C Software for Some New Autonomous Methods

ADRIAN IONESCU
Wagner College
Department of Mathematics and Computer Science
Staten Island, NY 10301
USA

OLIN JOHNSON
University of Houston
Department of Computer Science
Houston, TX 77204
USA

Abstract: This work is an extension of our previous paper [3], in which we have started introducing new C software for autonomous ordinary differential equation initial-value problems

$$\begin{aligned} y' &= f(y), \quad y \in \mathbb{R}^n, \\ y(x_0) &= y_0, \quad x_0 \in \mathbb{R},\ y_0 \in \mathbb{R}^n, \end{aligned}$$

which implements new Runge-Kutta methods [1], [2]. The novel feature of this approach is the replacement of evaluations of f by approximations or evaluations of f_y. The advantage of this new method lies in the fact that fewer evaluations of f are required than in the standard Runge-Kutta methods and, usually, f_y can be approximated to the desired accuracy with very little arithmetic. In effect, the new methods can be thought as multi-step Runge-Kutta methods. In this paper, we introduce some new Goeken-Johnson interpolation methods. We have also extended the capabilities of the software and we compare the classical Runge-Kutta methods of orders 3, 4 and 5, and the corresponding new Goeken-Johnson methods using both f_y and approximations of f_y. We present numerical results of these comparisons. These results (for performance and accuracy) indicate that the new methods can be at least comparable if not better than the classical methods.

Key–Words: Multi-step, Autonomous numerical methods, C software, Interpolation, Numerical comparisons

1 Introduction

The nonstiff first-order ordinary differential equation initial-value problem

$$\begin{aligned} y' &= f(x,y), \quad x \in \mathbb{R},\ y \in \mathbb{R}^n, \\ y(x_0) &= y_0, \quad x_0 \in \mathbb{R},\ y_0 \in \mathbb{R}^n, \end{aligned} \tag{1}$$

is often solved numerically using an explicit Runge-Kutta method of a specific order and with a number of evaluations (stages). For simplicity we consider the autonomous form of Problem (1): y and f have $n+1$ components with $y_{n+1} = x$ and $f_{n+1}(y) = 1$. Then we obtain the following first-order ODE initial value problem:

$$\begin{aligned} y' &= f(y), \quad x \in \mathbb{R},\ y \in \mathbb{R}^{n+1}, \\ y(x_0) &= y_0, \quad x_0 \in \mathbb{R},\ y_0 \in \mathbb{R}^{n+1}, \\ & (y_0)_{n+1} = x_0, \end{aligned} \tag{2}$$

One defines the following recursive algorithm with s stages, cf., Section 2 [1]:

$$\begin{aligned} k_i &= hf(y_j + \sum_{p=1}^{i-1} a_{ip}k_p + ha_{ii}f_y(y_j)k_1), \\ i &= 1, 2, \ldots, s \\ y_{j+1} &= y_j + \sum_{p=1}^{s} b_p k_p. \end{aligned} \tag{3}$$

2 Goeken-Johnson Autonomous Methods

For the first-order ODE initial-value problem (2) we assume that $f(y)$ has derivatives to the fourth order in a domain in $\mathbb{R}^{n+1}$ where $y \in \mathbb{R}^{n+1}$. We also assume that

$$\| f(y_1) - f(y_2) \|_2 \leq L \| y_1 - y_2 \|_2$$

for all y_1 and y_2 in the domain. Therefore, Problem (2) has a unique local solution.

2.1 Goeken-Johnson third-order method

The third-order Goeken-Johnson method (GJ32) has 2 stages, cf., Equation (3), given by

$$\begin{aligned} k_1 &= hf(y_j), \\ k_2 &= hf(y_j + a_{21}k_1 + ha_{22}f_y(y_j)k_1), \end{aligned} \tag{4}$$

b_1	b_2	a_{21}	a_{22}
1/4	3/4	2/3	2/9

Table 1: Example of Goeken-Johnson 3rd order coefficients.

b_1	1/6	1/6	1/6	1/10	1/10
b_2	1/6	2/3	2/3	1/2	1/2
b_3	2/3	1/6	1/6	2/5	2/5
a_{21}	1	1/2	1/2	1/3	1/3
a_{22}	1/2	1/8	-1/8	1/18	-1/6
a_{31}	3/8	-1	3	-25/24	35/24
a_{32}	1/8	2	-2	15/8	-5/8
a_{33}	0	-1/2	5/2	-5/18	5/6

Table 2: Example of Goeken-Johnson 4th order coefficients.

$$y_{j+1} = y_j + b_1k_1 + b_2k_2;$$

example coefficients are given in Table 1.

2.2 Goeken-Johnson fourth-order method

The fourth-order Goeken-Johnson method (GJ43) has 3 stages, cf., Equation (3), given by

$$\begin{aligned} k_1 &= hf(y_j), \\ k_2 &= hf(y_j + a_{21}k_1 + ha_{22}f_y(y_j)k_1), \\ k_3 &= hf(y_j + a_{31}k_1 + a_{32}k_2 \\ &\quad +ha_{33}f_y(y_j)k_1), \\ y_{j+1} &= y_j + b_1k_1 + b_2k_2 + b_3k_3; \end{aligned} \tag{5}$$

example coefficients are given in Table 2.

2.3 Goeken-Johnson fifth-order method

The fifth-order Goeken-Johnson method (GJ54) has 4 stages, cf., Equation (3), given by

$$\begin{aligned} k_1 &= hf(y_j), \\ k_2 &= hf(y_j + a_{21}k_1 + ha_{22}f_y(y_j)k_1), \\ k_3 &= hf(y_j + a_{31}k_1 + a_{32}k_2 \\ &\quad +ha_{33}f_y(y_j)k_1), \\ k_4 &= hf(y_j + a_{41}k_1 + a_{42}k_2 \\ &\quad +a_{43}k_3 + ha_{44}f_y(y_j)k_1), \\ y_{j+1} &= y_j + b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4; \end{aligned} \tag{6}$$

example coefficients are given in Table 3.

2.4 Goeken-Johnson Interpolation Methods

The Goeken-Johnson interpolation methods use basically the same algorithms as the ones presented in Sections 2.1, 2.2, 2.3. The only difference is that the derivatives f_y are replaced by approximations of f_y. The method starts by using a Runge-Kutta method in order to initialize the process and determine y_0, y_1, y_2, y_3. The cubic interpolation for y_0, y_1, y_2, y_3 allows us to determine an approximation of $f_y(y_4)$. For all the intervals from $j = 4$ to $j = m$ the corresponding Goeken-Johnsom method is used to advance from y_j to y_{j+1}. In order to obtain (approximation of) $f_y(y_j)$, cubic interpolation is used for $y_{j-3}, y_{j-2}, y_{j-1}, y_j$. Note that the interpolation algorithm is due to G.B. Rybicki, cf., [4].

b_1	5/48	1/24	5/54	1/14
b_2	27/56	125/336	250/567	32/81
b_3	125/336	27/56	32/81	250/567
b_4	1/24	5/48	1/14	5/54
a_{21}	1/3	1/5	3/10	1/4
a_{22}	1/18	1/50	9/200	1/32
a_{31}	-152/125	-52/27	-9/8	-329/250
a_{32}	252/125	70/27	15/8	252/125
a_{33}	-44/125	-8/27	-9/32	-259/1000
a_{41}	19/2	43/5	17/3	209/35
a_{42}	-72/7	-64/7	-490/81	-32/5
a_{43}	25/14	54/35	112/81	10/7
a_{44}	5/2	13/10	23/18	11/10

Table 3: Example of 5th order Goeken-Johnson coefficients.

3 Design of Goeken-Johnson Suite

For our intermediate value problems (IVP) we solve the autonomous case

$$\begin{aligned} y' &= f(y), \quad y \in \mathbb{R}^{n+1}, \\ y(x_0) &= y_0, \quad x_0 \in \mathbb{R},\ y_0 \in \mathbb{R}^{n+1}, \\ &\quad \text{where } (y_0)_{n+1} = x_0. \end{aligned}$$

The user can easily extend the software to cover other variants of the Goeken-Johnson methods. Our software allows the user to easily change the coefficients used in the computation, [3]. For other C-implementations as well as other numerical methods also see [4].

We provide a driver to call the appropriate method of the user's choice. The user has the flexibility of calling the routines as needed by simply changing the driver. The user provides the initial value ($x_0 = a$) and the final value ($x_m = b$), as well as the number of steps (m) which automatically determine h ($h = \frac{b-a}{m}$). Obviously, the user must also provide the function f and the functions f_y or the approximations

of f_y. Because of the complexity of polynomial interpolation for multi-variables, the software covers only the one-dimensional case when interpolation is used. In the multi-dimensional case, if f consists of elementary functions, one can determine f_y [3]. Moreover, the answers are also converted to C code which is then used in the derivative routines. Some examples are presented in Section 4.

4 Numerical Results

In this Section, some numerical comparisons of the classical Runge-Kutta (RK) methods of orders 3, 4 and 5, and of the corresponding new Goeken-Johnson (GJ) methods using both f' and approximations of f' are presented.

Example 1 *Solve the following system of ODE:*

$$\begin{aligned} \frac{dy_1}{dx} &= y_1\, y_2, \quad y_1(0) = 1, \\ \frac{dy_2}{dx} &= y_1 + y_2. \quad y_2(0) = -1, \end{aligned}$$

If we consider the initial value ($a = 0$), the final value ($b = 1$), and the number of steps ($m = 10$), we obtain the following y values:

- RK Method, Order 3 (RK33):
  $y_1[1.000] = 0.3057318786$,
  $y_2[1.000] = -1.5383294619$.
- RK Method, Order 4 (RK44):
  $y_1[1.000] = 0.3071159057$,
  $y_2[1.000] = -1.5675091726$.
- RK Method: Order 5 (RK56):
  $y_1[1.000] = 0.3071138201$,
  $y_2[1.000] = -1.5675108063$.
- GJ Method, Order 3 (GJ32):
  $y_1[1.000] = 0.3071159719$,
  $y_2[1.000] = -1.5674266680$.
- GJ Method, Order 4 (GJ43):
  $y_1[1.000] = 0.3071134365$,
  $y_2[1.000] = -1.5675079666$.
- GJ Method, Order 5 (GJ54):
  $y_1[1.000] = 0.3071138591$,
  $y_2[1.000] = -1.5675108474$.

In Table 4, we present time comparisons when running the software for both the Goeken-Johnson and the classical Runge-Kutta methods on a Dell PowerEdge SC430 with a Pentium D 2.8 GHz processor. For example, the results obtained by using the

Steps (m)	Goeken-Johnson	Runge-Kutta
	3rd Order	3rd Order
10,000	31 ms	31 ms
100,000	341 ms	341 ms
1,000,000	3,453 ms	3,454 ms
10,000,000	34.656 ms	34,562 ms
	4th Order	4th Order
10,000	46 ms	47 ms
100,000	469 ms	453 ms
1,000,000	4,688 ms	4,547 ms
10,000,000	47,078 ms	45,578 ms
	5th Order	5th Order
10,000	47 ms	62 ms
100,000	515 ms	625 ms
1,000,000	5,172 ms	6,172 ms
10,000,000	51,813 ms	61,687 ms

Table 4: Time comparisons for Goeken-Johnson and classical Runge-Kutta methods (Example 1.)

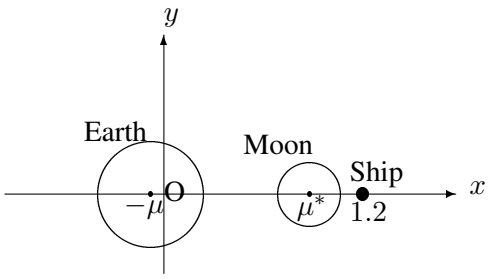

Figure 1: Configuration Earth, Moon, Spaceship ($\mu^* = 1 - \mu$).

Goeken-Johnson fifth-order method are about 20% faster than the corresponding classical Runge-Kutta method. Note that the fifth-order Goeken-Johnson method requires only four evaluations of f compared to six evaluations of the classical fifth-order Runge-Kutta method.

The following four-dimensional autonomous system describes the configuration shown in Figure 1 [3]. The spaceship is returning to the initial position in 6.19216933 days. The distances are relative to the distance between the centers of the Earth and Moon which is considered to be equal to a unit.

Example 2 *Solve the following system of ODE ($\mu = \frac{1}{82.45}$ and $\mu^* = 1 - \mu$):*

$$\begin{aligned} \frac{dy_1}{dt} &= y_2, \quad y_1(0) = 1.2, \\ \frac{dy_2}{dt} &= 2\, y_4 + y_1 - \frac{\mu^*(y_1 + \mu)}{([(y_1 + \mu)^2 + y_3^2]^{\frac{3}{2}}} \end{aligned}$$

$$
\begin{aligned}
& - \frac{\mu(y_1-\mu^*)}{([(y_1-\mu^*)^2+y_3^2]^{\frac{3}{2}}}, \quad y_2(0)=0,\\
\frac{dy_3}{dt} &= y_4, \quad y_3(0)=0,\\
\frac{dy_4}{dt} &= -2\,y_2+y_3-\frac{\mu^*\,y_3}{([(y_1+\mu)^2+y_3^2]^{\frac{3}{2}}}\\
& - \frac{\mu\,y_3}{([(y_1-\mu^*)^2+y_3^2]^{\frac{3}{2}}},\\
& y_4(0)=-1.04935751,\\
& \textit{where } t\in[0,6.19216933]
\end{aligned}
$$

One can complete the calculation of the required derivatives. For more complicated derivatives computer algebra systems have been integrated with the C software [3]. We then use the driver provided by the Goeken-Johnson Suite. For example, if we consider the initial value ($a = 0$), the final value ($b = 6.19216933$), and the number of steps ($m = 10,000$), we obtain the following y values:

- RK Method, Order 3 (RK33):
  $y_1[6.192] = 6.7797688840$,
  $y_2[6.192] = -0.7783518240$,
  $y_3[6.192] = -2.0295329919$,
  $y_4[6.192] = -7.1111445181$.
- RK Method, Order 4 (RK44):
  $y_1[6.192] = 1.2004042400$,
  $y_2[6.192] = 0.0004845485$,
  $y_3[6.192] = 0.0036875589$,
  $y_4[6.192] = -1.0494447749$.
- RK Method: Order 5 (RK56):
  $y_1[6.192] = 1.2004397146$,
  $y_2[6.192] = 0.0005478063$,
  $y_3[6.192] = 0.0036335424$,
  $y_4[6.192] = -1.0494826623$.
- GJ Method, Order 3 (GJ32):
  $y_1[6.192] = 1.1829030026$,
  $y_2[6.192] = -0.0351497113$,
  $y_3[6.192] = 0.0207671715$,
  $y_4[6.192] = -1.0321513538$.
- GJ Method, Order 4 (GJ43):
  $y_1[6.192] = 1.2005849363$,
  $y_2[6.192] = 0.0008386902$,
  $y_3[6.192] = 0.0034438378$,
  $y_4[6.192] = -1.0496364521$.
- GJ Method, Order 5 (GJ54):
  $y_1[6.192] = 1.2004677012$,
  $y_2[6.192] = 0.0006093921$,
  $y_3[6.192] = 0.0036027291$,
  $y_4[6.192] = -1.0495121148$.

Steps (m)	Goeken-Johnson	Runge-Kutta
	3rd Order	3rd Order
10,000	125 ms	78 ms
100,000	1,234 ms	875 ms
1,000,000	12,390 ms	8,719 ms
10,000,000	124,734 ms	88,250 ms
	4th Order	4th Order
10,000	156 ms	125 ms
100,000	1,578 ms	1,218 ms
1,000,000	15,719 ms	12,094 ms
10,000,000	158,906 ms	121,782 ms
	5th Order	5th Order
10,000	188 ms	172 ms
100,000	1,812 ms	1,703 ms
1,000,000	18,157 ms	17,109 ms
10,000,000	183,281 ms	171,625 ms

Table 5: Time comparisons for Goeken-Johnson and classical Runge-Kutta methods (Example 2.)

Note that we obtain the same final values as the initial ones. The fourth- and fifth-order methods give accurate answers. In this example, the third-order Goeken-Johnson autonomous method proved to be more accurate than the corresponding third-order classical Runge-Kutta method. Increasing the number of steps does not improve the accuracy of the answers significantly. For example, using $m = 10,000,000$, we obtain for the GJ Method, Order 5 (GJ54):
$y_1[6.192] = 1.2004445134$,
$y_2[6.192] = 0.0005582723$,
$y_3[6.192] = 0.0036281392$,
$y_4[6.192] = -1.0494876922$.

Note that we have complicated formulas for the derivatives [3]. This is the reason that the times for the fifth order Runge-Kutta method are somewhat better than the corresponding Goeken-Johnson method, cf., Table 5.

5 Conclusions

We have developed software to solve IVP when $y \in \mathbb{R}^{n+1}$, $n \geq 1$, in the autonomous case. It is relatively straightforward to address the cases when the dependent variable $y \in \mathbb{R}^{m\times n}$ or $y \in \mathbb{C}^n$.

The new package includes the base routines for the Runge-Kutta methods of order 3-5 and for the Goeken-Johnson methods of order 3-5 (autonomous case using f_y or interpolations of f_y). We have also included the documentation, and some solved examples, cf., Section 4. Software is available for download

at http://www.wagner.edu/~ionescu/papers/gj/gji.zip.

We compared the classical Runge-Kutta methods and the corresponding Goeken-Johnson methods and presented numerical results of these comparisons. These results indicate that the new methods can be at least comparable if not better than the classical methods. In examples which involve simple derivatives, the fifth-order Goeken-Johnson method is about 20% better than the corresponding Runge-Kutta method. In other examples, which involve complicated derivatives, the Runge-Kutta methods may run faster than Goeken-Johnson methods.

References:

[1] D. Goeken and O. Johnson, Fifth-Order Runge-Kutta with Higher Order Derivative Approximations, *Electronic Journal of Differential Equations* Conference 02, 1999, pp. 1–9.

[2] D. Goeken and O. Johnson, Runge-Kutta with Higher Order Derivative Approximations, *Applied Numerical Mathematics* 34, 2000, pp. 207-218.

[3] A. Ionescu, O. Johnson and R. Abbasian, Maple and C Software for Some Autonomous Numerical Methods, *Proceedings of MC 2005*, Waterloo, Canada, July 14–17, 2005.

[4] W.–H. Press, S.–A. Teukolsky, W.–T. Vetterling and B.–P. Flannary *Numerical Recipes in C,* Cambridge University Press, New York, 1997.