.862e-6	6.041e-6
iaolwp	5.753e-7	7.496e-7	1.072e-6	1.862e-6	6.041e-6

Table 7
Relative errors considering Δt=0.01 and varying t_f (case study 2)

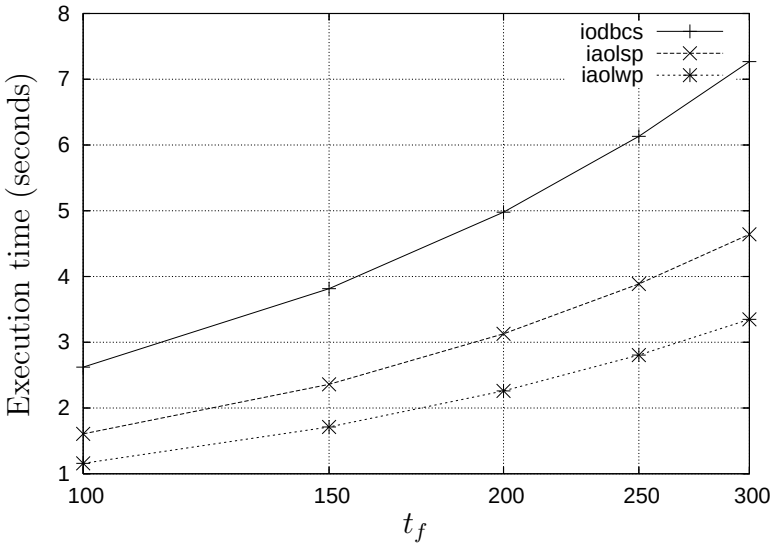

Fig. 3. Execution time of the MATLAB implementations considering $\Delta t = 0.01$ and varying t_f (case study 2)

4.3 Case study 3

This case study corresponds to the stiff autonomous ODE [14, pp. 29] defined by

$$\dot{x} = f(x),\ x = x(t) \in \mathbb{R}^3,\ 0 \leq t \leq 8 \cdot 10^5,$$
$$x(0) = [0, 1, 0]^T,$$

where

$$f([x_1, x_2, x_3]^T) = \begin{pmatrix} -k_1 x_1 + k_2 x_3 \\ -k_4 x_2 + k_3 x_3 \\ k_1 x_1 + k_4 x_2 - (k_1 + k_3) x_3 \end{pmatrix},$$

and

$$k_1 = 8.4303270 \cdot 10^{-10},\ k_2 = 2.9002673 \cdot 10^{11},$$
$$k_3 = 2.4603642 \cdot 10^{10},\ k_4 = 8.7600580 \cdot 10^{-6}.$$

This case study corresponds to the proton transfer hydrogen-hydrogen bond problem, where $x_1(t)$ and $x_2(t)$ are the solution components of the proton transfer and $x_3(t)$ is the quickly reacting intermediate component.

The optimal values of characteristic parameters were:

- *iodbcs*: r=2, tol_1=tol_2=10^{-14}, m=2, $\rho = 0.5$.
- *iaolsp*: $q = 1$.
- *iaolsp*: q=1.

The following tests were done:

- First test (Table 8 and Figure 9): t_f=100 and Δt variable. Since with small values of Δt very high precisions have been reached, only $\Delta t = 0.1$ and $\Delta t = 0.05$ have been considered.
- Second test (Table 10 and Figure 4): Δt=0.01 and t_f variable.

Conclusions for this case study:

- For the same step size, the relative errors committed by the implementation based on the piecewise-linearized method and on the diagonal Padé approach without scaling-squaring (*iaolwp*) have been minor in comparison to those committed by the other two implementations; also this implementation has the shortest execution time.
- The implementation based on the BDF method (*iodbcs*) has a lower relative error than the implementation based on the piecewise-linearized method and on the diagonal Padé approach with scaling-squaring (*iaolsp*), but execution time is longer.

Er	Δt=0.1	Δt=0.05
iodbcs	4.104e-14	8.202e-14
iaolsp	4.706e-12	2.358e-12
iaolwp	6.274e-15	6.065e-15

Table 8
Relative errors considering t_f=100 and varying Δt (case study 3)

Execution time (seconds)	Δt=0.1	Δt=0.05
iodbcs	0.199	0.300
iaolsp	0.137	0.250
iaolwp	0.069	0.139

Table 9
Execution time of the MATLAB implementations considering t_f=100 and varying Δt (case study 3)

Er	t_f=500	t_f=1000	t_f=1500	t_f=2000	t_f=2500
iodbcs	4.359e-12	8.787e-12	1.420e-12	1.995e-11	2.6616e-11
iaolsp	4.636e-12	9.071e-12	1.538e-12	2.130e-11	2.666e-11
iaolwp	3.838e-14	1.303e-13	1.927e-12	3.960e-12	5.814e-12

Table 10
Relative errors considering Δt=0.01 and varying t_f (case study 3)

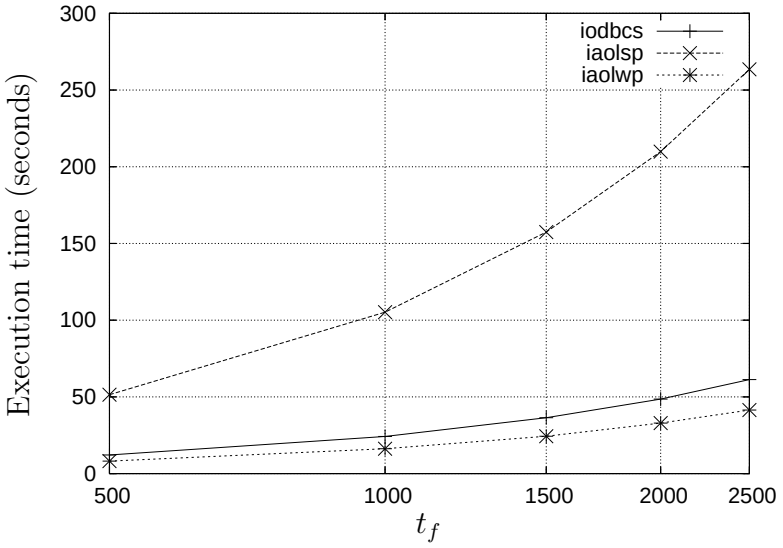

Fig. 4. Execution time of the MATLAB implementations considering $\Delta t = 0.01$ and varying t_f (case study 3)

Er	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	5.167e-06	1.171e-06	4.103e-08	1.009e-08	3.971e-10
inolsp	1.079e-15	1.447e-15	6.296e-15	2.268e-14	4.965e-14
inolwp	1.079e-15	1.447e-15	6.296e-15	2.268e-14	4.965e-14

Table 11
Relative errors considering t_f=10 and varying Δt (case study 4)

4.4 Case study 4

This case study corresponds to the non-stiff and non-autonomous ODE [6] defined as

$$\dot{x}(t) = (t - x(t))^2 + 1,\ t \geq 3,$$
$$x(3) = 2.$$

The analytic solution is

$$x(t) = t + \frac{1}{2 - t}.$$

The values chosen for the characteristic parameters were:

- *iodbcs*: r=2, tol_1=tol_2=10^{-12}, m=2, $\rho = 0.5$.
- *inolsp*: q=1.
- *inolwp*: q=1.

The following tests were done:

- First test (Tables 11 and 12): t_f=10 and Δt variable. Since the piecewise-linearized method presents a very small error for a step size equal to 0.1, we will show only the results obtained for that increase.
- Second test (Table 13 and Figure 5): Δt=0.1 and t_f is variable.

Conclusions for this case study:

- For the same step size, the implementation based on the BDF method *iodbcs* shows more significant error than the implementations based on the piecewise-linearized method.
- The implementation based on the piecewise-linearized method and on the diagonal Padé approach without scaling-squaring (*inolwp*) has the shortest execution time.

Execution time (seconds)	Δt=0.1	Δt=0.05	Δt=0.01	Δt=0.005	Δt=0.001
iodbcs	0.015	0.027	0.071	0.121	0.609
inolsp	0.010	0.017	0.046	0.092	0.456
inolwp	0.008	0.015	0.040	0.080	0.400

Table 12
Execution time the MATLAB implementations for t_f=10 and Δt (case study 4)

Er	t_f=100	t_f=200	t_f=300	t_f=400	t_f=500
iodbcs	1.192e-08	1.460e-09	4.295e-10	1.807e-10	9.228e-11
inolsp	1.236e-14	1.904e-14	1.762e-14	5.032e-14	6.209e-14
inolwp	1.236e-14	1.904e-14	1.762e-14	5.032e-14	6.209e-14

Table 13
Relative errors considering Δt=0.1 and varying t_f (case study 4)

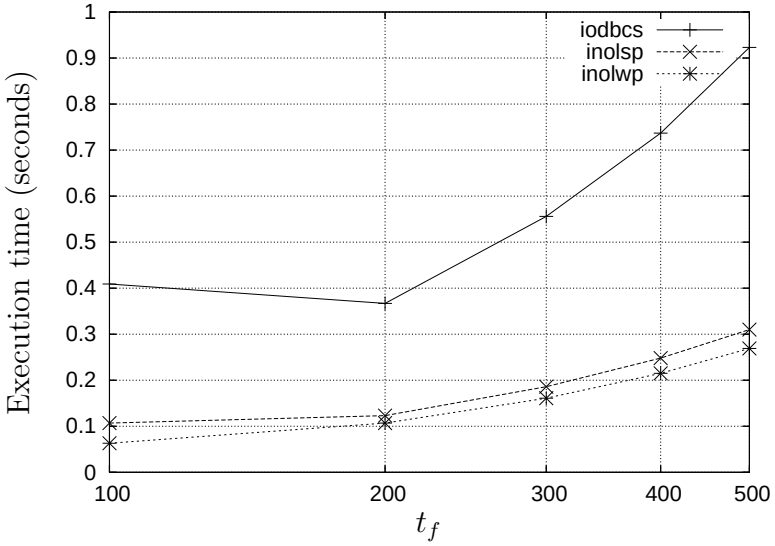

Fig. 5. Execution time of the MATLAB implementations considering $\Delta t = 0.1$ and varying t_f (case study 4)

4.5 Case study 5 (Medical Akzo Nobel problem)

This case study corresponds to a stiff non-autonomous ODE [19] defined as

$$\dot{x} = f(t,x)\,, t \geq 0, x = x(t) \in \mathbb{R}^{2N},\ 0 \leq t \leq 20,$$

$$x(0) = [0, v_0, 0, v_0, \cdots, 0, v_0] \in \mathbb{R}^{2N},\ N = 200.$$

Function f is defined as follows

$$\begin{aligned} f_{2j-1} &= \alpha_j \frac{x_{2j+1} - x_{2j-3}}{2\Delta\zeta} + \beta_j \frac{x_{2j-3} - 2x_{2j-1} + x_{2j+1}}{(\Delta\zeta)^2} - k x_{2j-1} x_{2j}, \\ f_{2j} &= -k x_{2j} x_{2j-1}, \end{aligned}$$

where

$$\begin{aligned} \alpha_j &= \frac{2(j\Delta\zeta - 1)^3}{c^2}, \\ \beta_j &= \frac{(j\Delta\zeta - 1)^4}{c^2}. \end{aligned}$$

The values of j are from 1 to$N$, $\Delta\zeta = \frac{1}{N}$,$x_{-1}(t) = \phi(t)$, $x_{2N+1} = x_{2N-1}$ and ϕ function is given by

$$\phi(t) = \begin{cases} 2, \ t \in (0, 5] \\ 0, \ t \in (5, 20] \end{cases}.$$

The Akzo Nobel research laboratories formulated this problem in their study of the penetration of radio-labeled antibodies into a tissue infected by a tumor. This study was carried out for diagnostic and therapeutic purposes. The desired results are the evolutions of the concentrations u and v of both elements (the antibodies and the tissue, respectively) along the discretized space in function of time. This problem can be formulated as the above ODE by using the lines method. In this formulation, the data vector is noted as x, where elements x_{2j-1}, $j = 1, \ldots, N$, represent concentrations u along the spatial dimension and elements x_{2j}, $j = 1, \ldots, N$, represent concentrations v. The chosen values of k, v_0 and c were 100, 1 and 4 respectively.

The optimal values of characteristic parameters were:

- *iodbcs*: r=2, tol_1=tol_2=10^{-14}, m=2, $\rho = 0.5$.
- *inolsp*: q=1.
- *inolwp*: q=1.

Figure 6 summarizes the execution time of the Fortran implementation of the algorithms considered in this work, considering $\Delta t = 10^{-6}$ and varying t. The relative error in all algorithms was approximately equal to 10^{-6}. In this case, the implementation based on the piecewise-linearized method and on the diagonal Padé approach without scaling-squaring (*inolwp*) has the shortest execution time.

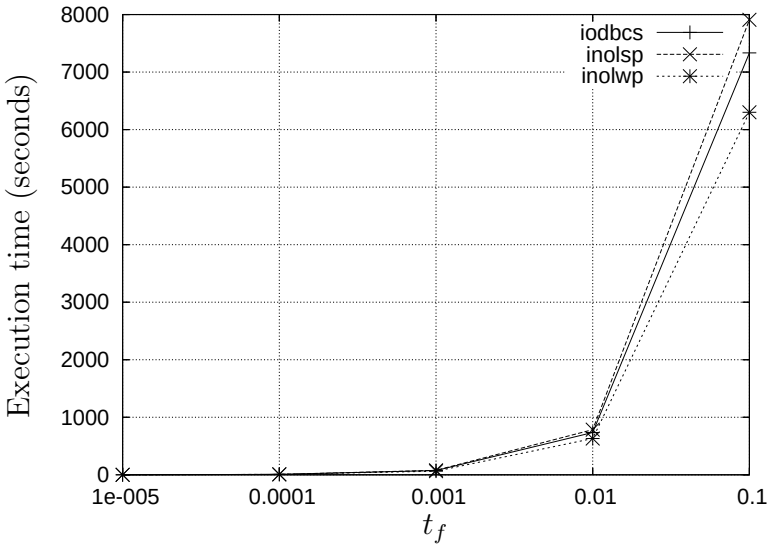

Fig. 6. Execution time of the Fortran implementations considering $\Delta t = 10^{-6}$ and varying t_f (case study 5)

5 Conclusions and future work

In this paper three methods for solving IVPs for ODEs have been developed and implemented. The first is a BDF method based on a Chord and Shamanskii iteration. The other two methods are based on the piecewise-linearized method and the diagonal Padé approximants. These methods are based on Theorem 1, which makes it possible to solve IVPs for ODEs. In addition, four algorithms, two for non-autonomous ODEs ($inolsp - inolwp$) and another two for autonomous ODEs ($iaolsp - iaolwp$), have been implemented. These algorithms have been compared to the BDF algorithm based on Chord-Shamanskii iteration (*idobcs*). According to experimental results, the algorithms based on the piecewise-linearized method and on the diagonal Padé approach without scaling-squaring ($iaolwp - inolwp$) behave better, both in terms of precision and computational costs, than the BDF algorithm (see Tables 14 and 15). Below is a summary of the five case studies:

(1) The optimal order of the BDF algorithm varies between 1 and 3, depending on the case study.
(2) The optimal order of algorithms based on diagonal Padé approximants varies between 1 (case studies 1, 3, 4 and 5) and 2 (case study 2).
(3) Considering the same step size, the BDF algorithm (*idobcs*) is generally less accurate than the piecewise-linearized algorithms. The more accurate algorithms are based on the diagonal Padé approximants without scaling-squaring (*iaolwp* and *inolwp* algorithms). Also these algorithms have the lowest computational cost.
(4) All implemented algorithms show good behavior in stiff ODEs.

Accuracy	1-A-S	2-A-S	3-A-S	4 NA-NS	5 NA-S
iodbcs	−	−		−	≅
iaolsp-inolsp		+	−	+	≅
iaolwp-inolwp	+	+	+	+	≅

Table 14
Comparative precision of implemented algorithms for the five case studies: The symbols +, − and ≅ indicate respectively greater, less and similar precision. A/NA denotes Autonomous/Non-Autonomous ODEs, and S/NS denotes whether the problem is Stiff or Non-Stiff.

Execution times	1-A-S	2-A-S	3-A-S	4 NA-NS	5 NA-S
iodbcs	+	+		+	
iaolsp-inolsp			+		+
iaolwp-inolwp	−	−	−	−	−

Table 15
Comparative execution times of the implemented algorithms for the five case studies: The symbols +, − indicate respectively greater and less execution time. A/NA denotes Autonomous/Non-Autonomous ODEs, and S/NS denotes whether the problem is Stiff or Non-Stiff.

As future work, new improvements will be developed, such as:

(1) To consider adapting the methodology described here in order to obtain efficient resolution of ODEs with sparse Jacobian matrices.
(2) To implement algorithms with error control in order to vary step size dynamically. The tests reported here considered constant step size. It is possible to improve the developed algorithms using a step size variable by estimating the error committed in each iteration [10].
(3) To do parallel implementation of the algorithms presented in this work in a distributed memory platform, using the message passing paradigm, MPI [20] and BLACS [21] for communications, and PBLAS [22] and ScaLAPACK [23] for computations.

References

[1] E. Hairer, G. Wanner, Solving ordinary differential equations II. Stiff and differential-algebraic problems., in: Springer Series in Computational Mathematics, Vol. 14, Springer-Verlag, 1996.

[2] C. F. Curtiss, J. O. Hirschfelder, Integration of stiff equations, in: Proc. Nat. Acad. Sci., Vol. 38, 1952, pp. 235–243.

[3] A. C. Hindmarsh, Lsode and lsodi, two new initial value ordinary differential equation solvers, ACM-Signum Newslett 15 (1980) 1011.

[4] C. W. Gear, Algorithm 407-difsub for solution of ordinary differential equations, Comm. ACM 14 (1971) 185190.

[5] P. N. Brown, G. D. Byrne, A. C. Hindmarsh, Vode: A variable-coefficient ode solver, SIAM J. Sci. Statist. Comput. 10 (1989) 10381051.

[6] J. I. Ramos, C. M. García-López, Piecewise-linearized methods for initial-value problems, Applied Mathematics and Computation 82 (1997) 273–302.

[7] C. M. García-López, Métodos de linealización para la resolución numérica de ecuaciones diferenciales, Ph.D. thesis, Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga (1998).

[8] U. M. Ascher, L. R. Petzold, Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, SIAM, 1998.

[9] C. T. Kelley, Iterative Methods for Linear and Nonlinear Equations, Philadelphia, 1995.

[10] C. M. García-López, Piecewise-linearized and linearized θ-methods for ordinary and partial differential equation problems, Computer & Mathematics with Applications 45 (2003) 351–381.

[11] G. H. Golub, C. V. Loan, Matrix Computations, 3rd Edition, Johns Hopkins Studies in Mathematical Sciences, The Johns Hopkins University Press, 1996.

[12] C. B. Moler, C. V. Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Review 45 (2003) 3–49.

[13] N. J. Higham, The scaling and squaring method for the matrix exponential revisited, Tech. Rep. 452, Manchester Centre for Computational Mathematics (2004).

[14] L. S. Shampine, I. Gladwell, S. Thomson, Solving ODEs with MAtlab, Cambridge University Press, 2003.

[15] S. Graphics, SGI Altix Applications Development an Optimization, Release August 1 Edition (2003).

[16] J. Dongarra, J. D. Croz, S. Hammarling, R. J. Hanson, An extended set of FORTRAN Basic Linear Algebra Subroutines, ACM Transactions on Mathematical Software.

[17] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, LAPACK Users' Guide, SIAM, 1992.

[18] `http://www.grycap.upv.es/odelin/`.

[19] W. M. Lioen, J. J. B. de Swart, Test set for initial value problems solvers, release 2.0 (December 1998).

[20] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, 1994.

[21] J. J. Dongarra, R. A. V. D. Geijn, Two dimensional basic linear algebra communications subprograms, Tech. rep., Department of Computer Science, University of Tennessee (1991).

[22] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, A proposal for a set of parallel basic linear algebra subprogram, Tech. Rep. UT-CS-95-292, Department of Computer Science, University of Tennessee (1995).

[23] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, ScaLAPACK Users' Guide, SIAM, 1997.