We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Improving the efficiency of Runge-Kutta reintegration by means of the RKGL algorithm

Justin S. C. Prentice
Department of Applied Mathematics, University of Johannesburg
Johannesburg, Republic of South Africa

1. Introduction

Initial-value problems (IVPs) of the form

$$\frac{dy}{dx} = f(x,y) \qquad y(x_0) = y_0 \tag{1}$$

often arise in the simulation of physical systems, particularly if the system is time-dependent. The numerical solution of these problems is often achieved using an explicit *Runge-Kutta* (RK) method (Shampine, 1994; Butcher, 2000; Hairer et al., 2000), which typically involves a large number of evaluations of the function *f(x,y)*. This is the most significant factor that determines the efficiency, or lack thereof, of the RK method. If it is necessary to control the global error in the numerical solution, it may be necessary to use a *reintegration* algorithm, which, as will be shown, would require at least three applications of RK to the problem. Thus, it is easy to see that a reintegration process could be inefficient if a large number of evaluations of *f(x,y)* is required. We note here that the more accurate an RK method is, so the greater the number of evaluations of *f(x,y)* that is required. Demanding very strict global tolerance via reintegration could be particularly expensive in terms of computational effort. For this reason, *local error control* (to be discussed later) is preferred over global error control and, consequently, very little work has been done regarding reintegration. Indeed, Shampine has recently commented that such schemes "…are generally thought to be too expensive…for production codes" (Shampine, 2005). To the best of our knowledge, no attempt has been made to improve the efficiency of RK methods with regard to reintegration, which is the topic of this paper.

We have developed a numerical method for nonstiff IVPs, designated RKGL, which is a modification of a standard explicit RK method. The RKGL method has been designed to reduce the number of function evaluations, relative to the underlying RK method. The method also has an enhanced global order, relative to the RK method, which is a very powerful mechanism for improving efficiency in the context of reintegration.

In this article we will show (a) how RKGL can be used to enhance the performance of the reintegration algorithm, in comparison with RK methods, and (b) how RKGL can achieve better accuracy than RK, for equal computational effort. Additionally, we will introduce a

reintegration algorithm that we believe is original in nature. We must stress, however, that our emphasis is on the relative efficiencies of the two methods, and the reintegration algorithm used here is intended primarily as a vehicle to facilitate such efficiency analysis.

2. Scope and Structure

The objective of this article is to demonstrate the improvement in efficiency and accuracy of the RKGL algorithm relative to the underlying RK method, and this will be achieved via theoretical arguments and numerical examples. However, it is necessary that we also describe the RKGL algorithm in some detail, and indicate a few important properties of the algorithm. We will also describe the general idea of reintegration as a mechanism for global error control. Some mathematical detail is inevitable, but in this regard we will be as economical as possible. These discussions will be presented entirely in the next section, with the sections thereafter devoted to efficiency analysis and numerical work.

3. Relevant Concepts

In this section, we describe concepts relevant to the current paper, and introduce appropriate terminology and notation.

3.1 Runge-Kutta Methods

To solve an IVP of the form in (1) on an interval $[a,b]$, using an explicit RK method, requires that a set of discrete nodes $\{x_i\,;\, i = 0,1,\ldots,N\}$, with $x_0 = a$ and $x_N = b$, be defined on $[a,b]$. The numerical solution y_{i+1} at the node x_{i+1} is obtained via the explicit RK method by

$$y_{i+1} = y_i + h_i \sum_{j=1}^{s} \beta_j k_j \equiv y_i + h_i F\left(x_i y_i\right), \tag{2}$$

where

$$\begin{aligned} k_1 &= f\left(x_i, y_i\right) \\ k_2 &= f\left(x_i + \gamma_2 h_i, y_i + h_i a_{2,1} k_1\right) \\ &\vdots \\ k_s &= f\left(x_i + \gamma_s h_i, y_i + h_i\left(a_{s,1} k_1 + a_{s,2} k_2 + \ldots + a_{s,s-1} k_{s-1}\right)\right). \end{aligned} \tag{3}$$

In these equations, the function $F(x,y)$ has been implicitly defined, and the various coefficients $\{\alpha, \beta, \gamma\}$ are specific to the particular RK method being used. The parameter h_i is the spacing between x_i and x_{i+1}, and is known as the *stepsize*. We label the stepsize with a subscript since, in general, it need not have uniform magnitude. The parameter s indicates the number of *stages* of the method; each stage requires an evaluation of $f(x,y)$. An RK method is known as a *one-step* method since the solution y_{i+1} is computed using information at the previous node only. We note here that the RK method is termed *explicit* since y_{i+1} does

not appear on the right-hand side of (2); if it does, the method is termed *implicit*. In the remainder of this article, the abbreviation RK indicates an explicit method.
RK methods are known to be consistent, convergent and stable with respect to roundoff error (zero-stable). Moreover, an RK method of *order r*, denoted RK*r*, has global error

$$\Delta_i \equiv y_i - y(x_i) = O(h^r), \tag{4}$$

where $y(x_i)$ is the exact value at x_i, and h is the average stepsize on $[a,b]$. It is always true that $s \geq r$, so that greater accuracy in an RK method implies greater computational effort.
There does exist a class of RK methods, known as *embedded* methods, that offer greater efficiency if a lower-order and higher-order method need to be used in tandem (Butcher, 2003). Such scenarios arise typically in error control algorithms, as will be described later. From (2) we see that an RK method requires a linear combination of its stages; an embedded method has the property that two different linear combinations of the *same* stages yield two methods of different order. We usually speak of an RK(r,q) pair, which is an embedded method containing both RK*r* and RK*q*.

3.2 Gauss-Legendre Quadrature

Gauss-Legendre (GL) quadrature is an algorithm for numerically evaluating the integral of a continuous function (Burden & Faires, 2001). Indeed, we have

$$\int_u^v f(x, y(x))dx \approx h\sum_{j=1}^{m} C_j f(x_j, y(x_j)) \tag{5}$$

for m-point GL quadrature (denoted GLm). Here, the x_j are m nodes on $[u,v]$ (given by the roots of the mth degree Legendre polynomial on $[-1,1]$ and then translated to $[u,v]$); h is the average separation of these nodes; and the C_j are appropriate weights. GL quadrature is *open* quadrature in the sense that the quadrature nodes are interior to the interval of integration $[u,v]$. In (5), we write $y(x_i)$, although in the context of the RKGL algorithm we actually use y_i. The error associated with GLm quadrature is $O(h^{2m+1})$.

3.3 The RK*r*GL*m* Algorithm

The RK*r*GL*m* algorithm (Prentice, 2008; Prentice, 2009) is best described with reference to Figure 1. The interval of integration $[a,b]$ is subdivided into N' subintervals, denoted H_j where $j = 1, 2,\ldots, N'$. On each subinterval we define m nodes suitable for GLm quadrature. The numerical solution at these nodes (indicated RK in the figure) is obtained using RK; the solution at the endpoint of each subinterval H_j is determined using GLm. Defining $p \equiv m+1$, we have

$$y_{i+1} = y_i + h_i F(x_i y_i), \tag{6}$$

where $i = (j-1)p, (j-1)p+1,\ldots, (j-1)p+m-1$ at the RK nodes, and

$$y_{jp} = y_{(j-1)p} + h \sum_{i=(j-1)p+1}^{(j-1)p+m} C_i f(x_i, y_i) \tag{7}$$

at the GL nodes.

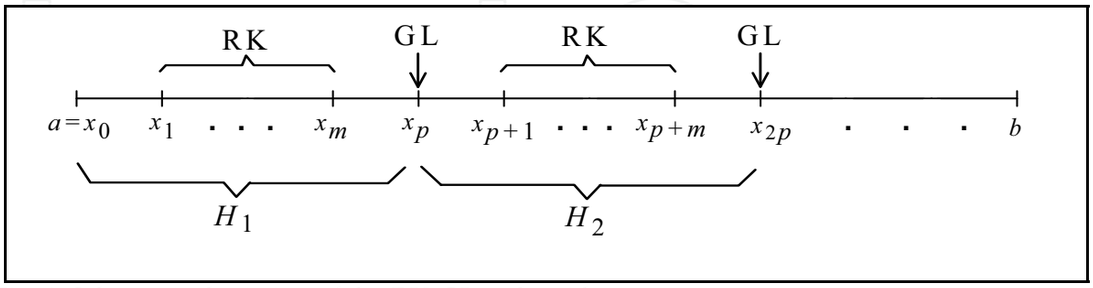

Fig. 1. Schematic depiction of the RK*r*GL*m* algorithm.

We have shown that RK*r*GL*m* is consistent, convergent and zero-stable. It is clear that RK*r*GL*m* does not require any evaluations of $f(x,y)$ at each GL node $\{x_p, x_{2p}$, etc.$\}$, which is a clear reduction in computational effort. Furthermore, the global error in RK*r*GL*m* has the form

$$\Delta_i = A_i h^{r+1} + B_i h^{2m} = O\left(h^{\min(r+1,2m)}\right), \tag{8}$$

so that if we choose r and m such that $2m \geq r+1$, the global error is of order h^{r+1}, which is one order better than the underlying RK*r* method.

In Figure 2 we show, for example, the global error for RK5 and RK5GL3 applied to a test problem (which will be described later). The growth of the RK error is evident, whereas the RKGL error is *quenched* (at each GL node; see Figure 1), thus slowing its rate of growth. The technicalities of this error quenching need not concern us here; it is due to the factor h in (7). Rather, this example shows how RKGL improves the global accuracy of its underlying RK method. Furthermore, since there is no need to evaluate $f(x,y)$ at each GL node, the RK5GL3 calculation in this example required less computational effort than RK5. Indeed, the relative computational effort for this example is 3/4, a saving of some 25%, and furthermore, the maximum global error is about five times smaller.

3.4 Local Error Control (LEC)

We must discuss the concept of local error control (Hairer et al., 2000), since it plays a role in the reintegration process.

For an RK*r* method, we define the local error at x_{i+1} by

$$\varepsilon_{i+1} \equiv \left[y(x_i) + h_i F(x_i, y(x_i))\right] - y(x_{i+1}) = O\left(h_i^{r+1}\right). \tag{9}$$

Note that the local error is one order higher than the global error. Note also that the exact value $y(x_i)$ is used in the RK method in the square brackets. As such, the local error is the RK

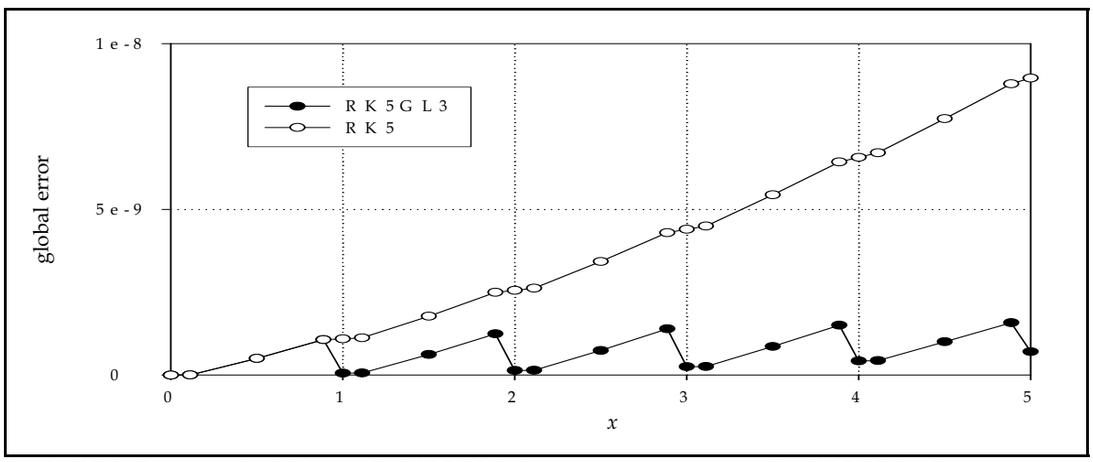

Fig. 2. Global errors for RK5 and RK5GL3. The greater accuracy of RK5GL3 is clear.

error made on the subinterval $[x_i,x_{i+1}]$, assuming that the exact solution is known at x_i. The local error is controlled by adjusting the stepsize h_i. If we assume

$$\varepsilon_{i+1} = L_{i+1} h_i^{r+1} , \tag{10}$$

where L_{i+1} is the local error coefficient, then control of the local error clearly requires a good estimate of L_{i+1}.
We will consider a type of error estimation that requires the use of two RK methods of differing order, such as a RK(r,q) pair. This will be described in section 3.5.2.

3.5 Reintegration

Reintegration relies on the fact that as the stepsize tends to zero, so the global error also tends to zero. A numerical method that exhibits this property is said to be *convergent*. Since RK and RKGL are known to be convergent, they are suitable for the application of a reintegration algorithm.
A reintegration algorithm typically consists of three phases:

1. Determining a node distribution in a systematic way.
2. Error estimation at these nodes.
3. Reintegration using a refined node distribution.

Phase 1 involves LEC and will be discussed at the end of this section. For now, we assume that a numerical solution has been obtained using some uniform stepsize h – which we term the *initial stepsize*. Assume also that a good estimate of the global error at each node has been made. If a tolerance δ is imposed, then a new stepsize h^* is determined from

$$h^* = 0.9 \left(\frac{\delta}{\max|G_i|} \right)^{1/r} , \tag{11}$$

where G_i is the constant of proportionality in (4), the so-called *global error coefficient* (this coefficient is dependent on x, but not on h). The factor 0.9 is a 'safety factor' to cater for the

possibility of underestimating the magnitude of G_i. The RK method is then reapplied using this new stepsize. Clearly, the need to reapply the method is the source of the computational effort that we hope to reduce using RKGL.
However, the control mechanism described here seeks to control the absolute error, whereas it is the relative error

$$\frac{y_i - y(x_i)}{y_i} \tag{12}$$

that we should rather attempt to control, since finite-precision computing devices distinguish between numerical values in a relative sense only. Hence, we have

$$\frac{|y_i - y(x_i)|}{|y_i|} = \left|\frac{G_i}{y_i}\right| h^r \leq \max\left(\left|\frac{G_i}{y_i}\right|\right) h^r . \tag{13}$$

Now, if we impose a tolerance δ on the relative error, we have the condition

$$\frac{|y_i - y(x_i)|}{|y_i|} \leq \delta \Rightarrow |y_i - y(x_i)| \leq \delta |y_i| \tag{14}$$

which becomes problematic when y_i is very close to zero (because the stepsize that would then be required would be intolerably small). To counter this, we replace the term on the right in (14) by $\max\{\delta_A, \delta_R |y_i|\}$ – where δ_A is a so-called *absolute tolerance* and δ_R is a *relative tolerance*, and these tolerances are not necessarily the same. Hence, the tolerance is δ_A when $\delta_R |y_i| < \delta_A$, and $\delta_R |y_i|$ otherwise. The expression for h^* now becomes

$$h^* = 0.9\left(\frac{\delta_R}{G_i^M}\right)^{1/r} \text{ where } G_i^M \equiv \max\left(\frac{|G_i|}{\max\left(\frac{\delta_A}{\delta_R}, |y_i|\right)}\right). \tag{15}$$

This ensures that G_i is never divided by a number smaller than δ_A/δ_R, and that, under this condition, the denominator is maximized – which, of course, yields the smallest upper bound for h^*. We often use a uniform tolerance δ, as in $\delta = \delta_A = \delta_R$.

3.5.1 Global Error Estimation

It is clear that the estimate of the global error (Phase 2) is a very important part of the reintegration algorithm. The ability to make a reliable estimate of this quantity is crucial to the effectiveness of the algorithm. The favoured approach is to use a method of higher order to obtain a solution that is then held to be more accurate than that obtained with the lower

order method. If $y_i(r)$ and $y_i(q)$ denote lower-order (RKr) and higher-order (RKq) solutions, respectively, at x_i, then we have

$$\begin{aligned} y_i(r) - y_i(q) &= y_i(r) - y(x_i) - \left(y_i(q) - y(x_i)\right) \\ &= G_i(r)h^r - G_i(q)h^q \\ &\approx G_i(r)h^r . \end{aligned} \tag{16}$$

Here, $y(x_i)$ is the exact solution, $G_i(r)$ and $G_i(q)$ indicate global error coefficients for the lower- and higher-order methods, respectively, and q is the order of the higher-order method (obviously, $q > r$). The approximation in (16) holds if h is sufficiently small. In other words, if methods of differing order are used to obtain numerical solutions at each node on the interval of integration, then (16) can be used to estimate the global error coefficient in the lower-order solution, which is then used in (11) or (15) to determine h^*. This method of reintegration – using two methods of differing order – will form the basis of the reintegration algorithm, designated **LHR**, that we will use in this paper. The **LHR** algorithm is based on the reintegration process described in section 3.5, and from this point onwards we will refer generically to that reintegration process as **LHR**, culminating in a complete description of **LHR** in section 5.3.

Until now, we have assumed that the initial stepsize h is known, and we have described the reintegration algorithm accordingly. But how does one choose a suitable initial stepsize h? If h is too large the asymptotic expressions for the error, as in (4), become unreliable since higher-order terms in the error expansion make significant contributions; if h is too small, we might use many more nodes than is necessary, which is inefficient, and if h is extremely small, problems in the RK method itself, due to roundoff, will persist.

A possible solution to this problem involves the use of LEC, before estimating the global error. This approach allows an appropriate stepsize to be determined systematically. Before discussing this, we must describe error estimation in the local error control procedure.

3.5.2 Local Error Estimation via Local Extrapolation

Local error estimation can be achieved by using a higher-order method, say RKq, in similar manner to that described previously. Indeed, we have

$$h_i^* = 0.9\left(\frac{\max\left(\delta, \delta|y_{i+1}|\right)}{|L_{i+1}|}\right)^{1/r+1} \tag{17}$$

into which relative LEC has been incorporated, and where L_{i+1} is estimated from

$$\begin{aligned} y_{i+1}(r) - y_{i+1}(q) &= L_{i+1}(r)h_i^{r+1} - L_{i+1}(q)h_i^{q+1} \\ &\approx L_{i+1}(r)h_i^{r+1} . \end{aligned} \tag{18}$$

We have retained the subscript on the stepsize to emphasize that, due to the nature of this type of LEC, the stepsize can vary from subinterval to subinterval – it does not need to be constant, as we have assumed previously. We point out that if the local error estimate is less

than or equal to the tolerance, then no stepsize adjustment is necessary, and we proceed to the next subinterval. Nevertheless, it is certainly possible to determine a new stepsize, even if the error estimate satisfies the tolerance, and a new solution using this stepsize can be determined. We refer to this procedure as *forced* LEC (FLEC). In this case, the resultant stepsizes on $[a,b]$ are all consistent with the desired tolerance; none of them are any smaller than is necessary.

A particularly important feature of this algorithm concerns the propagation of the higher-order solution. Since the algorithm relies on the exact solution being known at x_i, as in (9), we must always use the more accurate RKq solution at x_i as input for both the RKr and RKq methods. Hence, the RKr and RKq solutions at x_{i+1} are both generated using $y_i(q)$. This feature is known as *local extrapolation* (LeVeque, 2007).

3.5.3 Starting stepsize

To implement either the LEC or FLEC algorithm, it is necessary to estimate a *starting stepsize* h_0. A practical way to do this is to assume $L_1 = 1$, so that

$$h_0 = (\delta_{loc})^{1/r+1} \tag{19}$$

if δ_{loc} is the desired accuracy in the local error.

3.5.4 Initial Stepsize Estimation

The idea here is to use FLEC with a moderate tolerance to obtain a node distribution, and then to compute an average stepsize that can be used in Phase 2 of the reintegration algorithm. This is the so-called initial stepsize referred to previously, not to be confused with the starting stepsize described in the previous section. We propose that, if ultimately a global tolerance of δ is required, then a local tolerance of $\delta_{loc} = \sqrt{\delta}$ should be used (we are assuming, of course, that $\delta < 1$). Assume that this results in a non-uniform node distribution $\{x_0,x_1,x_2,\ldots,x_N\}$, which has an average stepsize h. We now assume that the solution has a maximum global error of $N\delta_{loc}$ so that an error coefficient may be determined from $G = N\delta_{loc}/h^r$. This allows a new stepsize to be determined, as in (15), where we treat δ_{loc} as a global tolerance, and this new stepsize is then used as the initial stepsize.

In this section, we have described the principles behind typical reintegration, with reference to the RK method. In the next section, we study the efficiency of reintegration, including RKGL-based reintegration.

4. Efficiency Analysis

Here, we will present a theoretical analysis of the relative efficiencies of the **LHR** reintegration method discussed previously, for both RK and RKGL. We will attempt to count both the number of evaluations of $f(x,y)$, and the number of arithmetical operations involved in each method. In the next section, we will present numerical work demonstrating this efficiency analysis.

To facilitate our efficiency analysis, we define various symbols in Table 1.

Symbol	Meaning		Symbol	Meaning
D	Length of interval of integration $[a,b]$		A_f	Number of arithmetic operations in $f(x,y)$
Δ_{RK}	Global error using RKr		A_1	Number of arithmetic operations, per node, using RKrGLm
Δ_{RKGL}	Global error using RKrGLm		A_2	Number of arithmetic operations, per node, using RKr
N_1	Number of nodes on $[a,b]$ for RKrGLm, excluding first node		Φ_1	N_1F_1 = total number of function evaluations on $[a,b]$, using RKrGLm
N_2	Number of nodes on $[a,b]$ for RKr, excluding first node		Φ_2	N_2F_2 = total number of function evaluations on $[a,b]$, using RKr
F_1	Number of evaluations of $f(x,y)$, per node, using RKrGLm		Ψ_1	N_1A_1 = total number of arithmetic operations on $[a,b]$, using RKrGLm
F_2	Number of evaluations of $f(x,y)$, per node, using RKr		Ψ_2	N_2A_2 = total number of arithmetic operations on $[a,b]$, using RKr

Table 1. Definition of miscellaneous symbols to be used in the efficiency analysis.

Consider the first subinterval H_1 for RKrGLm (see Figure 1). There are m + 1 nodes on this interval at which numerical solutions must be determined (it is not necessary to determine y_0 since this is the given initial value). RKr, with s stages, is applied to find solutions at m of these nodes, and the GLm algorithm requires the evaluation of $f(x,y)$ at the mth node itself. This means that ms + 1 function evaluations are required by RKrGLm on H_1. This holds for all subsequent subintervals. Since there are m + 1 nodes on each subinterval (we exclude the first node, since we regard it as being part of the previous subinterval), we have

$$\begin{aligned} F_1 &= \frac{ms+1}{m+1} \\ F_2 &= s \end{aligned} \tag{20}$$

where the expression for F_2 is due to the fact that RKr is an s-stage method. Note that $F_1 < F_2$ for $s > 1$, and that $F_1 = F_2$ only when $s = 1$.
Referring to the RKr method in (2) and (3), we see that multiplication of $f(x,y)$ by h is performed $s - 1$ times; in the argument of $f(x,y)$, multiplication by h occurs $s - 1$ times, and addition of γh to x occurs $s - 1$ times; in the y-component of $f(x,y)$ there are $s(s - 1)/2$ multiplications and an equal number of additions; and finally, in (2), there are s multiplications and s additions. This all gives

$$A_2 = s^2 + 4s - 2 + sA_f \,. \tag{21}$$

For RKrGLm on each subinterval H_i, we use RKr m times and then we compute the solution at the endpoint (see (5)) by means of one evaluation of $f(x,y)$, m multiplications (by the weights), $m - 1$ additions, one multiplication by h and one more addition. Hence,

$$\begin{aligned} A_1 &= \frac{m\left(s^2 + 4s - 2 + sA_f\right) + 2m + 1 + A_f}{m+1} \\ &= \frac{mA_2 + 2m + 1 + A_f}{m+1}. \end{aligned} \tag{22}$$

Note that

$$\begin{aligned} F_1 &= \Omega_1 A_1 \\ F_2 &= \Omega_2 A_2 \end{aligned} \tag{23}$$

where

$$\begin{aligned} \Omega_1 &= \Omega_1\left(m, s, A_f\right) = \frac{ms+1}{mA_2 + 2m + 1 + A_f} \\ \Omega_2 &= \Omega_2\left(s, A_f\right) = \frac{s}{s^2 + 4s - 2 + sA_f} \end{aligned} \tag{24}$$

are method- and problem-dependent proportionality constants.
Since we are interested in comparing the efficiencies of the two methods, we must consider the ratios of their respective arithmetical operations, function evaluations and global errors. Hence, we define the quantities

$$R_F \equiv \frac{\Phi_1}{\Phi_2}$$
$$R_A \equiv \frac{\Psi_1}{\Psi_2} \tag{25}$$
$$R_\Delta \equiv \frac{\Delta_{\mathrm{RKGL}}}{\Delta_{\mathrm{RK}}}$$

The first of these is the ratio of the total number of function evaluations, and the second is the ratio of the total number of arithmetical operations, over the whole interval [*a,b*]. As such, these ratios measure the relative efficiency of the two methods. The third ratio is that of the maximum global errors for the two methods. For R_Δ we have

$$R_\Delta = \frac{G_1 {h_1}^{r+1}}{G_2 {h_2}^{r}} = \left(\frac{G_1 \theta^r}{G_2} \right) h_1 \tag{26}$$

where

$$h_1 = \frac{D}{N_1} \qquad h_2 = \frac{D}{N_2} \qquad \theta \equiv \frac{h_1}{h_2} = \frac{N_2}{N_1}. \tag{27}$$

We see that R_Δ must tend to zero as h_1 tends to zero; this simply means that as Δ_{RKGL} is made smaller (by reducing h_1), so RKGL inevitably becomes more accurate than RK. In (26), the coefficients G_1 and G_2 can be absolute error coefficients, as in (11), or relative error coefficients, such as G_i^M in (15).

To make a sensible comparison of the two methods, we require that the global error (absolute or relative) of each satisfy some user-defined tolerance δ, and then compute the ratio R_F. If both methods satisfy the tolerance, we have $\Delta_{\mathrm{RKGL}} = \Delta_{\mathrm{RK}} = \delta$, so that $R_\Delta = 1$. Hence,

$$R_\Delta = 1 \Rightarrow \left(\frac{G_1 h_1}{G_2} \right)^{1/r} = \frac{h_2}{h_1} = \frac{N_1}{N_2}. \tag{28}$$

Using $\Phi_1 = N_1 F_1$, $\Phi_2 = N_2 F_2$ and (27), we find

$$\begin{aligned} R_F &= \frac{N_1}{N_2}\frac{F_1}{F_2} = \left[\frac{F_1}{F_2} \left(\frac{G_1}{G_2} \right)^{1/r} \right] (h_1)^{1/r} \\ &= \left[\frac{F_1}{F_2} \left(\frac{G_1}{G_2} \right)^{1/r} \left(\frac{1}{G_1} \right)^{1/r^2+r} \right] (\Delta_{\mathrm{RKGL}})^{1/r^2+r} \end{aligned} \tag{29}$$

where we have used

$$\Delta_{\mathrm{RKGL}} = G_1 h_1^{r+1} \Rightarrow (h_1)^{1/r} = \left[\left(\frac{1}{G_1}\right)^{1/r^2+r}\right](\Delta_{\mathrm{RKGL}})^{1/r^2+r}. \tag{30}$$

So we have

$$R_F \propto (\delta)^{1/r^2+r}. \tag{31}$$

This is an important result. It shows that as we seek stricter tolerances, so R_F becomes smaller, i.e. RK*r*GL*m* becomes ever more efficient than RK*r*. For example, if r = 4 and δ = 10^{-5}, then $R_F \propto 0.56$, and if $\delta = 10^{-10}$, then $R_F \propto 0.32$.

We are now in a position to study the efficiency of the **LHR** method. Such efficiency will be studied in terms of arithmetical operations; by virtue the linear relation (23), our analysis holds for function evaluations as well.

4.1 Efficiency of the LHR Algorithm

We have described the essentials of the **LHR** algorithm with regard to RK; the implementation using RKGL is similar. We use two methods, RK*r*GL*m* and RK*q*GL*m*, where the latter has higher order (i.e. $r < q$, as before). Note that both methods use GL*m* quadrature, meaning that they have a common node distribution. Hence, global error coefficients can be determined at each node, and a new stepsize satisfying a tolerance δ can be found.

Assume that an initial average stepsize h_1 is used to find numerical solutions using RK*r*GL*m* and RK*q*GL*m* (i.e. we are considering Phase 2 of **LHR**). We refer to an average stepsize since the RKGL nodes are not uniformly spaced. Hence, we have

$$\begin{aligned} \Psi_1^r &= N_1 A_1^r \\ \Psi_1^q &= N_1 A_1^q \end{aligned} \tag{32}$$

where the superscripts indicate RK*r*GL*m* or RK*q*GL*m*. A new stepsize gives a new node distribution of N_1^* nodes, where

$$N_1^* = \frac{D}{h_1^*}, \tag{33}$$

so that

$$\Psi_1^{r^*} = N_1^* A_1^r. \tag{34}$$

This is the number of arithmetic operations required by the reintegration phase of the **LHR** algorithm (Phase 3). The total number of arithmetic operations, for Phases 2 and 3, in the RKGL implementation of **LHR** is then

$$\Psi_1 = \Psi_1^r + \Psi_1^q + \Psi_1^* \tag{35}$$

Analogous quantities may be derived for the RK*r* and RK*q* implementation, using the same initial stepsize h_1 (for RK this stepsize is uniform), giving

$$\Psi_2 = \Psi_2^r + \Psi_2^q + \Psi_2^* . \tag{36}$$

It must be noted that the new stepsize h_2^* is not necessarily equal to h_1^*.
Now consider

$$\frac{\Psi_1^r}{\Psi_2^r} = \frac{A_1^r}{A_2^r}, \quad \frac{\Psi_1^q}{\Psi_2^q} = \frac{A_1^q}{A_2^q}, \quad \frac{\Psi_1^*}{\Psi_2^*} = \frac{N_1^* A_1^r}{N_2^* A_2^r} . \tag{37}$$

The first two of these ratios give the relative efficiencies (in terms of arithmetical operations) for RK*r*GL*m* and RK*r*, and RK*q*GL*m* and RK*q*, as regards the error estimation phase of **LHR** (Phase 2). The third ratio gives the relative efficiency of RK*r*GL*m* and RK*r*, as regards Phase 3 of **LHR**. Since RKGL requires fewer arithmetical operations per node than RK, we have that the first two ratios are certainly never greater than one. The factor in the third ratio leads, as in (29)−(31), to

$$\frac{\Psi_1^*}{\Psi_2^*} \propto (\delta)^{1/r^2+r} . \tag{38}$$

All of this shows that the RKGL implementation of **LHR** must become more efficient than the RK implementation, as stricter tolerances are imposed.
Additionally, it is possible to include a *quality control* mechanism in **LHR**. In such mechanism, we use RK*q*GL*m* or RK*q* to obtain a numerical solution using stepsizes h_1^* or h_2^*. This solution can then be used to check the error in the solution obtained with RK*r* or RK*r*GL*m*. If this quality control reveals that the imposed tolerance has not been satisfied, then a new stepsize can be determined and further reintegration can be done. Quality control necessarily requires $N_1^* A_1^q$ or $N_2^* A_2^q$ additional arithmetical operations. Additional reintegration, if required, may be regarded as a *fourth phase* of **LHR**, and will require more than $N_1^* A_1^r + N_1^* A_1^q$ or $N_2^* A_2^r + N_2^* A_2^q$ additional arithmetical operations.

4.2 Initial Stepsize

To implement FLEC using RK*r* and RK*q* requires $A_2^r + 2A_2^q$ arithmetical operations per node. Using RK*r*GL*m* and RK*q*GL*m* requires $A_1^r + 2A_1^q$ operations per node. Hence, the RKGL implementation is more efficient than the RK implementation. Of course, for the sake of comparison, we have assumed, as previously, that the average stepsize for the two methods is the same. This is not unreasonable, since both methods (RK*r* and RK*r*GL*m*) have the same order (*r* + 1) in their local errors.

A note with regard to RKGL local error control: consider the subinterval H_1. We obtain solutions at the nodes using RK*r*GL*m* and RK*q*GL*m*, with y_0 as input. We estimate the error at each node; effectively, this is an estimate of the global error on H_1. We assume that these errors are proportional to h_1^{r+1}, where h_1 is the average stepsize on H_1, and then determine a new stepsize h_1^*. The length of the new subinterval is $(m+1)h_1^*$. RK*q*GL*m* is then used to find solutions at the nodes on this new subinterval, and the solution so obtained at x_p is used as input for RK*r*GL*m* and RK*q*GL*m* on the next subinterval H_2. In a sense, then, we control global error per subinterval, with local extrapolation at the endpoint of each subinterval, exploiting the fact that the global order of RK*r*GL*m* is the same as the local order of RK*r*.
It is clear from the foregoing analysis that the three-phase reintegration algorithm will most likely be more efficient when implemented using RK*r*GL*m*, than when using RK*r*. This is due partly to the design of RK*r*GL*m*, through which fewer arithmetical operations per node are required, and partly to the higher global order of RK*r*GL*m*. In the next section we will demonstrate this superior efficiency.

4.3 Accuracy for Equal Effort

It is instructive to consider the accuracy of the two methods, assuming equal computational effort. In such case we have $R_A = 1$, and so

$$R_A = 1 \Rightarrow N_1 A_1 = N_2 A_2 \Rightarrow h_1 = \left(\frac{A_1}{A_2}\right) h_2 \,. \tag{39}$$

Using $\Delta_{\mathrm{RKGL}} = G_1 h_1^{r+1}$ and $\Delta_{\mathrm{RK}} = G_2 h_2^r$ gives

$$\Delta_{\mathrm{RKGL}} = \left[\left(\frac{A_1}{A_2}\right)^{r+1}\left(\frac{1}{G_2}\right)^{1+1/r} G_1\right](\Delta_{\mathrm{RK}})^{1+1/r} \tag{40}$$

which shows that the RKGL error is improved, relative to the RK error, by a factor of $(\Delta_{\mathrm{RK}})^{1/r}$.

5. Numerical Examples

We will use the equations

$$\begin{aligned} &\frac{dy}{dx} = \frac{y}{4}\left(1 - \frac{y}{20}\right) \text{ on } [0,20] \text{ with } y(0) = 1 \\ &\frac{dy}{dx} = y \text{ on } [0,10] \text{ with } y(0) = 1 \end{aligned} \tag{41}$$

to demonstrate the theoretical results obtained in the previous section. These have the solutions

$$y(x) = \frac{20}{1+19e^{-x/4}}$$
$$y(x) = e^{x} \tag{42}$$

respectively. The first of these (which we call P1) is one of the test equations used by Hull et al. (Hull et al, 1972), and is also the equation that we used to generate the plots in Figure 2. The second (P2) is the Dahlquist equation (with $\lambda = 1$).

We will use three sets of values for r and q : $(r,q) = (2,3)$, $(3,4)$ and $(4,5)$. In other words, we have the pairs of methods RK2 and RK3, RK3 and RK4, RK4 and RK5. In each case, the higher order method is used for error estimation. Moreover, the methods RK2, RK3 and RK4 are independent – not embedded – so that RK2 and RK3 constitute a tandem pair, as do RK3 and RK4 (Butcher, 2003). The pair RK4 and RK5 is an embedded pair, due to Fehlberg (Kincaid & Cheney, 2002), and so, to avoid notational confusion, we will indicate this pair by RKF4 and RKF5. Note that RK4 and RKF4 are not the same. The corresponding RKGL pairs are RK2GL2 and RK3GL2, RK3GL3 and RK4GL3, RKF4GL3 and RKF5GL3. Note that the choice of m in each case is the smallest m such that $r + 1 \leq 2m$, subject to the condition that m must be the same for both methods within any given pair.

To begin with, we determine the quantities F_1, F_2, A_1, A_2, Ω_1 and Ω_2 for these methods, using $A_f = 4$ (see the RHS of P1 in (41)). These are all shown in Tables 2 and 3. In these tables, s denotes the number of stages in the RK method.

s	F_1 ($m = 2$)	A_1 ($m = 2$)	F_1 ($m = 3$)	A_1 ($m = 3$)	F_2	A_2
2	1.67	15.00	1.75	16.25	2	18
3	2.33	23.67	2.50	26.00	3	31
4	3.00	33.67	3.25	37.25	4	46
5	3.67	45.00	4.00	50.00	5	63
6	4.33	57.67	4.75	64.25	6	82

Table 2. Number of function evaluations and arithmetical operations per node, with $A_f = 4$.

s	Ω_1 ($m = 2$)	Ω_1 ($m = 3$)	Ω_2
2	0.1111	0.1077	0.1111
3	0.0986	0.0962	0.0968
4	0.0891	0.0872	0.0870
5	0.0815	0.0800	0.0794
6	0.0715	0.0739	0.0732

Table 3. Proportionality constants Ω_1 and Ω_2, with $A_f = 4$.

We see in Table 3 that the constants of proportionality between the number of function evaluations and arithmetical operations are essentially the same for RK and RKGL, for each of the values of s considered.

In Table 4 we show the ratio of the number of function evaluations and arithmetical operations per node for RK and RKGL, again with $A_f = 4$.

s	F_1/F_2 (m = 2)	A_1/A_2 (m = 2)	F_1/F_2 (m = 3)	A_1/A_2 (m = 3)
2	0.833	0.833	0.875	0.903
3	0.779	0.763	0.833	0.839
4	0.750	0.732	0.813	0.810
5	0.733	0.714	0.800	0.794
6	0.722	0.703	0.792	0.784

Table 4. Ratio of number of function evaluations and arithmetical operations per node for RK and RKGL, with $A_f = 4$.

From Table 4 we see that, as far as function evaluations and arithmetical operations per node are concerned, RKGL is always more efficient than RK. The best case here is when $s = 6$ and $m = 2$, for which the ratio of arithmetical operations per node is about 70%, and that of function evaluations per node is about 72%. For phases 1 and 2 of the reintegration algorithm, where the number of nodes used by RK and RKGL is essentially the same, the ratios in Table 4 are indicative of the relative effort of the two methods. Clearly, the gain in using RKGL is in the vicinity of 10% to 30%. It could be argued that this is not actually all that impressive – indeed, as we will see in the next section, the most significant gain is to be had in Phase 3 of the reintegration algorithm, where the higher order of RKGL is exploited.
In problem P2 we have $A_f = 0$, and, for completeness' sake, the appropriate parameters are shown in Tables 5−7.

s	F_1 (m = 2)	A_1 (m = 2)	F_1 (m = 3)	A_1 (m = 3)	F_2	A_2
2	1.67	8.33	1.75	9.25	2	10
3	2.33	14.33	2.50	16.00	3	19
4	3.00	21.67	3.25	24.25	4	30
5	3.67	30.33	4.00	34.00	5	43
6	4.33	40.33	4.75	45.25	6	58

Table 5. Number of function evaluations and arithmetical operations per node, with $A_f = 0$.

s	Ω_1 (m = 2)	Ω_1 (m = 3)	Ω_2
2	0.2000	0.1892	0.2000
3	0.1628	0.1563	0.1579
4	0.1385	0.1340	0.1333
5	0.1209	0.1176	0.1163
6	0.1074	0.1050	0.1034

Table 6. Proportionality constants Ω_1 and Ω_2, with $A_f = 0$.

s	F_1/F_2 (m = 2)	A_1/A_2 (m = 2)	F_1/F_2 (m = 3)	A_1/A_2 (m = 3)
2	0.833	0.833	0.875	0.925
3	0.779	0.754	0.833	0.842
4	0.750	0.722	0.813	0.808
5	0.733	0.705	0.800	0.791
6	0.722	0.695	0.792	0.780

Table 7. Ratio of number of function evaluations and arithmetical operations per node for RK and RKGL, with $A_f = 0$.

5.1. Efficiency Curves

In Figures 3 and 4 we show R_A as a function of the imposed tolerance δ, for each of the test problems.

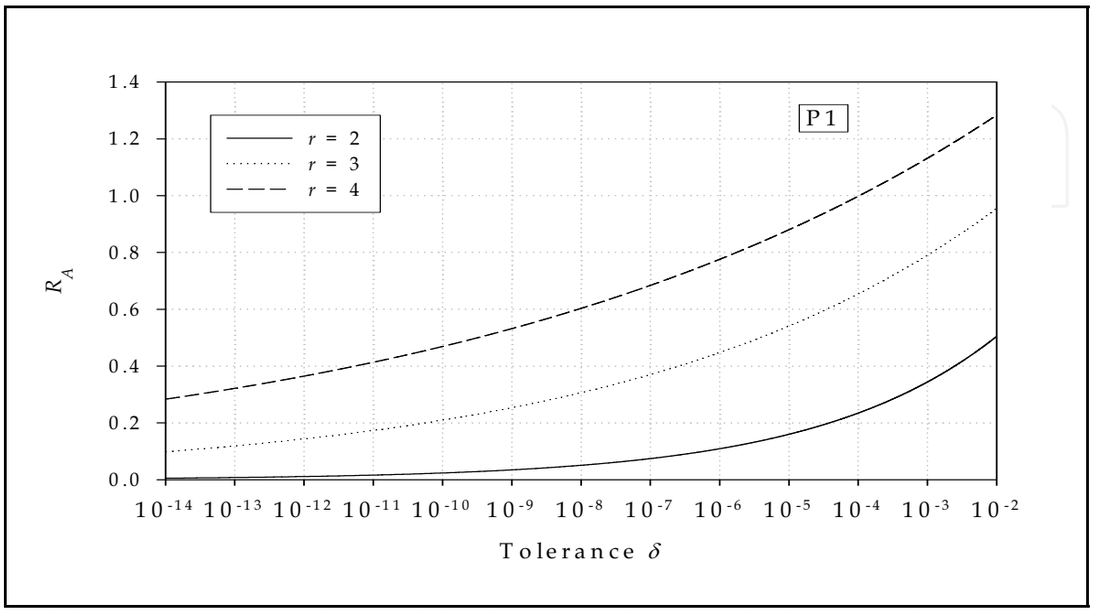

Fig. 3. R_A as a function of tolerance δ, for the indicated values of r, for test problem P1.

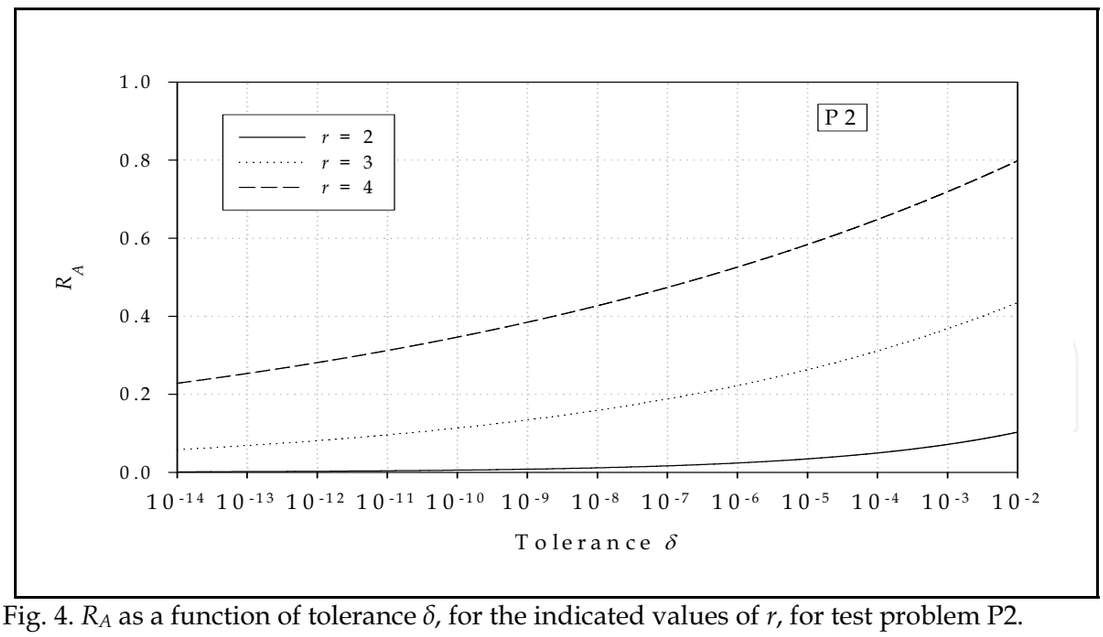

Fig. 4. R_A as a function of tolerance δ, for the indicated values of r, for test problem P2.

In each of these figures, two features are obvious: R_A decreases as δ decreases, and, for any δ, R_A increases with increasing r. Both of these features are easily understood in terms of (38). Note that $r^2 + r$ equals 6, 12 and 20 for the values of r considered here. We see that R_A is less

than one for all values of r and δ for P2, but for P1 the RKF4GL3 method is somewhat less efficient than RKF4 for large tolerances. This is due to the fact that, in this case, the global error coefficient for RKF4GL3 is twice as large as that for RKF4. We see in (29) that the ratio of global error coefficients does play a role in determining R_A. For the smallest tolerance considered here, R_A is approximately 0.3 or less, for all r in both test problems, and for r = 2 we have R_A = 0.0052 (P1) and R_A = 0.0013 (P2). These results serve to indicate just how much more efficient RKGL can be, particularly when very strict tolerances are imposed on the problem.

5.2. Equal Effort

From (40) we have

$$\ln(\Delta_{\mathrm{RKGL}}) = \left(1+\frac{1}{r}\right)\ln(\Delta_{\mathrm{RK}}) + \ln\left[\left(\frac{A_1}{A_2}\right)^{r+1}\left(\frac{1}{G_2}\right)^{1+1/r} G_1\right] \tag{43}$$

so that a plot of $\ln(\Delta_{\mathrm{RKGL}})$ against $\ln(\Delta_{\mathrm{RK}})$ will yield a straight line with slope $1+1/r$. Recall that this holds for the case of equal effort (R_A = 1). In Table 8 we show numerical results for the test problems.

P1		
r	**1+1/*r* (theory)**	**1+1/*r* (actual)**
2	1.50	1.5002
3	1.33	1.3331
4	1.25	1.2536
P2		
r	**1+1/*r* (theory)**	**1+1/*r* (actual)**
2	1.50	1.4910
3	1.33	1.3303
4	1.25	1.2439

Table 8. Slopes of equation (43) for the test problems.

There is good agreement between theory and experiment. As an example of the improved accuracy for RKGL, we found that for problem P1 with r = 3, $\Delta_{\mathrm{RK}} = 6\times10^{-10}$ and $\Delta_{\mathrm{RKGL}} = 2\times10^{-12}$.

5.3. The LHR Algorithm

Lastly, we apply the complete **LHR** algorithm to the test problems. We impose tolerances of $\delta = 10^{-6}$ and $\delta = 10^{-12}$ on the relative global error. The four phases, as described previously, of **LHR** comprise the following:

1. Phase 1 – application of FLEC with a moderate tolerance of $\sqrt{\delta}$ on the relative local error.

2. Phase 2 – determining relative global error coefficients using a uniform node distribution with average stepsize determined from Phase 1.
3. Phase 3 – reintegration with a new stepsize determined from Phase 2, and quality control.
4. Phase 4 – Further reintegration, if necessary, also with quality control.

The results are summarised in Tables 9−12, where we show the number of arithmetical operations for each phase, and the ratio R_A. Of course, the number of function evaluations is easily obtained from these data by means of the proportionality constants in Tables 3 and 6.

P1 $\delta = 10^{-6}$						
	***r* = 2**		***r* = 3**		***r* = 4**	
	RK	**RKGL**	**RK**	**RKGL**	**RK**	**RKGL**
Phase 1	1520	748	984	804	656	1028
Phase 2	4067	1160	1232	1012	492	1028
Phase 3	105154	16356	11550	5819	1886	1799
Phase 4	0	19952	0	8349	2296	2056
Total	**110741**	**38216**	**13766**	**15984**	**5330**	**5911**
$R_A = \Psi_1/\Psi_2$	0.345 (0.156)		1.161 (0.504)		1.109 (0.895)	

Table 9. Arithmetical operations for **LHR** applied to P1 with $\delta = 10^{-6}$.

P1 $\delta = 10^{-12}$						
	***r* = 2**		***r* = 3**		***r* = 4**	
	RK	**RKGL**	**RK**	**RKGL**	**RK**	**RKGL**
Phase 1	15440	2992	4920	2412	2296	2056
Phase 2	131418	6844	10549	3542	2296	1799
Phase 3	107845227	1938476	1246014	240603	70520	35466
Total	**107992085**	**1948312**	**1261483**	**246557**	**75112**	**39321**
$R_A = \Psi_1/\Psi_2$	0.018 (0.018)		0.195 (0.193)		0.523 (0.503)	

Table 10. Arithmetical operations for **LHR** applied to P1 with $\delta = 10^{-12}$.

P2 $\delta = 10^{-6}$						
	***r* = 2**		***r* = 3**		***r* = 4**	
	RK	**RKGL**	**RK**	**RKGL**	**RK**	**RKGL**
Phase 1	2784	2553	2054	3612	1044	2172
Phase 2	12818	6460	3822	6279	928	2172
Phase 3	410959	43316	38514	17388	5394	4525
Phase 4	0	0	0	0	6554	0
Total	**426561**	**52329**	**44390**	**27279**	**13920**	**8869**
$R_A = \Psi_1/\Psi_2$	0.123 (0.105)		0.615 (0.451)		0.637 (0.839)	

Table 11. Arithmetical operations for **LHR** applied to P2 with $\delta = 10^{-6}$.

	P2 $\delta = 10^{-12}$					
	***r* = 2**		***r* = 3**		***r* = 4**	
	RK	**RKGL**	**RK**	**RKGL**	**RK**	**RKGL**
Phase 1	29232	11322	12403	11610	5104	6154
Phase 2	435841	46784	41503	26565	6612	7240
Phase 3	415844688	4466920	4050340	576863	209148	80545
Total	**416309761**	**4525026**	**4104246**	**615038**	**220864**	**93939**
$R_A = \Psi_1/\Psi_2$	0.011 (0.011)		0.150 (0.142)		0.425 (0.385)	

Table 12. Arithmetical operations for **LHR** applied to P2 with $\delta = 10^{-12}$ **.**

In these tables, the ratio R_A is the ratio of the total number of arithmetic operations (indicated in bold) for each value of *r*. The ratio R_A in parentheses is that for Phase 3 only, except for *r* = 4 in Table 9, where it has been computed for Phase 4. Phase 4 was required in Table 9 (for the RKGL methods) and in Table 11 (for RKF4), and in those cases the ratio R_A includes the contribution from Phase 4. The arithmetical operations count for Phase 3 includes quality control, i.e. the contribution due to the higher order method.

All values of R_A for Phase 3 only are less than one, and in only two other cases the overall value of R_A is greater than one. In both of these cases, seen in Table 9, this is due to the need for Phase 4 reintegration. Generally, the RKGL implementation of **LHR** is more efficient than the RK implementation. This is particularly evident when *r* and δ are small, as in tables 10 and 12. The preliminary phases (1 and 2) of **LHR** contribute most significantly to the overall computational effort when δ is relatively large, but when δ is small the computational load is due almost entirely to Phase 3 (and Phase 4, if necessary).

We have confirmed that all applications of **LHR** described in the above tables have yielded numerical solutions that satisfy the imposed tolerances. In this sense, **LHR** has proved to be 100% effective. These results are shown in Table 13. In Figure 5 we show, for example, relative global error profiles for both the RK and RKGL implementations of **LHR**, with regard to problem P1. For the sake of clarity, we only show the solutions obtained by RK3GL3 and RKF4GL3 in Figure 6. The other error profiles are similar. Note the 'zigzag' character evident in Figure 2 is also present in these solutions. Due to space restrictions, we have not shown similar plots for problem P2.

		***r* = 2**		***r* = 3**		***r* = 4**	
	δ	**RK**	**RKGL**	**RK**	**RKGL**	**RK**	**RKGL**
Δ_{max} **(P1)**	10^{-6}	8.5×10^{-7}	7.3×10^{-7} ♦	9.2×10^{-7}	7.1×10^{-7} ♦	6.7×10^{-7} ♦	9.5×10^{-7} ♦
	10^{-12}	8.2×10^{-13}	8.1×10^{-13}	7.6×10^{-13}	9.7×10^{-13}	8.8×10^{-13}	9.3×10^{-13}
Δ_{max} **(P2)**	10^{-6}	8.3×10^{-7}	8.0×10^{-7}	8.5×10^{-7}	7.8×10^{-7}	7.0×10^{-7} ♦	9.8×10^{-7}
	10^{-12}	8.0×10^{-13}	7.5×10^{-13}	7.5×10^{-13}	6.9×10^{-13}	7.5×10^{-13}	7.6×10^{-13}

Table 13. Maximum relative global error Δ_{max} using **LHR**, for the indicated tolerances δ. The symbol ♦ indicates that Phase 4 of **LHR** was required.

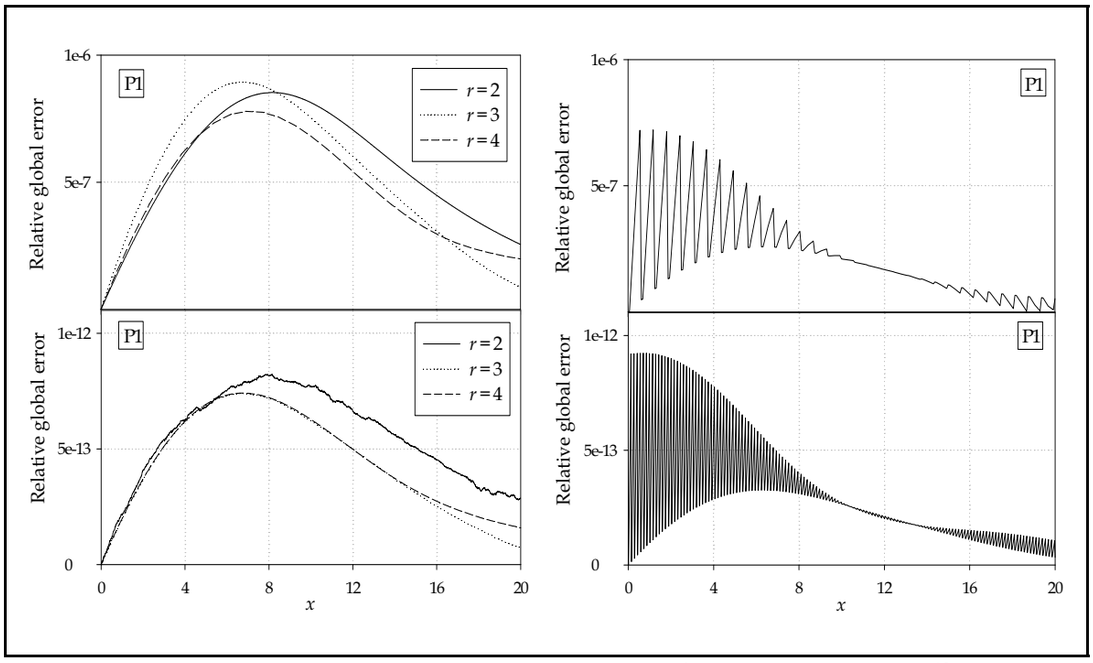

Fig. 5. (Left) Relative global errors for tolerances $\delta = 10^{-6}$ (upper) and $\delta = 10^{-12}$ (lower), for the RK version of **LHR**, applied to problem P1. (Right) Relative global errors for tolerances $\delta = 10^{-6}$ (upper) and $\delta = 10^{-12}$ (lower), for the RKGL version of **LHR**, applied to problem P1. Upper plot: RK3GL3, lower plot: RKF4GL3.

6. Conclusion and Scope for Further Work

We have studied the RKGL algorithm for the numerical solution of IVPs, with regard to improving the efficiency of global error control via reintegration. Our theoretical investigations have shown that the RK*r*GL*m* method requires fewer arithmetical operations and function evaluations per node, than its underlying RK*r* method. Moreover, since the RK*r*GL*m* method is of one order higher than RK*r*, we have shown that the relative efficiency of the two methods is proportional to the (r^2+*r*)th root of the user-defined tolerance imposed on the problem. Hence, as this tolerance becomes stricter, so the RK*r*GL*m* method becomes more efficient. This effect is more pronounced for small *r*, which is of great benefit, since it is usually true that lower order methods are computationally expensive.
In a similar vein, we have shown that when computational effort is equal, the accuracy of the RK*r*GL*m* method can be expected to be better than that of RK*r*.
Various numerical experiments have demonstrated these properties; in particular, the **LHR** algorithm for controlling relative global error has been shown to be generally more efficient when implemented using RK*r*GL*m*, than when using RK*r*.

6.1. Further Work

There are a number of issues that need to be considered with regard to the research presented in this paper:

- The need for the use of Phase 4 suggests that the global error coefficient has not been correctly estimated in Phase 3. Of course, Phase 4 represents a correction phase and so is important, but it is also computationally expensive. The only reason the RKGL version was less efficient than the RK version in Table 9 was the need to use Phase 4. A better estimate of the global error coefficient in Phase 3 could have prevented this.
- The use of $\delta_{loc} = \sqrt{\delta}$ in Phase 1 is not necessarily optimal. A case in point is for **LHR** applied to P1 with $r = 2$ and $\delta = 10^{-6}$: Here, when $\delta_{loc} = 10^{-3}$, Phase 4 was required. However, with $\delta_{loc} = 10^{-4}$, Phase 4 was not required and the total arithmetical operations count was only 21957.
- The effect of the length of the interval of integration should be investigated. A larger interval will require more nodes, and it is clear from Tables 9−12 that when the number of nodes used in Phase 3 is much larger than that used in Phases 1 and 2, the efficiency will be due essentially to Phase 3 only (and Phase 4, if needed). For $r = 4$, the improvement in relative efficiency of RKGL might be more significant over larger intervals of integration.

7. References

Burden, R.L. & Faires, J.D. (2001). *Numerical Analysis (7th ed.),* Brooks/Cole, ISBN: 0-534-38216-9, Pacific Grove.

Butcher, J.C. (2000). Numerical methods for ordinary differential equations in the 20th century. *Journal of Computational and Applied Mathematics*, 125, (April 2000) 1−29, ISSN : 0377-0427.

Butcher, J.C. (2003). *Numerical Methods for Ordinary Differential Equations,* Wiley, ISBN: 0-471-96758-0, Chippenham.

Hairer, E.; Norsett, S.P. & Wanner, G. (2000). *Solving Ordinary Differential Equations I : Nonstiff Problems,* Springer-Verlag, ISBN: 3-540-56670-8, Berlin.

Hull T.E.; Enright, W.H.; Fellen, B.M. & Sedgwick, A.E. (1972). Comparing numerical methods for ordinary differential equations. *SIAM Journal on Numerical Analysis*, 9, 4, (December 1972) 603−637, ISSN: 0036-1429.

Kincaid, D. & Cheney, W. (2002). Numerical Analysis : Mathematics of Scientific Computing, Brooks/Cole, ISBN : 0-534-38905-8, Pacific Grove.

LeVeque, R.J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations,* SIAM, ISBN: 978-0-898716-29-0, Philadelphia.

Prentice, J.S.C. (2008). The RKGL method for the numerical solution of initial-value problems. *Journal of Computational and Applied Mathematics*, 213, (April 2008) 477−487, ISSN : 0377-0427.

Prentice, J.S.C. (2009). General error propagation in the RK*r*GL*m* method. *Journal of Computational and Applied Mathematics*, 228, (June 2009) 344−354, ISSN : 0377-0427.

Shampine, L. (1994). *Numerical Solution of Ordinary Differential Equations,* CRC Press, ISBN: 978-0-412051-51-7, Boca Raton.

Shampine, L. (2005). Error estimation and control for ODEs. *Journal of Scientific Computing*, 125, 1, (October 2005) 3−16, ISSN : 0885-7474.

Advanced Technologies

Edited by Kankesu Jayanthakumaran

ISBN 978-953-307-009-4

Hard cover, 698 pages

Publisher InTech

Published online 01, October, 2009

Published in print edition October, 2009

This book, edited by the Intech committee, combines several hotly debated topics in science, engineering, medicine, information technology, environment, economics and management, and provides a scholarly contribution to its further development. In view of the topical importance of, and the great emphasis placed by the emerging needs of the changing world, it was decided to have this special book publication comprise thirty six chapters which focus on multi-disciplinary and inter-disciplinary topics. The inter-disciplinary works were limited in their capacity so a more coherent and constructive alternative was needed. Our expectation is that this book will help fill this gap because it has crossed the disciplinary divide to incorporate contributions from scientists and other specialists. The Intech committee hopes that its book chapters, journal articles, and other activities will help increase knowledge across disciplines and around the world. To that end the committee invites readers to contribute ideas on how best this objective could be accomplished.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Justin S. C. Prentice (2009). Improving the Efficiency of Runge-Kutta Reintegration by Means of the RKGL Algorithm, Advanced Technologies, Kankesu Jayanthakumaran (Ed.), ISBN: 978-953-307-009-4, InTech, Available from: http://www.intechopen.com/books/advanced-technologies/improving-the-efficiency-of-runge-kutta-reintegration-by-means-of-the-rkgl-algorithm

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen