A Generalized Runge Kutta Method of Order three

Per G. Thomsen *

April 22, 2002

Abstract

We present a numerical method for the solution of stiff systems of ODE’s and index one DAE’s. The type of method is a 4 stage Generalized Linear Method that is reformulated in a special Semi Implicit Runge Kutta Method of SDIRK type. Error estimation is by imbedding a method of order 4 based on the same stages as the method and the coefficients are selected for ease of implementation. The method has 4 stages and the stage order is 2. For purposes of generating dense output and for initializing the iteration in the internal stages a continuous extension is derived. The method is A-stable and we present the region of absolute stability and the order star of the order 3 method that is used for delivering the solution.

1 Introduction

The inspiration for the present study came from the results by Prothero and Robinson (1974) [7] where they discovered the order reduction of implicit Runge Kutta methods, when applied to stiff systems of differential equations. This observation led to new concepts of stability and eventually to a better understanding of the importance of stage order in connection with the overall properties of one-step methods.

Experiences from work on SDIRK-methods [8] on a three stage method of second order with stage order one, and following discussions with John C. Butcher, the idea of designing a generalized Runge Kutta method with stage order higher than one became interesting and this resulted in the method reported in this paper.

2 Derivation of the Generalized Runge Kutta Method

In the classical reference on Runge-Kutta and General Linear methods [5] Butcher introduces the generalized Runge Kutta Scheme.

*Informatics and Mathematical Modelling, DTU, Lyngby, Denmark
\[ v_n = \tilde{A}u_n + h\tilde{B}f(v_n) \quad (1) \]
\[ u_{n+1} = Au_n + hBf(v_n) \quad (2) \]

The scheme is characterized by the choice of the coefficient matrices and by choosing these we can obtain methods with the properties we are looking for. In schematic form our method can be expressed by the tableau

\[
\begin{array}{c|c}
\tilde{A} & \tilde{B} \\
A & B \\
\end{array}
\]

Schematic or matrix form of a general linear method.

Among the possible methods we chose one where the last function value from the previous step is used and three stages where the last stage is at the right endpoint of our step. This type of method is referred to as First Same As Last or FSAL. The choice will lead to the following form.

\[
\begin{array}{ccc|ccc}
\gamma & 0 & 0 & 1 & a_{21} \\
a_{32} & \gamma & 0 & 1 & a_{31} \\
a_{42} & a_{43} & \gamma & 1 & a_{41} \\
\hline
0 & 0 & 1 & 1 & b_1 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

General linear form of the method.

It is now possible to express the same scheme as a semi-implicit Runge Kutta scheme with an explicit first stage. In this form the method can be written as follows.

\[ v_n = u_n + hAf(v_n) \quad (3) \]
\[ u_{n+1} = u_n + hbf(v_n) \quad (4) \]

In this form each of the stages is a system of nonlinear equations that are solved stage by stage. The Butcher scheme for this is the following,


\[ 
\begin{array}{c|ccc}
0 & 0 \\
c_2 & a_{21} & \gamma \\
c_3 & a_{31} & a_{32} & \gamma \\
1 & b_1 & b_2 & b_3 & \gamma \\
\hline
y_{n+1} & b_1 & b_2 & b_3 & \gamma \\
e_{n+1} & d_1 & d_2 & d_3 & d_4 \\
\end{array}
\]

The last line in the scheme contains the coefficients of the estimator for the local error obtained from imbedding an order 4 method in the scheme and computing the difference between the two solutions to obtain an estimate of the error for the method of order three.

2.1 \textbf{Order conditions, stage order.}

We want to satisfy the conditions for order 3 and stage order 2. The order conditions for the semi-implicit stages become:

\text{Stage 2:}

\[ c_2 = a_{21} + \gamma = 2\gamma \implies a_{21} = \gamma; \quad (5) \]

\text{Stage 3:}

\[ a_{32}c_2 + \gamma c_2 = \frac{c_3^2}{2}; \quad (6) \]

\text{Stage 4:}

\[ b_2c_2 + b_3c_3 = \frac{1}{2} - \gamma; \quad b_2c_2^2 + b_3c_3^2 = \frac{1}{3} - \gamma; \quad (7) \]

2.2 \textbf{Order conditions, method order.}

The order condition for the imbedded error estimator, asking for the solution based on all the stages to be a fourth order solution leads to the equation.

\[ d_2(\frac{c_3^2}{3} - \gamma c_2^2) + d_3(\frac{c_3^2}{3} - a_{32}c_2^2 - \gamma c_2^2) = 0 \quad (8) \]
This condition may be expressed by means of the errors from stages 2 and 3.

\[ d_2e_2 + d_3e_3 = 0 \quad (9) \]

where \( e_2 \) is the error from stage 2 and \( e_3 \) is the error from stage 3.

\[ e_2 = \left( \frac{c_2^3}{3} - \gamma c_2^2 \right) = \frac{c_2^3}{3} - \gamma (c_2^3 - c_2^2). \quad (10) \]

\[ e_3 = \left( \frac{c_3^3}{3} - a_{32}c_2^2 - \gamma c_2^2 \right) = -\frac{4}{3} \gamma^3. \quad (11) \]

### 3 Determination of the value \( \gamma \)

In order to define the method we must find the value of \( \gamma \). This may be done by specifying the property that our method must be L-stable. This requires [2] that the following condition is satisfied:

\[ [c_3(1 - 6\gamma + 12\gamma^2) - 2\gamma(2 - 9\gamma + 12\gamma^2)] (c_3 - 2\gamma) = 0 \quad (12) \]

The choice \( c_3 = 2\gamma = c_2 \) is not desirable and we are left with the other condition

\[ c_3(1 - 6\gamma + 12\gamma^2) - 2\gamma(2 - 9\gamma + 12\gamma^2) = 0 \quad (13) \]

From which we find \( \gamma = \frac{5}{12} \) as a reasonable choice giving \( c_3 = \frac{10}{21} \).

We now use the conditions (4) to (8) and the method is defined and may present the Butcher tableau:

\[
\begin{array}{ccc|cc}
0 & 0 & & & \\
\frac{5}{6} & \frac{5}{12} & \frac{5}{12} & & \\
\frac{10}{21} & \frac{95}{198} & \frac{49}{78} & \frac{5}{12} & \\
1 & \frac{59}{600} & \frac{31}{75} & \frac{539}{600} & \frac{5}{12} & \\
\frac{y_{n+1}}{2} & \frac{59}{600} & \frac{31}{75} & \frac{539}{600} & \frac{5}{12} & \\
\frac{e_{n+1}}{2} & \frac{55}{600} & \frac{55}{75} & \frac{-245}{600} & -\frac{5}{12} & \\
\end{array}
\]

Coefficients for the 4-stage GERK - method of order 3.
4 Stability Properties

In order to obtain suitable properties for the solution of index 1 DAE’s we must at least satisfy the conditions for A-stability. This can be found from considering the test equation and the resulting rational approximation to the exponential function. A straightforward calculation leads to the rational function.

\[ R(z) = \frac{P(z)}{Q(z)} \]  

The rational fraction that has been obtained by the GERK-method is given by

\[ R(z) = \frac{1 - \frac{1}{4}z + \frac{11}{16}z^2 - \frac{17}{128}z^3}{(1 - \frac{1}{12}z)^3} \]  

This rational function is a third order approximation to \( \exp(z) \) and the stability properties of the GERK method is closely related to the acceptability of this approximation.

4.1 Stability region

The region of absolute stability is shown in the figure below.

![Stability region](image)

**Figure 1:** Stability region for Gerk(4)

4.2 Order Star

Following the analysis of [4] we apply the theory of order stars to verify the A-stability property of the GERK-scheme. The verification is equivalent to observing that no branch of the order star crosses the imaginary axis. For the
purpose of presenting the overall picture of the properties of the rational approximation we present the order star of type 1 in the figure below.

By inspecting the order star we see that no “finger” crosses the imaginary axis

![Figure 2: Order star for Gear(4)](image)

and the method is indeed A-stable.

5 Continuous Extension

In the GERK-method every stage except the first involves the solution of a system of nonlinear equations. For this system some kind of iterative solver is applied and in order to get the iterations started we need a starting guess. For that purpose and also to generate dense output of the solution we supply a continuous extension or interpolation formula of the form

\[ u(t_{n} + \theta h) = u_{n} + h \sum_{s=1}^{r} d_{s}(\theta) f(v_{s}) \]  \hspace{1cm} (16)

The derivation of this formula for the interpolation follows the same theory as was developed in [13] where the methods were all explicit. The basic ideas however are very similar and have been used in [11], here we give the resulting coefficient matrix.

\[ d_{s}(\theta) = d_{s,1} \theta + d_{s,2} \theta^{2} + d_{s,3} \theta^{3} \]  \hspace{1cm} (17)

here the coefficients are given in the form of three vectors.
\[ d_1 \begin{bmatrix} 20 & -1020 & 1451/24 \\ 211 & 649 & -337/24 \\ 216 & -3888 & 216/24 \end{bmatrix} \]

Polynomial coefficients \( d_s(\theta) \) for the continuous extension.

6 Implementation

For testing the GERK method we have developed an Object-Oriented C++ program package SDIRK [10] that is intended for solution of stiff ODE’s. In this section we describe how to use this implementation and some of the strategies that have been used for stepsize and convergence control. The basic ideas are similar to those in Gustavsson [6] and have been developed in the GODESS-project [3]. This implementation is still a research code mainly for use in research and has proven helpful also in teaching. The basic ideas in the Object Oriented software development that are applied has been helpful to illustrate the basic ideas in ODE-solvers.

A third implementation in Matlab has been used for comparing control strategies. This code includes facilities for switching between different methods and stepsize control strategies and is made available to be copied from my homepage, www.imm.dtu.dk/~pgt/GERK. The code is mainly intended for testing but may be used for the solution of stiff ODE’s in general since it is indeed very efficient and flexible. A documentation is found at the same website.

6.1 PI-control of stepsize.

In order to optimize the stepsize-control strategy a Matlab implementation of the method has been implemented under the name gerk.m. As a special feature the gerk code has the possibility to choose from a number of stepsize-control strategies. These have all been forged in the same template that is derived from the PI-controllers that have been developed by Soderlind [9]. The basic formula for estimating the stepsize for the current step from data gathered in previous steps is the following.

\[ h_{n+1} = h_n \left( \frac{\tau}{e_n} \right)^{\beta_1} \left( \frac{\tau}{e_{n-1}} \right)^{\beta_2} \left( \frac{h_n}{h_{n-1}} \right)^{-\alpha_2} \]  \hspace{1cm} (18)

For the purpose of relating this control to the traditional step-control strategy and others from the literature we give a couple of examples on typical choices of parameters.

- ordinary step control
  \[ \alpha_2 = 0, \beta_1 = 1/3, \beta_2 = 0. \]


- Watts step control
  \[\alpha_2 = 0, \beta_1 = 1/3, \beta_2 = 1/3.\]

- Choice used by Gustavsson
  \[\alpha_2 = 1, \beta_1 = 0.3/3, \beta_2 = 0.4/3.\]

- Second order PI-control
  \[\alpha_2 = 1/2, \beta_1 = 1/6, \beta_2 = 1/6.\]

For further discussion on the properties of the PI-control strategies we refer to [9]. Results from testing the four strategies is found in the example-section below.

6.2 The structure of the SDIRK code

A general ODE-solver has been implemented in C++, named SDIRK. The Object Oriented implementation makes use of the high degree of structuring that is offered by the C++ environment.

For details we refer the reader to the users manual of SDIRK [10]. The structure of the program and the connectivity is shown in the figures below.

6.3 The implementation in GODESS.

Until recently, the testing of ODE/DAE solvers has been limited to comparing software. The complex process of developing software from a mathematically specified method entails constructing control structures and objectives, selecting termination criteria for iterative methods, choosing norms and many more decisions. Most software constructors have taken a heuristic approach to these design choices, and as a consequence two different implementations of the same method may show significant differences in performance. Yet it is common to try to deduce from software comparisons that one method is better than another. Such conclusions are not warranted, however, unless the testing is carried out under true ceteris paribus conditions. Moreover, testing is an empirical science and as such requires a formal test protocol; without it conclusions are questionable, invalid or even false. We argue that ODE/DAE software can be constructed and analyzed by proven, "standard" scientific techniques instead of heuristics, and that each solver should have a complete specification of its algorithmic content. Further, we indicate that a test protocol can be devised such that firm conclusions may be drawn from careful testing. The goal is reproducibility as well as improved software quality.

6.3.1 Results from testing in GODESS

The implementation that uses Krylov subspace techniques for the solution of the linear subproblems that arise is developed with the intention to obtain knowledge about the performance of different iterative algorithms and different types of preconditioning. Godess proved to be an efficient platform for making such
ODE/DAE METHODS, ALGORITHMS AND SOFTWARE

comparisons. Details can be found in [12] here we bring a comparison of the
GEARK-method and seven other methods for stiff systems.
The testproblem is a 2D model of the production of Ozone in the stratosphere
[1]. The model consists of two coupled PDE’s

\[
\begin{align*}
\frac{\partial c^i}{\partial t} &= K_h \frac{\partial^2 c^i}{\partial x^2} + \frac{\partial}{\partial z} \left( K_v(z) \frac{\partial c^i}{\partial z} \right) \\
&\quad + V \frac{\partial c^i}{\partial x} + R^i(c^1, c^2, t) \quad (i = 1, 2) \\
K_h &= 4 \times 10^{-6}, \quad K_v(z) = 10^{-8} e^{z/5}, \quad V = 0.01 \\
R^1(c^1, c^2, t) &= -k_1 c^1 - k_2 c^1 c^2 + k_3(t) * 7.4 \times 10^{16} + k_4(t) c^2 \\
R^2(c^1, c^2, t) &= k_1 c^2 - K_2 c^1 c^2 - k_4(t) c^2 \\
k_1 &= 6.031, \quad k_2 = 4.66 \times 10^{-16} \\
k_3 &= \begin{cases} 
\exp(-22.62/\sin(\pi t/43200)) & , \quad t < 43200 \\
0 & , \quad \text{otherwise}
\end{cases} \\
k_4 &= \begin{cases} 
\exp(-7.601/\sin(\pi t/43200)) & , \quad t < 43200 \\
0 & , \quad \text{otherwise}
\end{cases}
\end{align*}
\]

Here the concentrations of oxygen $c^1$ and Ozone $c^2$ are the variables and the
equations represent reaction-transport with horizontal diffusion and advection.
The boundaries are $0 \leq x \leq 20, \ 30 \leq z \leq 50, \ 0 \leq t \leq 86400$. The Jacobian
has a banded structure reflecting the discretization used for the derivatives on
a uniform rectangular grid. The system is stiff and the spectrum of the Jacobian
matrix can be found in [1]. The absolute tolerance is $10^{-3}$ and the relative
tolerance is $10^{-5}$. For the Krylov method we have selected an Arnoldi type type
of INcomplete Orthogonalization in order to compare with the results from [1].
The preconditioning is a simple diagonal preconditioning for this test. Other
preconditionings are available in the GODESS implementation. From the results

<table>
<thead>
<tr>
<th>Method</th>
<th>Steps</th>
<th>GKD</th>
<th>GNI</th>
<th>AMV</th>
<th>ALA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESDIRK23a</td>
<td>1052</td>
<td>2.56</td>
<td>2.04</td>
<td>16505</td>
<td>55.7</td>
</tr>
<tr>
<td>ESDIRK23b</td>
<td>865</td>
<td>2.75</td>
<td>2.10</td>
<td>15015</td>
<td>54.5</td>
</tr>
<tr>
<td>ESDIRK45a</td>
<td>257</td>
<td>3.62</td>
<td>2.53</td>
<td>11731</td>
<td>57.0</td>
</tr>
<tr>
<td>ESDIRK45b</td>
<td>239</td>
<td>3.62</td>
<td>2.74</td>
<td>11800</td>
<td>57.7</td>
</tr>
<tr>
<td>Hairmann</td>
<td>1430</td>
<td>2.49</td>
<td>2.77</td>
<td>34500</td>
<td>123.2</td>
</tr>
<tr>
<td>sd34var</td>
<td>414</td>
<td>3.02</td>
<td>2.66</td>
<td>14800</td>
<td>57.7</td>
</tr>
<tr>
<td>GERRK</td>
<td>515</td>
<td>3.16</td>
<td>2.55</td>
<td>12466</td>
<td>51.1</td>
</tr>
<tr>
<td>BDF</td>
<td>610</td>
<td>2.61</td>
<td>2.18</td>
<td>3476</td>
<td>10.5</td>
</tr>
</tbody>
</table>

Table 1: Comparison of different methods and the dimension of the Krylov subspace
used in the iterations. Number of time steps (Steps), Average Dimension of subspace
(GKD), Average Number of Iterations (GNI), Number of Matrix-vector Operations
(AMV), Other Linear algebra operations (ALA) in millions.
we see, that the GERK method is the most efficient of the one-step methods while the BDF-method is the overall winner since it does less work per step although it takes more steps than GERK. The general impression is that the dimension of the subspace is indeed very small compared to the size of the system on a 10x10 grid this is 100 and the dimension of the Krylov subspace is near 3. This shows that in general for this type of systems the Krylov type of linear solver is very efficient.

6.4 Using the SDIRK - solver

We will illustrate the general usage of the SDIRK package by applying the code on the Van Der Pol equation. Two tests have been carried out in order to check the performance for different types of problems.

6.4.1 Example: the Van Der Pol equation

The Van Der Pol problem is a well known test example that may be used to illustrate the performance of a piece of code for solving stiff and non-stiff ODE’s. The Van Der Pol system describes a model of an electronic oscillator with a nonlinear element and has one parameter $\mu$ that may be varied or even made dependent on the integration time.

$$y'' - \mu(1 - y^2)y' + y = 0$$

This is a second order ODE but may be transformed into two coupled first order ODE’s which then becomes partly stiff when the parameter $\mu$ is large.

In the first graph it is shown how the total amount of work depends on the error tolerances that are prescribed. Tolerances are varied between 0.1 and $10^{-10}$ the results show that the work is roughly proportional to the required accuracy. The parameter $\mu = 200$ has been chosen to make the problem moderately stiff.

In the second test the parameter $\mu$ in the Van Der Pol equation is varied over the values $0 < \mu < 10000$ the work is measured as the number of steps needed for the solution over a fixed interval $[0, 1000]$. This will in all cases include the initial transient. The relative error tolerance is $10^{-6}$ while the absolute tolerance is $10^{-9}$ this is a quite severe test ranging over non-stiff to very stiff.

We see that the number of steps varies a lot with the $\mu$-value and since the stiffness of the problem is roughly equivalent to the size of $\mu$ it is obvious that the method is coping very well with stiffness but at the same time will be relatively expensive for non-stiff problems. This is due to the fact that the order of the method is relatively low.

A third test is performed for the Van Der Pol equation with $\mu = \mu_0 exp(t - t_0)$ This gives a severe test for the solver since the problem gradually becomes more and more stiff and harder to solve when time increases. The solution may be pictured in the Phase-plane and is shown below as it was generated by the solver. The value of $\mu$ where the solver gave up was close to 35000 (this was enforced when the step size became smaller than $10^{-8}$).
Figure 3: Steps versus tolerance test, log-log plot.

Figure 4: Van Der Pol solutions for variable $\mu$. 
6.4.2 Testing the GERK - solver

Using the above test example we can illustrate the performance of different choices of stepsize strategies. The table shows the number of accepted steps (steps), the number of failed steps (FSTP) and the percentage of failed steps (PFSTP) for the four different control strategies and for two different values of the relative error tolerance.

We see from these results that the number of failed steps are very important for the performance of the strategies and the asymptotic is best in both the stiff and the non-stiff case when the tolerance is moderate (reps= 10^{-4}) while the second order control is better for tighter tolerance (reps = 10^{-6}). However an inspection of the stepsize sequence gives a different picture altogether since the Gustavsson and second order strategies lead to a much smoother stepsize sequence than the asymptotic and this in most cases is what is wanted to make the stepsize control robust.

<table>
<thead>
<tr>
<th>Method</th>
<th>Steps</th>
<th>FSTP</th>
<th>PFSTP</th>
<th>Steps</th>
<th>FSTP</th>
<th>PFSTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu = 20 ), nonstiff</td>
<td>( 10^{-4} )</td>
<td></td>
<td></td>
<td>( 10^{-6} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>asymptotic</td>
<td>880</td>
<td>46</td>
<td>14.23</td>
<td>2634</td>
<td>267</td>
<td>9.20</td>
</tr>
<tr>
<td>Watts</td>
<td>1077</td>
<td>280</td>
<td>20.63</td>
<td>3525</td>
<td>936</td>
<td>20.98</td>
</tr>
<tr>
<td>Gustavsson</td>
<td>1052</td>
<td>221</td>
<td>17.56</td>
<td>2985</td>
<td>372</td>
<td>11.08</td>
</tr>
<tr>
<td>Second order</td>
<td>919</td>
<td>167</td>
<td>15.38</td>
<td>2752</td>
<td>413</td>
<td>13.05</td>
</tr>
<tr>
<td>( \mu = 200 ), stiff</td>
<td>( 10^{-4} )</td>
<td></td>
<td></td>
<td>( 10^{-6} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>asymptotic</td>
<td>1615</td>
<td>286</td>
<td>15.04</td>
<td>4982</td>
<td>673</td>
<td>11.91</td>
</tr>
<tr>
<td>Watts</td>
<td>2011</td>
<td>535</td>
<td>21.01</td>
<td>6646</td>
<td>1877</td>
<td>22.02</td>
</tr>
<tr>
<td>Gustavsson</td>
<td>1818</td>
<td>397</td>
<td>17.92</td>
<td>5125</td>
<td>716</td>
<td>12.26</td>
</tr>
<tr>
<td>Second order</td>
<td>1626</td>
<td>336</td>
<td>17.13</td>
<td>4779</td>
<td>534</td>
<td>10.05</td>
</tr>
</tbody>
</table>

Table 2: Comparison of stepsize-strategies on the VanDerPol equation

When integrating large systems of ODEs or DAEs where Jacobian matrices are expensive to calculate and decompose the reduction in the number of needed Jacobian matrices will translate into a reduction of the CPU-time needed to perform the integration. This effect is not tested in the example from table 2, but included in the data in figure 3.

7 Conclusion

The present work introduces a Generalised Runge Kutta solver for ODE’s and DAE’s of third order with four stages and stage order two. The method has been implemented and tested in two different implementations. The GODESS package is a test-environment for ODE-solvers and the properties of the GERK-method are demonstrated to be very promising especially for stiff systems. Qualitatively the tolerance proportionality in the implementation turns out to be
very good.
The second implementation SDIRK is a general purpose ODE-solver where different stepsize strategies are available and the tests have shown that PI-control strategies are marginally better that the traditional stepsize control based purely on the asymptotic error behaviour. This is demonstrated on several tests on the VanDerPol equation.
Some comparisons using a Matlab implementation have led to the conclusion that a second order PI-control is a very good choice for a general ODE-solver that may be used for stiff as well as nonstiff problems for reasonably strict error tolerances.
The use of the GERK method in connection with retarded ODE’s is presented in [11]. The use of the GERK method for general DAE’s is referred to a later paper.

References
