Punjab University
Journal of Mathematics (ISSN 1016-2526)
Vol. 51(2)(2019) pp. 1-19

Numerical Approximation of Stiff Problems Using Efficient Nested Implicit Runge–Kutta Methods

Attique Ur-Rehman*
Department of Mathematics,
Government College University Lahore, Pakistan.
Email: attique.ur-rehman@gcu.edu.pk*

Shafiq Ur Rehman
Department of Mathematics,
University of Engineering and Technology, Lahore, Pakistan.
Email: srehman@uet.edu.pk

Junaid Ahmad
Department of Mathematics,
University of Engineering and Technology, Lahore, Pakistan.
Email: junaidshaju@gmail.com

Received: 02 March, 2018 / Accepted: 01 June, 2018 / Published online: 10 December, 2018

Abstract. The aim of this paper is to investigate the accuracy and efficiency of Gauss-type nested implicit Runge–Kutta (NIRK) methods. These methods possess many important practical properties of implicit Runge–Kutta (IRK) methods, such as, good stability, high order, and symmetry. Moreover, as with IRK methods, NIRK methods can also be presented in embedded form that helps local error estimation. However, the implementation of NIRK methods is practically cheaper than other classes of IRK methods as the NIRK formulas have explicit internal stages.

We test an order six NIRK method that uses an embedded order four method for estimating the local error. Our test problems are those from stiff DETEST. As part of the testing, we compare the method with the Matlab integrator ODE15s and assess the effectiveness of four variations on the local error estimation.

AMS (MOS) Subject Classification Codes: 65L04; 65L05; 65L06
Key Words: Gauss-type methods, Nested implicit Runge-Kutta methods, Stiff problems, Local error estimation.

1. Overview

Many systems of ordinary differential equations (ODEs) model various physical phenomena from science, engineering and economics. The solution of these ordinary differential equations tells us about the behaviour of the underlying physical system. In order to get a unique solution of a differential equation system, some additional information is required in the form of initial conditions (ICs). The initial value problems (IVPs) naturally arise in many physical phenomena and are generally written as

$$y'(t) = f(y(t)), \quad y(t_0) = y_0, \tag{1. 1}$$

where $y_0 \in \mathbb{R}^N$ denote the initial positions, the operator $'$ denotes differentiation with respect to time, N is the dimension of the IVP, and $f : \mathbb{R}^N \to \mathbb{R}^N$ is a sufficiently smooth function. It is not always possible to find analytical solutions for many IVPs. We therefore resort to numerical approximations of the exact solutions using numerical methods. These numerical methods can be classified as one step methods or multistep methods. In this paper, we will only consider the one step methods.

In one step methods, the numerical solution advances in time from t_k to t_{k+1} by using information for the previous time t_k only. The first one step method was the famous method of Euler, introduced by Euler in 1768 that was published in his three volume work *Institutiones Calculi Integralis*. Heun (1900) and Kutta (1901) further contributed and characterized the methods that later became Runge-Kutta (RK) methods. The general form of RK methods is

$$Y_i = y_k + h\sum_{j=1}^{s} a_{ij} f(t_k + c_j h, Y_j), \quad i = 1, 2, ..., s, \tag{1. 2 a}$$

$$y_{k+1} = y_k + h\sum_{i=1}^{s} b_i f(t_k + c_i h, Y_i), \tag{1. 2 b}$$

where $h = t_{k+1} - t_k$ is the step size and Y_i are the stage values. Early focus was on explicit Runge-Kutta (ERK) methods but with recognition of stiff ODEs, attention moved towards implicit Runge-Kutta (IRK) methods as well.

The IRK methods have several advantages that make them a popular choice. They include special classes of methods that have excellent stability properties. Stability of a numerical method is connected with the ability of the method to control the propagation of the error introduced in initial approximations. Furthermore, IRK methods can have high order of convergence and can be implemented in a variable step size regime.

Suppose we are trying to solve an N–dimensional system of ODEs $y' = f(t, y(t))$ with an s–stage fully IRK method. We need to solve a system of sN nonlinear equations for the unknown stages $Y_1, Y_2, \ldots, Y_s$. These stage values can be formulated as

$$\begin{aligned}
Y_1 - y_0 - h\Big(a_{11} f(t_0 + c_1 h, Y_1) + \ldots + a_{1s} f(t_0 + c_s h, Y_s)\Big) &= 0,\\
Y_2 - y_0 - h\Big(a_{21} f(t_0 + c_1 h, Y_1) + \ldots + a_{2s} f(t_0 + c_s h, Y_s)\Big) &= 0,\\
&\vdots\\
Y_s - y_0 - h\Big(a_{s1} f(t_0 + c_1 h, Y_1) + \ldots + a_{ss} f(t_0 + c_s h, Y_s)\Big) &= 0.
\end{aligned} \tag{1. 3}$$

The above system can be written in a more compact form as

$$Y_i - y_k - h\sum_{j=1}^{s} a_{ij} f(t_k + c_j h, Y(j)) = 0, \quad i = 1, 2, \ldots, s. \tag{1. 4}$$

The main issue in the implementation of an IRK method is to achieve an efficient solution to the above nonlinear system. Let $g_i = Y_i - y_k$, then (1. 4) implies

$$g_i = h\sum_{j=1}^{s} a_{ij} f(t_k + c_j h, y_k + g_j), \quad i = 1, 2, \ldots, s. \tag{1. 5}$$

If the solutions g_i of the above system are known, the step update (1. 2 b) become explicit for y_{k+1}. This needs s additional function evaluations unless the coefficient matrix $A = [a_{ij}]$ is non-singular [8]. We can write (1. 5) in the form

$$\begin{bmatrix} g_1 \\ \vdots \\ g_s \end{bmatrix} = A \begin{bmatrix} hf(t_k + c_1 h, y_k + g_1) \\ \vdots \\ hf(t_k + c_s h, y_k + g_s) \end{bmatrix}, \tag{1. 6}$$

so (1. 2 b) become equivalent to

$$y_{k+1} = y_k + \sum_{j=1}^{s} q_i g_i \tag{1. 7}$$

where $(q_1, \ldots, q_s) = (b_1, \ldots, b_s)A^{-1}$. The formula (1. 7) is advantageous in a sense that the values $g_1, \ldots, g_s$ are calculated iteratively. A fixed point iterative scheme can be used but it destroys the stability properties, especially in case of stiff problems. On the other hand, Newton's method deals with such problems effectively. When we apply Newton's method to system (1. 5), we need to solve a linear system with the coefficient matrix

$$\begin{bmatrix} I - ha_{11}\frac{\partial f}{\partial y}(t_k + c_1 h, y_k + g_1) & \cdots & -ha_{1s}\frac{\partial f}{\partial y}(t_k + c_s h, y_k + g_s) \\ \vdots & & \vdots \\ -ha_{s1}\frac{\partial f}{\partial y}(t_k + c_1 h, y_k + g_1) & \cdots & I - ha_{ss}\frac{\partial f}{\partial y}(t_k + c_s h, y_k + g_s) \end{bmatrix},$$

in every iteration. To simplify the above matrix, we can replace the Jacobians $\frac{\partial f}{\partial y}(t_k + c_j h, y_k + g_j)$ by the following approximation

$$J \approx \frac{\partial f}{\partial y}(t_k, y_k).$$

We can choose to perform the evaluation of the Jacobians at every step or at every iteartion or after specific number of steps. The simplified Newton iterative scheme for (1. 5) can be written as

$$\begin{aligned} (I - hA \otimes J)\Delta G^n &= -G^n + h(A \otimes I)F(G^n), \\ G^{n+1} &= G^n + \Delta G^n, \end{aligned} \tag{1. 8}$$

where n is the iteration index, $\otimes$ is the Kronecker product, $G^n = (g_1^n, \ldots, g_s^n)^T$ and $\Delta G^n = (\Delta g_1^n, \ldots, \Delta g_s^n)^T$ are the n-th approximations to the numerical solution and the increments, respectively. Also

$$F(G^k) = (f(t_0 + c_1 h, y_0 + g_1^k), \ldots, f(t_0 + c_j h, y_0 + g_j^k))^T.$$

Every iteration needs s evaluations and the solution of sN linear equations with the same matrix $(I - hA \otimes J)$ for all iterations.

Although, IRK methods have many attractive properties, however they are costly to implement. Several techniques are employed to overcome the high cost which include the introduction of methods, such as, singly diagonally implicit Runge-Kutta (SDIRK) methods [1, 13, 14], singly implicit Runge-Kutta (SIRK) methods [2, 3, 4], and mono-implicit Runge-Kutta (MIRK) methods [5].

The main factor that increases the cost of IRK methods is the implicit nature of its stages. However these are also important to retain so as to solve the stiff systems efficiently. A new class of methods termed as nested Implicit Runge-Kutta (NIRK) methods were introduced by G. Yu. Kulikov and S. K. Shindin [12, 10] to reduce the cost of implementation. The idea is that the stages are evaluated explicitly, however they depend on the unknown output value. This leads to a nonlinear implicit equation involving the output approximation only and this can be solved by Newton iterations.

In the next section, we present Gauss-type nested implicit Runge-Kutta (NIRK) methods and important error estimation techniques used in the implementation of these methods are presented. An important part of this paper is the numerical testing that we discuss in Section 3. Different classes of test problems and comparison of numerical methods applied on these problems are included. The numerical results are also presented and discussed in this section. Finally, Section 4 provides conclusions drawn on the basis of numerical experiments.

2. Gauss-type nested implicit Runge-Kutta methods

NIRK formulas are based on Gauss quadrature formulas and have been developed up to order six. The NIRK methods are A-stable and stiffly accurate [10]. In these methods the internal stages are calculated in an explicit manner so their practical implementation is potentially cheaper than IRK methods. Also, some of the stage values calculated in the integration step are sufficiently accurate, giving a dense output of the same order as of the method applied. With these two properties, the NIRK methods are suitable for solving stiff IVPs. The integration of Hamiltonian problems, partial differential equations and differential algebraic systems is likewise successful [12].

An s-stage nested implicit Runge–Kutta method to solve IVP (1. 1) is presented as

$$Y_j^2 = a_{j1}^2 y_k + a_{j2}^2 y_{k+1} + h\Big(d_{j1}^2 f(t_k, y_k) + d_{j2}^2 f(t_{k+1}, y_{k+1})\Big), \quad j = 1, 2, \tag{2. 9 a}$$

$$\begin{aligned} Y_j^i = {} & a_{j1}^i y_k + a_{j2}^i y_{k+1} + h\Big(d_{j1}^i f(t_k, y_k) + d_{j2}^i f(t_{k+1}, y_{k+1})\Big) \\ & + h \sum_{m=1}^{i-1} d_{j,m+2}^i f(T_m^{i-1}, Y_m^{i-1}), \quad i = 3, 4, \ldots, s, \quad j = 1, 2, \ldots, i, \end{aligned} \tag{2. 9 b}$$

$$y_{k+1} = y_k + h \sum_{i=1}^{s} b_i f(T_i^s, Y_i^s), \tag{2. 9 c}$$

where $T_j^i = t_k + hc_j^i$ and Y_j^i are the i-th level stage values. For an N-dimensional differential system the step update formula (2. 9 c) can be expanded as a system of N nonlinear equations as

$$y_{k+1} - y_k - h\Big(b_1 f(T_1^s, Y_1^s) + b_2 f(T_2^s, Y_2^s) + \ldots + b_s f(T_s^s, Y_s^s)\Big) = 0. \tag{2. 10}$$

The formulas (2. 9 a) and (2. 9 b) represent two levels at which the internal stage values are evaluated. The NIRK method use Hermite interpolation for approximating the true solution at fixed nodes. If two numerical solutions y_k and y_{k+1} and their derivatives y'_k and y'_{k+1} at the grid points t_k and t_{k+1} are available then a Hermite polynomial of degree three can help in constructing the method of order four [11]. To obtain order six NIRK method, the solutions $y(t_k + c_1^2 h)$ and $y(t_k + c_2^2 h)$ at two fixed points $t_k + c_1^2 h$ and $t_k + c_2^2 h$ are approximated by using Hermite polynomial of degree three. These solutions become second level stage values (2. 9 a). Similarly, we use degree five Hermite polynomial to approximate the solutions $y(t_k + c_1^3 h)$, $y(t_k + c_2^3 h)$ and $y(t_k + c_3^3 h)$ at three fixed points $t_k + c_1^3 h$, $t_k + c_2^3 h$ and $t_k + c_3^3 h$ with the help of calculated information. These solutions become 3-rd level stage values ((2. 9 b), for $i = 3$). These second and third level stage values are used in the step update formula (2. 9 c). Note that the numerical solutions y_k computed at t_k and y_{k+1} computed at t_{k+1} can be considered as first level stage values.

An additional property of NIRK methods is that their coefficients a_{ij}, b_j and c_j satisfy

$$\begin{aligned} &a_{j1}^i + a_{j2}^i = 1, \quad i = 2, 3, \ldots, s \quad j = 1, 2, \ldots, i, \\ &c_j^i = a_{j2}^i + \sum_{l=1}^{i+1} d_{jl}^i. \end{aligned} \tag{2. 11}$$

NIRK methods can be represented in Butcher tableau as

0	0	0	0	$\cdots$	0	0	0
c^2	d_1^2	0	0	$\cdots$	0	$a^2 b^T$	d_2^2
c^3	d_1^3	D^3	0	$\cdots$	0	$a^3 b^T$	d_2^3
c^4	d_1^4	0	D^4	$\cdots$	0	$a^4 b^T$	d_2^4
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
c^s	d_1^s	0	0	$\cdots$	D^s	$a^s b^T$	d_2^s
1	0	0	0	$\cdots$	0	b^T	0
	0	0	0	$\cdots$	0	b^T	0

TABLE 1. Butcher tableau for Gauss-type NIRK methods, which shows that these methods can be expressed in the form of IRK methods with $s+1$ block stages.

where,

$$D^i = \begin{bmatrix} d_{13}^i & \cdots & d_{1,i+1}^i \\ \vdots & \ddots & \vdots \\ d_{i3}^i & \cdots & d_{i,i+1}^i \end{bmatrix}, \quad d_1^i = \begin{bmatrix} d_{11}^i \\ \vdots \\ d_{i1}^i \end{bmatrix}, \quad d_2^i = \begin{bmatrix} d_{12}^i \\ \vdots \\ d_{i2}^i \end{bmatrix},$$

$$a^i = \begin{bmatrix} a^i_{12} \\ \vdots \\ d^i_{i2} \end{bmatrix}, \quad c^i = \begin{bmatrix} c^i_1 \\ \vdots \\ c^i_i \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_s \end{bmatrix}.$$

When dealing with stiff problems, a Newton iterative scheme is more efficient than the fixed point iterative scheme to solve the nonlinear equation of the step update formula. Kulikov presented a modified Newton scheme in [9] as given below

$$Y^{2,(l)}_j = a^2_{j1}\bar{y}_k + a^2_{j2}y^{(l-1)}_{k+1} + h\Big(d^2_{j1}f(t_k,\bar{y}_k) + d^2_{j2}f(t_{k+1},y^{(l-1)}_{k+1})\Big), \quad j = 1,2, \tag{2. 12 a}$$

$$Y^{3,(l)}_j = a^3_{j1}\bar{y}_k + a^3_{j2}y^{(l-1)}_{k+1} + h\Big(d^3_{j1}f(t_k,\bar{y}_k) + d^3_{j2}f(t_{k+1},y^{(l-1)}_{k+1})\Big) + h\sum_{m=1}^{2} d^3_{j,m+2}f(T^2_m,Y^{2,(l-1)}_m), \quad j = 1,2,3, \tag{2. 12 b}$$

$$U(hJ)(y^{(l)}_{k+1} - y^{(l-1)}_{k+1}) = -y^{(l-1)}_{k+1} + \bar{y}_k + h\sum_{i=1}^{s} b_i f(T^s_i, Y^{s,(l)}_i), \quad l = 1,2,\ldots,N, \tag{2. 12 c}$$

where $J \overset{\text{def}}{=} \partial_t f(t_{k+1}, y^{(0)}_{k+1})$ represents the Jacobian of the function provided in IVP (1. 1) and $\bar{y}_k = y^M_k, k = 0,1,\ldots,N-1$ is the numerical solution obtained by method (2. 9), using M iterations of (2. 12) per step.

A major constraint for iteration (2. 12) is its practical efficiency because the coefficient matrix $U(hJ)$ in (2. 12 c) is a matrix-valued cubic polynomial of the form

$$U(hJ) \overset{\text{def}}{=} I_m - \frac{1}{2}hJ + \frac{1}{10}(hJ)^2 - \frac{1}{20}(hJ)^3, \tag{2. 13}$$

where I_m is an $m \times m$ identity matrix. The evaluation of the coefficient matrix U as in (2. 13) is impractical due to its large computational cost. One way to overcome this is to use its linear part and ignore the higher degree terms. However this approach does not reduce the CPU time considerably and damages the stability of MIRK methods including NIRK formulas [6, 10, 11]. Cash and Singhal, in [6], suggested to approximate the coefficient matrix U by some power equal to the degree of the original matrix-valued polynomial (2. 13). Kulikov approximated it for NIRK methods [9] as

$$\widetilde{U}(hJ) \overset{\text{def}}{=} \Big(I_m - \frac{1}{6}hJ\Big)^3. \tag{2. 14}$$

The above polynomial has many advantages. Firstly, it is effectively utilized in iterative scheme (2. 12). The matrix $I_m - hJ/6$ is decomposed once per integration step. The three linear systems are then solved with the decomposed matrix. So, this uses reasonably less computational cost. Secondly, the polynomial (2. 14) is an approximation of order 2, that is $U(hJ) = \widetilde{U}(hJ) + O(h^2J^2)$. So, replacement of $U(hJ)$ with $\widetilde{U}(hJ)$ has no negative influence on the convergence rate of the iteration.

In case of Gauss–type NIRK method, a pair of embedded RK formulas can be formed by order four and order six methods [12]. Both of these methods are applied to obtain two numerical solutions of the given IVP. The difference of both numerical solutions estimates the error. Thus a family of order six Gauss–type methods with built-in error estimation is achieved [9]. Kulikov and Shindin introduced an embedded stage error estimation technique in [12, 10] and tested it successfully on order four NIRK methods. They termed it as the *Embedded Stages Approach (ESA)*. It is based on the result attained in [11], that the stage order of the NIRK methods is raised by one when a specific value of the free parameter θ is used [9, 11]. The primary role of θ is to change the coefficients in the coefficient matrix of the Butcher tableau. Moreover, the values of θ are for controlling the stage values of the method. For example, $\theta = 1/2 + 2\sqrt{3}/9$ in Table 2 and $\theta = 9\sqrt{15}/500 - \sqrt{27}/200$ in Table 3 gives the stage values for the Gauss–type NIRK methods of order 4 and order 6, respectively. The following Butcher tableau present Gauss–type NIRK methods of order four and order six with built in error estimations, as constructed in above cited papers.

$$
\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
c_1^2 & \frac{1}{2}c_1^2 + \frac{1}{2}\theta - \frac{5}{12} & \frac{1}{2} - \frac{1}{2}\theta & \frac{1}{2} - \frac{1}{2}\theta & \frac{1}{2}c_1^2 + \frac{1}{12}\theta - \frac{7}{12} \\
1 - c_1^2 & \frac{7}{12} - \frac{1}{2}c_1^2 - \frac{1}{12}\theta & \frac{1}{2}\theta & \frac{1}{2}\theta & \frac{5}{12} - \frac{1}{2}c_1^2 - \frac{1}{12}\theta \\
1 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\
\hline
 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\
\hline
 & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2}
\end{array}
$$

TABLE 2. Butcher tableau for embedded Gauss–type NIRK methods of order 4 for built in error estimation.

$$
\begin{array}{c|ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
c_2 & \frac{c_3}{6} & 0 & 0 & \frac{35}{108} - \frac{10c_3}{27} & \frac{14}{27} - \frac{16c_3}{27} & \frac{35}{108} - \frac{10c_3}{27} & \frac{c_3}{6} - \frac{1}{6} \\
c_3 & \frac{c_2}{6} & 0 & 0 & \frac{35}{108} - \frac{10c_2}{27} & \frac{14}{27} - \frac{16}{27}c_2 & \frac{35}{108} - \frac{10c_2}{27} & \frac{c_2}{6} - \frac{1}{6} \\
c_4 & \frac{c_6}{10} - \frac{3}{200} & \frac{9c_3}{50} - \frac{9}{100} + \theta & \theta & \frac{v_1}{360} - \frac{5\theta}{9} & \frac{v_1}{225} - \frac{8\theta}{9} & \frac{v_1}{360} - \frac{5\theta}{9} & \frac{3}{200} - \frac{c_4}{10} \\
c_5 & \frac{1}{32} & \frac{\sqrt{27}}{32} & -\frac{\sqrt{27}}{32} & \frac{5}{36} & \frac{2}{9} & \frac{5}{36} & -\frac{1}{32} \\
c_6 & \frac{c_4}{10} - \frac{3}{200} & -\theta & \frac{9c_2}{50} - \frac{9}{100} - \theta & \frac{v_2}{360} + \frac{5\theta}{9} & \frac{v_2}{225} + \frac{8\theta}{9} & \frac{v_2}{360} + \frac{5\theta}{9} & \frac{3}{200} - \frac{c_6}{10} \\
1 & 0 & 0 & 0 & \frac{5}{18} & \frac{4}{9} & \frac{5}{18} & 0 \\
\hline
 & 0 & 0 & 0 & \frac{5}{18} & \frac{4}{9} & \frac{5}{18} & 0 \\
\hline
 & 0 & \frac{1}{2} & \frac{1}{2} & -\frac{5}{18} & -\frac{4}{9} & -\frac{5}{18} & 0
\end{array}
$$

TABLE 3. Butcher tableau for embedded Gauss–type NIRK methods of order 6 for built in error estimation.

Four different error estimation techniques are examined numerically for order four and order six Gauss–type NIRK methods in [10] and [9], respectively. We consider these techniques here for the order six methods:

- Embedded method error estimation (EMEE)- This scheme estimates the local error by comparing the two numerical solutions obtained from the embedded NIRK method of order four and order six NIRK methods, respectively.

- Modified embedded method error estimation (MEMEE)- It might be time consuming to implement directly the EMEE for some large-scale differential equations. This issue can be resolved by applying Shampine's idea [8] to modify the EMEE technique. The Shampine's idea is to modify the error estimation technique for an embedded method, especially for stiff equations in which the embedded error becomes unbounded. The resulting MEMEE technique is suitable for stiff ODEs.
- Embedded stage error estimation (ESEE)- Kulikov and Shindin have recently presented a new error estimation technique in [12, 10]. This ESEE technique is successfully tested for order 4 NIRK methods in [12, 10] and is also extended to order six NIRK methods.
- Modified embedded stage error estimation (MESEE)- Similar to EMEE, ESEE cannot be effective for stiff problems. So again using Shampine's idea, the error estimation technique ESEE is modified. The MESEE technique is bounded for any step size and is cheaper as well [9].

3. Numerical testing

In this section we apply the order six NIRK methods on a variety of stiff differential equations. The exact solutions of these problems are not available so we compared our solutions with reference solutions. These reference solutions were computed using a highly accurate Matlab integrator. We first give a brief introduction of the test problems and then discuss our numerical testing. A complete list of test problems is given in the Appendix.

We have applied our numerical methods on the stiff DETEST problems [7]. These problems are divided into different classes depending upon their special features. The problem classes used in this paper are as follows:

- Problem class A (linear with real eigenvalues)-This class consists of four systems. These systems have different dimensions. They are ordered according to increasing stiffness ratio.
- Problem class B (linear with complex eigenvalues)- This class consists of five systems. The first system requires many methods to take variable step sizes. The four remaining systems differ in their eigenvalues.
- Problem class C (nonlinear coupling)- There are five systems in this class. All of these are nonlinear having real eigenvalues.

We apply two numerical methods to stiff DETEST problems. The first method is the Gauss-type NIRK method of order six with built-in error estimation. The second method is the highly accurate Matlab integrator ode15s. It is an RK method that is very effective for stiff ODEs. The details of this integrator can be found in [15].

We have always used modified Newton's method to iteratively solve for the stages. We allow a maximum of three to four iterations for convergence. The iterative scheme uses a coefficient matrix U and we take two different values of the matrix U as given in (2. 13) and (2. 14). The convergence of our iterative scheme is bounded by a small positive number that is 10^{-2} times local error tolerance. We have checked it with other values as well but we did not find any considerable difference. The local error tolerance (TOL) is taken as

$$10^{-i}, i = 2, 3, \dots, 10.$$

This variation in TOL is used to perform different experiments. We evaluate an exact Jacobian and check our results by providing it in every step or in every iteration. Although we have experimented with all the problems in five classes, we have only included a few results here.

```
for $prob_{ID} = 1 : 14$                          % Loop over the 14 problems.
    for $i = 2 : 10$                              % Loop over the tolerance.
        $TOL = 10^{-i}$                           % Set the tolerance.
        Initialise the integration
        NIRK                                      % Attempt the integration using NIRK.
        if ind == 1                               % Test if the integration was successful.
            Calculate $NFE$ or $t_{CPU}$          % It was. Calculate the function ...
                                                  % ... evaluations or CPU time.
            Calculate $ERROR$                     % Estimate the global error.
            Print $ERROR$ along with              % Print results.
                $TOL$ or $NFE$ or $t_{CPU}$
        else
            Print error message                   % Integration failed. Print error message.
        end if
    end for
end for
```

TABLE 4. Pseudo-code of one step of Gauss-type NIRK methods implemented on the collection of 14 test problems.

Table 4 gives pseudo-code for our driver. There are two nested loops. The outer loop is over the 14 test problems $(prob_{ID})$ of stiff DETEST and the inner loop is over the 9 local error tolerances (TOL), giving 126 integrations in total. Once the tolerance has been set, we initialise the integration by setting the initial value of t and y, and the initial step size. We then attempt the integration using the NIRK method. If the integration is successful (ind is 1 on return from the integrator), we calculate the number of function evaluations (NFE) and CPU time (t_{CPU}) used for the integration. We then use the reference solution to estimate the L2 norm of the end-point global error $(ERROR)$. We print this norm against TOL, NFE or t_{CPU}. If the integration was not successful, an error message is printed.

EXPERIMENT 1

In our first set of experiments, we check the accuracy of order six NIRK methods in terms of end-point global error. Figures 1 – 3 contain the graphs of the end-point global errors (ERROR) computed at various tolerances (TOL) for the three classes of stiff DETEST problems.

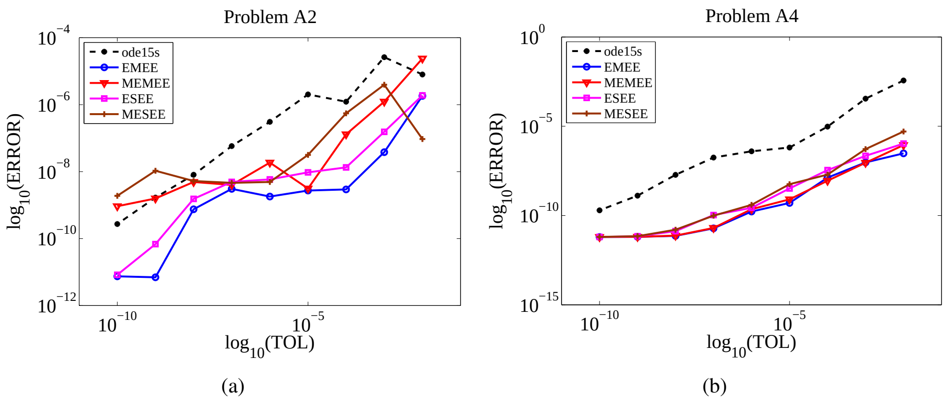


FIGURE 1. Experiment 1a, tolerance vs end-point global error for NIRK and ode15s, (a) = with $\widetilde{U}$ and mitn = 4, (b) = with U and mitn = 4.

For problem class A (linear with real eigenvalues), the NIRK method is more accurate than ode15s method at all tolerances, except for problem A2, where ode15s exhibits a similar or more accuracy at smaller tolerances as shown in Figure 1(a). Moreover, the EMEE technique is the most accurate and MESEE is the least for these set of problems. However, MEMEE and ESEE are more or less accurate relative to one another in different tests. For example, in tests with the coefficient matrix U, MEMEE performs slightly better than ESEE but it is vice versa for tests with the value of $\widetilde{U}$ as in above Figure 1(b). On the other hand, the increasing of maximum allowed iterations from three to four shows slight changes in the results.

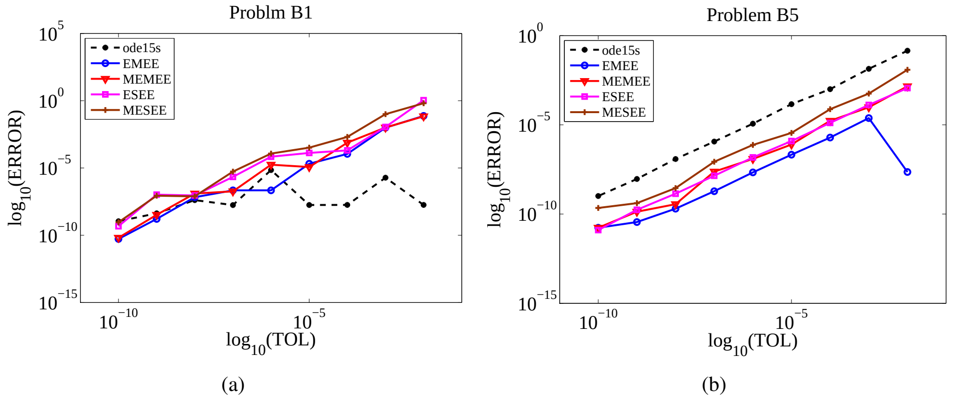


FIGURE 2. Experiment 1b, tolerance vs end-point global error for NIRK and ode15s, (a) = with U and mitn = 3, (b) = with $\widetilde{U}$ and mitn = 3.

In different tests on problems of class B (linear with non-real eigenvalues), the NIRK method exhibits more accurate result than ode15s scheme for problems B2 and B3, but in case of problems B1 and B4, the former method is less accurate at larger tolerances as in Figure 2(a). Problem B5 gives good accuracy with NIRK method when $\widetilde{U}$ is used, however by using U in the iterative scheme, ode15s performs better than ESEE and MESEE techniques for this problem as shown in Figure 2(b). In comparison of different error estimate techniques, EMEE is the most accurate. The accuracy level of ESEE and MESEE is similar to one another in the most experiments and is lower than EMEE and MEMEE.

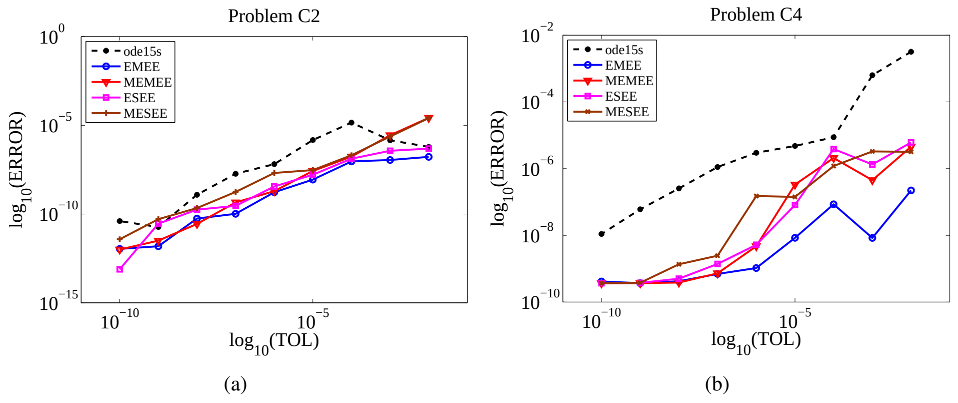


FIGURE 3. Experiment 1c, tolerance vs end-point global error for NIRK and ode15s, (a) = with U and mitn = 3, (b) = with $\widetilde{U}$ and mitn = 3.

For problem class C (nonlinear coupling), ode15s is less accurate than NIRK method for problem C4 and C5. However, for problems C2 and C3, the NIRK method is a little more accurate than ode15s as shown in the Figure 3(a). In case of problem C1, both methods have a similar behaviour in terms of accuracy. Moreover, for this class also, EMEE is the most accurate, however MEMEE exhibits slightly better accuracy than ESEE at higher tolerances when coefficient matrix $\widetilde{U}$ is used in the iterative scheme. Figure 3(b) is an example of this result that also shows that MESEE is the least accurate technique. Most of the other results with U are quite similar to those with $\widetilde{U}$.

EXPERIMENT 2

One way to measure the efficiency of a NIRK scheme is to count the number of function evaluations (NFE) against the end-point global errors (ERROR), computed at different tolerances

$$10^{-i}, i = 2, 3, \ldots, 10.$$

The results for the second set of experiments are displayed in Figures 4 – 6. The initial values are same as in the case of first set of experiments.

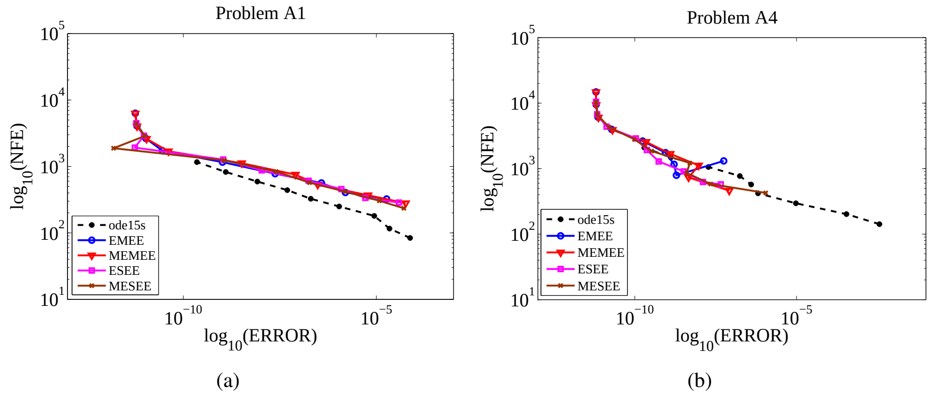


FIGURE 4. Experiment 2a, end-point global error vs function evaluation for NIRK and ode15s, (a) = with U and mitn = 3, (b) = with $\widetilde{U}$ and mitn = 4.

For problem class A, the NIRK method uses a little more function evaluations than ode15s. Maximum increase is less than two times in case of problem A1 with coefficient matrix U as shown in Figure 4(a). However with $\widetilde{U}$, this difference is decreased for the EMEE scheme. Problem A2 exhibits a similar result. For problem A3, both methods show nearly same efficiency using U, while the MEMEE and MESEE techniques are not as efficient as the other two schemes when $\widetilde{U}$ is used. Figure 4(b) shows that for problem A4, the NIRK scheme is more efficient than ode15s. All four error estimation techniques of NIRK method uses nearly same number of function evaluations for all problems of this class.

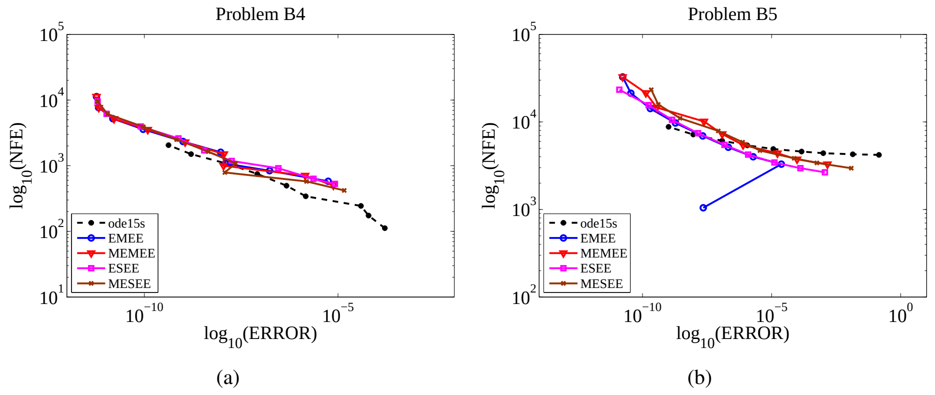
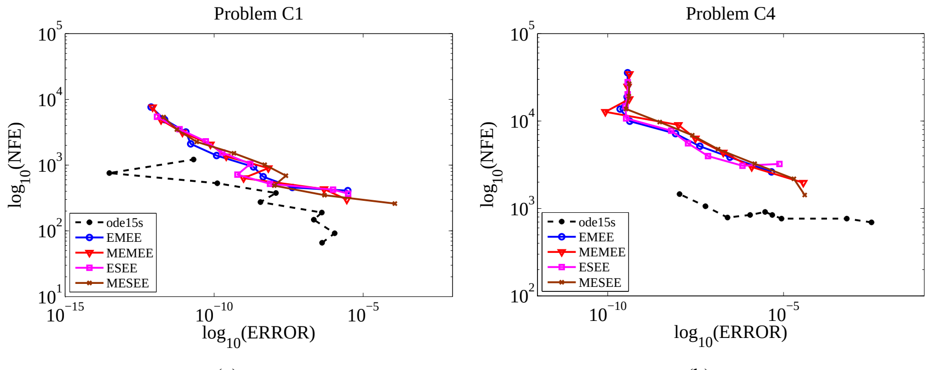


FIGURE 5. Experiment 2b, end-point global error vs function evaluation for NIRK and ode15s, (a) = with U and mitn = 4, (b) = with $\widetilde{U}$ and mitn = 3.

In problem class B, problems B2, B3 and B4 show similar efficiency as A1. Problem B4 is shown in Figure 5(a). The integrator ode15s uses a maximum of 2 times less function evaluations. In problem B1, the behaviour of ode15s is different, although it used less function evaluations than the NIRK method. The reason of this behaviour is due to its irregular global errors for this problem as in Figure 2(a). For problem B5, the NIRK shows poor efficiency when $\widetilde{U}$ is used in the iteration scheme. On the other hand, its efficiency is very much improved by using U as given in Figures 5(b). Also, in this problem, EMEE and ESEE techniques are more efficient than the other two and the EMEE scheme uses about five times less function evaluations at an error of near 10^{-8}.



(a) (b)

FIGURE 6. Experiment 2c, end-point global error vs function evaluation for NIRK and ode15s, (a) = with $\widetilde{U}$ and mitn = 3, (b) = with U and mitn = 3.

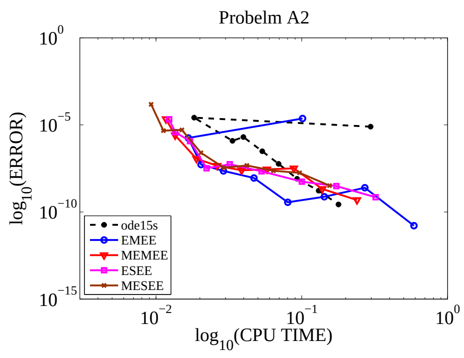
For problem class C, the results of all problems are similar to each other except problem C1. For all problems, ode15s scheme is found to be more efficient than NIRK method. The difference in the function evaluations is small for problem C1, but is increased up to factor of five in the case of problem C4 as shown in Figures 6(a) and 6(b). The choice of coefficient matrix U or $\widetilde{U}$ does not have an effect on the results. Moreover all error estimation schemes show similar efficiency for all problems.

EXPERIMENT 3

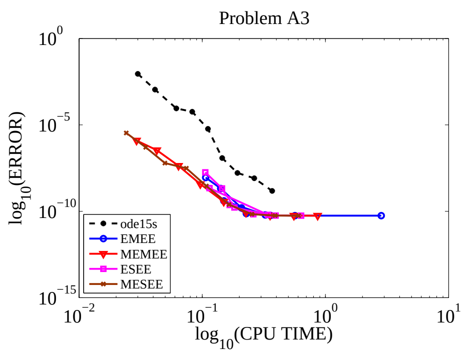
Finally, we conduct a set of experiments to compute the computational times (CPU TIME) and end-point global errors (ERROR) at nine different tolerances

$$10^{-i}, i = 2, 3, \ldots, 10.$$

The results of this set of experiments give a good estimate of the efficiency of NIRK methods. Some of the results are shown in Figures 7 – 9.



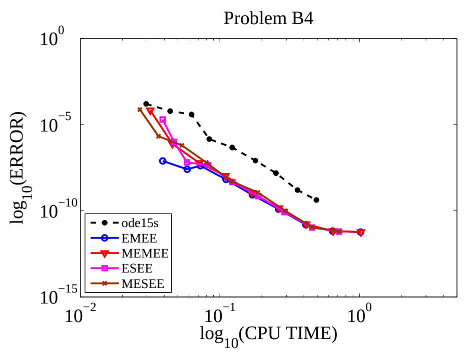
(a)



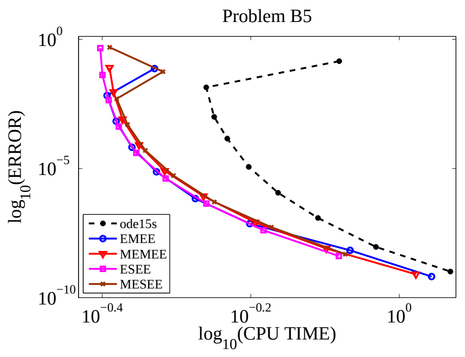
(b)

FIGURE 7. Experiment 3a, CPU time vs end-point global error for NIRK and ode15s, (a) = with U and mitn = 3, (b) = with $\widetilde{U}$ and mitn = 4.

For problem class A, the NIRK method uses less computational time compared to ode15s for all problems except A2 in which ode15s performs better at smaller tolerances. Based on the Jacobian, the only difference found in most of the tests is that in case of Jacobian evaluation at each step, the CPU time at tolerance of 10^{-2} is greater as compared to remaining smaller tolerances. The results of two problems A2 and A3 are shown in Figures 7(a) and 7(b), respectively. The problems A1 and A4 give similar behaviour to problem A3. The choice of the coefficient matrix U or $\widetilde{U}$ and maximum number of allowed iterations 3 or 4 did not make any major effect on the computational times. Moreover all four error estimation techniques of NIRK scheme exhibit nearly same behaviour for all of these problems.



(a)



(b)

FIGURE 8. Experiment 3b, CPU time vs end-point global error for NIRK and ode15s, (a) = with $\widetilde{U}$ and mitn = 4, (b) = with U and mitn = 3.

In problems of class B, NIRK method takes less CPU time than ode15s scheme for problems B2, B3 and B5. In case of problem B4, ode15s performs better than NIRK when the maximum allowed iterations is 3 and U is taken as the coefficient matrix. For $\widetilde{U}$ and 4 allowed iterations, again NIRK approximates the solutions in less computational time. However, for problem B1, at smaller tolerances, ode15s scheme exhibits better efficiency than the NIRK method. This result is due to irregular behaviour of global errors for this problem as we have seen in Figure 2(a). The choice of Jacobian evaluation within each step or iteration has similar effect as in case of problem class A. The results of problems B4 and B5 are displayed in Figures 8(a) and 8(b), respectively. Also we can see form these figures that different error estimation techniques are almost using relatively the same CPU time.

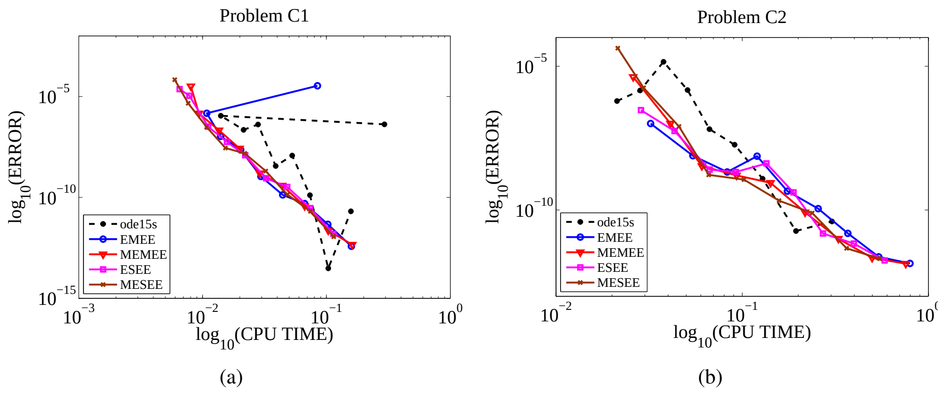


FIGURE 9. Experiment 3c, CPU time vs end-point global error for NIRK and ode15s, (a) = with U and mitn = 3, (b) = with $\widetilde{U}$ and mitn = 4.

In case of problem class C, for problems C4 and C5, the NIRK method is more efficient than ode15s scheme whether U or $\widetilde{U}$ is used as the coefficient matrix. Also the choice of Jacobian evaluation and maximum number of allowed iterations has no effects on the results. However for problems C1, C2 and C3, at smaller tolerance of near 10^{-9}, the ode15s scheme takes less computational time as compared to the NIRK method as shown in Figure 9(b). In case of problem C1, NIRK and ode15s schemes use same CPU time at some tolerances as in Figure 9(a). Also we can see in this Figure that at tolerance of 10^{-2} both schemes use greater time when the Jacobian is evaluated at each step. Among four error estimation techniques MEMEE performs slightly better at smaller tolerances than others.

4. CONCLUSIONS

Gauss-type NIRK methods of order 2, 4 and 6 have recently been introduced by G. Yu. Kulikov and S. K. Shindin in [10, 12, 9]. These methods have cheap practical implementation as compared to IRK methods due to their explicit internal stages. The NIRK methods have built-in local error estimates and we have used four different techniques in this paper. A variety of linear and nonlinear, stiff problems from stiff

DETEST with both real and complex eigenvalues were numerically solved with NIRK and ode15s methods. We have performed three experiments to compute the end-point global errors, function evaluations and computational times at different tolerances. The accuracy and efficiency of NIRK methods were analysed problem-wise and the results of some problems were presented graphically. The overall conclusions for the three problem classes are summarized below.

- For solving linear problems with real eigenvalues, the NIRK method exhibited good results compared to ode15s. This method was more accurate than ode15s. Moreover, the NIRK method was also more efficient than ode15s method. However, in some tests, NIRK method used a little more function evaluations.
- In case of linear problems with complex eigenvalues, NIRK method performed better than ode15s. The NIRK method used slightly more function evaluations than ode15s.
- For nonlinear coupled problems, the accuracy of NIRK method was good compared to ode15s. However, this method used more function evaluations than ode15s. The computational time of NIRK method was relatively less except at a few smaller tolerances. So the NIRK method retained its efficiency for these type of problems as well.

In our comparison of the NIRK method with ode15s, we conclude that the accuracy of NIRK method was promising. The NIRK method was more efficient for most of the problems in terms of CPU time. However, the NIRK method showed poor results in terms of function evaluations for most of the problems. When we analysed the four error estimation techniques, we found the EMEE technique performed better as compared to others.

5. Appendix Test problems (Stiff DETEST)

The problems tested in this paper were the well known stiff DETEST problems. These were all stiff IVPs provided in [7]. In this paper, the problems were classified into three classes that represent linear and nonlinear categories with corresponding real and non-real eigenvalues. The differential equations of all problems, their intervals of integration and their initial conditions and step sizes are listed below. Moreover the eigenvalues of the Jacobian are also given for each problem except where these values are obvious. A constant eigenvalue throughout the interval of integration is given by a single number whereas if an eigenvalue is monotonically increasing or decreasing, then its value at the initial point and in the vicinity of the endpoint are given, with an arrow indicating the change from one to the other [7].

Problem Class A: *Linear With Real Eigenvalues*

A1:
$$\begin{aligned} y_1' &= -0.5y_1, & y_1(0) &= 1,\\ y_2' &= -y_2, & y_2(0) &= 1,\\ y_3' &= -100y_3, & y_3(0) &= 1,\\ y_4' &= -90y_4, & y_4(0) &= 1, \end{aligned}$$

$$t_{end} = 20, \quad h_{initial} = 10^{-2}.$$

A2:
$$\begin{aligned}
y_1' &= -1800y_1 + 900y_2, && y_1(0) = 0, \\
y_j' &= y_{j-1} - 2y_j + y_{j+1}, && y_j(0) = 0, \qquad j = 2, 3, \ldots, 8, \\
y_9' &= 1000y_8 - 2000y_9 + 1000, && y_9(0) = 0,
\end{aligned}$$

$$t_{end} = 120, \quad h_{initial} = 5 \times 10^{-4},$$

(Eigenvalues: $-0.10, -0.10, -1.0, -1.0, -2.6, -3.3, -3.8, -2000, -2000$)

A3:
$$\begin{aligned}
y_1' &= -10^4y_1 - 100y_2 - 10y_3 + y_4, && y_1(0) = 1, \\
y_2' &= -10^3y_2 + 10y_3 - 10y_4, && y_2(0) = 1, \\
y_3' &= -y_3 + 10y_4, && y_3(0) = 1, \\
y_4' &= -0.1y_4, && y_4(0) = 1,
\end{aligned}$$

$$t_{end} = 20, \quad h_{initial} = 10^{-5}.$$

A4: $y_j' = -j^5y_j, \qquad y_j(0) = 1, \quad j = 1, 2, \ldots, 10,$

$$t_{end} = 1, \quad h_{initial} = 10^{-5}.$$

PROBLEM CLASS B: *Linear With Complex Eigenvalues*

B1:
$$\begin{aligned}
y_1' &= -y_1 + y_2, && y_1(0) = 1, \\
y_2' &= -10^2y_1 - y_2, && y_2(0) = 0, \\
y_3' &= -10^3y_3 + y_4, && y_3(0) = 1, \\
y_4' &= -10^4y_3 - 10^2y_4, && y_4(0) = 0,
\end{aligned}$$

$$t_{end} = 20, \quad h_{initial} = 7 \times 10^{-3},$$

(Eigenvalues: $-1 \pm 10i, -100 \pm 100i$)

B2:
$$\begin{aligned}
y_1' &= -10y_1 + \mu y_2, && y_1(0) = 1, \\
y_2' &= -\mu y_1 - y_2, && y_2(0) = 1, \\
y_3' &= -4y_3, && y_3(0) = 1, \\
y_4' &= -y_4, && y_4(0) = 1, \\
y_5' &= -0.5y_5, && y_5(0) = 1, \\
y_9' &= -0.1y_6, && y_6(0) = 1,
\end{aligned}$$

$$t_{end} = 20, \quad h_{initial} = 10^{-2}, \quad \mu = 3,$$

(Eigenvalues: $-0.1, -0.5, -1, -4, -10 \pm i\mu$)

B3: Same as **B2** with $\mu = 8$.

B4: Same as **B2** with $\mu = 25$.

B5: Same as **B2** with $\mu = 100$.

PROBLEM CLASS C: *Nonlinear Coupling*

C1:
$$\begin{aligned} y_1' &= -y_1 + y_2^2 + y_3^2 + y_4^2, && y_1(0) = 1,\\ y_2' &= -10y_2 + 10y_3^2 + 10y_4^2, && y_2(0) = 1,\\ y_3' &= -40y_3 + 40y_4^2, && y_3(0) = 1,\\ y_4' &= -100y_4 + 2, && y_4(0) = 1, \end{aligned}$$
$$t_{end} = 20, \quad h_{initial} = 10^{-2},$$
(Coupling from transient components to smooth components)

C2:
$$\begin{aligned} y_1' &= -y_1 + 2, && y_1(0) = 1,\\ y_2' &= -10y_2 + \nu y_1^2, && y_2(0) = 1,\\ y_3' &= -40y_3 + 4\nu y_1^2 + 4\nu y_2^2, && y_3(0) = 1,\\ y_4' &= -100y_4 + 10\nu y_1^2 + 10\nu y_2^2 + 10\nu y_3^2, && y_4(0) = 1, \end{aligned}$$
$$t_{end} = 20, \quad h_{initial} = 10^{-2}, \quad \nu = 0.1,$$
(Coupling from smooth components to transient components)

C3: Same as **C2** with $\nu = 1$.

C4: Same as **C2** with $\nu = 10$.

C5: Same as **C2** with $\nu = 20$.

REFERENCES

[1] R. Alexander, *Diagonally implicit Runge–Kutta methods for stiff ODEs*, SIAM J. Numer. Anal. **14**, No. 6 (1977) 1006-1024.

[2] K. Burrage, *A special family of Runge–Kutta methods for solving stiff differential equations*, BIT. **18**, No. 1 (1978) 22-41.

[3] K. Burrage, J. C. Butcher and F. H. Chipman, *An implementation of singly implicit Runge–Kutta methods*, BIT. **20**, No. 3 (1980) 326-340.

[4] J. C. Butcher, *A generalization of singly-implicit methods*, BIT. **21**, No. 2 (1981) 175-189.

[5] J. R. Cash, *A class of implicit Runge–Kutta methods for the numerical integration of stiff ordinary differential equations*, J. Assoc. Comput. Mach. **22**, No. 4 (1975) 504-511.

[6] J. R. Cash and A. Singhal, *Mono-implicit Runge–Kutta formulae for the numerical integration of stiff differential systems*, IMA J. Numer. Anal. **2**, No. 2 (1982) 211-227.

[7] W. H. Enright, T. E. Hull and B. Lindberg, *Comparing numerical methods for stiff systems of ODEs*, BIT. **15**, No. 1 (1975) 10-48

[8] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential Algebraic Problems*, Springer-Verlag, 1996.

[9] G. Yu. Kulikov, *Automatic error control in the Gauss-type nested implicit Runge–Kutta formula of order 6*, Russ J. Numer. Anal. Math. Model. **24**, No. 2 (2009) 123-144.

[10] G. Yu. Kulikov, A. I. Merkulov and S. K. Shindin, *Asymptotic error estimate for general Newton-type methods and its applications to differential equations*, Russ. J. Numer. Anal. Math. Model. **22**, No. 6 (2007) 567-590.

[11] G. Yu. Kulikov and S. K. Shindin, *On a family of cheap symmetric one-step methods of order four*, In International Conference on Computational Science. Springer, Berlin, Heidelberg (2006) 781-785.

[12] G. Yu. Kulikov and S. K. Shindin, *Adapted nested implicit Runge–Kutta formulas of Gauss type*, Appl. Numer. Math. **59**, No. 3-4 (2009) 707-722.

[13] A. Kværnø, *Singly diagonally implicit Runge–Kutta methods with an explicit first stage*, BIT. **44**, No. 3 (2004) 489-502.

[14] S. P. Nørsett, *Semi-explicit Runge–Kutta methods*, Matematisk Institut, Universitet I, 1974.

[15] L. F. Shampine and M. W. Reichelt, *The matlab ode suite*, SIAM J SCI COMPUT. **18**, No. 1 (1997) 1-22.