Numerical results for a parallel linearly–implicit Runge–Kutta method

Jürgen Bruder

Abstract

For the parallelisation of implicit Runge–Kutta methods for stiff ODE's a parallel computation of the stages is obvious. In this paper we consider the parallelisation of the stages of linearly-implicit Runge-Kutta methods. The construction and implementation of a parallel linearly-implicit Runge-Kutta method is described. The numerical results are compared with the code PSODE of van der Houwen/Sommeijer [6] and a straightforward parallelisation of RADAU5 [5]. All methods are based on the 3-stage implicit Radau IIA method.

Keywords: *linearly-implicit Runge-Kutta methods, parallel implementation*

1 Introduction

A widely used class of methods for the numerical solution of the stiff initial value problem

$$y' = f(x, y), \quad y(x_0) = y_0 \tag{1}$$

with $x \in \mathrm{R}$, $y \in \mathrm{R}^n$, $f : \mathrm{R} \times \mathrm{R}^n \to \mathrm{R}^n$ are implicit Runge–Kutta methods. These methods require the solution of nonlinear equations, in general by modified Newton methods. For a parallel implementation it is obvious to compute each stage of the method on a different processor and to assemble the results finally. Because the stages of the implicit Runge–Kutta method are coupled with each other by a system of nonlinear resp. linear equations, a direct approach is using several processors to solve this systems of linear equations. In this paper, however, we want to exclude a parallelisation of the systems of linear equations. We only consider a parallelisation across the ODE-solver. A survey on parallel methods for ODE's is given in [3]. The standard approach is to construct Runge–Kutta methods with one single eigenvalue or multi implicit Runge–Kutta methods with only real eigenvalues of the Runge-Kutta matrix [7] and to decouple the stages by a transformation to a diagonal matrix.

In the second section of this paper two methods using several processors for the stages of implicit Runge–Kutta methods are summarized. We first describe linearly–implicit Runge–Kutta methods with functional iteration where the stages are decoupled a priori and thus can be computed parallel. In the second approach of van der Houwen/Sommeijer [6] a diagonal iteration matrix is used to decouple the stages of a given Runge-Kutta method. For comparison we use a modified version of the code RADAU5 of [5]. This code is a sequential code, but because the Runge–Kutta matrix is transformed to a block–diagonal matrix two processors can be used parallel. Details of the implementation of the methods and numerical results are given in the third section.

2 Parallel Methods

Both approaches we consider for the parallel computation of stages are based on implicit Runge–Kutta methods. Thus we first want to give our notations of these methods. For simplicity of notation we assume autonomous problems.

For the usual Butcher tableau

$$\begin{array}{c|c} \bar{c} & \bar{A} \\ \hline 1 & \bar{b}^T \end{array}$$

with $\bar{A} \in \mathrm{R}^{s\times s},\ \bar{b} \in \mathrm{R}^s,\ \bar{c} = \bar{A}\bar{e},\ \bar{c} \in \mathrm{R}^s,\ \bar{e} = (1,\ldots,1)^T \in \mathrm{R}^s$ we define

$$A = \begin{pmatrix} \bar{A} & \bar{0} \\ \bar{b}^T & 0 \end{pmatrix}, \qquad c = \begin{pmatrix} \bar{c} \\ 1 \end{pmatrix}.$$

Now the parameters of an implicit Runge–Kutta method are given by

$$\begin{array}{c|c} c & A \end{array}$$

with $A \in \mathrm{R}^{(s+1)\times(s+1)},\ c \in \mathrm{R}^{s+1}, c = Ae,\ e = (1,\ldots,1)^T \in \mathrm{R}^{s+1}$ and the method itself is given in vector notation by

$$U_{m+1} = e \otimes u_m + h\ (A \otimes I_n)\ F(U_{m+1}) \tag{2}$$

with vectors $F(U_{m+1}) = (f(x_0 + c_1 h, u_{m+1}^{(1)})^T, \ldots, f(x_0 + c_{s+1}h, u_{m+1}^{(s+1)})^T)^T$ and $U_{m+1} = (u_{m+1}^{(1)}{}^T, \ldots, u_{m+1}^{(s+1)T})^T$ of length n(s+1).

If we solve (2) by a Newton-like iteration we get

$$\begin{aligned} U_{m+1}^{(k+1)} &= (I_{s+1} \otimes I_n - h\ A \otimes T_N)^{-1}(e \otimes u_m) \\ &\quad + h(I_{s+1} \otimes I_n - h\ A \otimes T_N)^{-1}(A \otimes I_n)\left(F(U_{m+1}^{(k)}) - (I_{s+1} \otimes T_N)U_{m+1}^{(k)}\right) \end{aligned} \tag{3}$$

where T_N approximates the Jacobian f_y of the right hand side f in (1). We assume $h\|A \otimes T_N\| < 1$ and insert $(I_{s+1} \otimes I_n - hA \otimes T_N)^{-1} = \sum_{k=0}^{\infty} h^k (A \otimes T_N)^k$ in (3) to get finally the expression

$$
\begin{aligned}
U_{m+1}^{(k+1)} &= \sum_{k=0}^{\infty} h^k (A \otimes T_N)^k (e \otimes u_m) \\
&\quad + \sum_{k=0}^{\infty} h^{k+1} A^{k+1} \otimes T_N^k \left(F(U_{m+1}^{(k)}) - (I_{s+1} \otimes T_N) U_{m+1}^{(k)} \right). \qquad (4)
\end{aligned}
$$

2.1 Linearly–implicit Runge–Kutta methods

Linearly–implicit Runge–Kutta methods (LIRKM) are usually given by

$$
\begin{aligned}
u_{m+1}^{(1)} &= u_m \\
u_{m+1}^{(i)} &= R_0^{(i)}(c_i h T_S) u_m + h \sum_{j=1}^{i-1} A_{ij}(hT_S)(f(u_{m+1}^{(j)}) - T_S u_{m+1}^{(j)}), \qquad (5) \\
&\qquad\qquad i = 2, \ldots, s+1 \\
u_{m+1} &= u_{m+1}^{(s+1)}
\end{aligned}
$$

where $R_0^{(i)}(z)$ are rational functions that approximate the exponential function. Especially we consider functions of the form

$$
R_0^{(i)}(z) = \frac{b_0 + b_1 z + \ldots + b_{q_i} z^{q_i}}{(1 - \gamma_i z)^{q_i}}.
$$

$A_{ij}(z)$ are rational functions with the same denominator as $R_0^{(i)}(z)$ and with $A_{ij}(\infty) = 0$, whose coefficients are determined from consistency conditions. The matrix T_S is arbitrary in general, the order of consistency is independent of T_S. For stiff problems T_S has to be an approximation to the Jacobian f_y of the right hand side f in (1). For further details we refer to [8]. With the definitions

$$
\begin{aligned}
\mathbf{R}(hT_S) &= (R_0^{(1)}(c_1 h T_S), \ldots, R_0^{(s+1)}(c_{s+1} h T_S))^T \in \mathrm{R}^{n(s+1) \times n} \\
\mathbf{A}(hT_S) &= (A_{ij}(c_i h T_S))_{i,j=1,\ldots,s+1} \in \mathrm{R}^{n(s+1) \times n(s+1)}
\end{aligned}
$$

we get the vector notation

$$
U_{m+1} = \mathbf{R}(hT_S) u_m + h \mathbf{A}(hT_S) \left(F(U_{m+1}) - (I_{s+1} \otimes T_S) U_{m+1} \right) \qquad (6)
$$

which is summarized in the tableau

$$
c \;\big|\; \mathbf{A}(hT_S). \qquad (7)
$$

For $T_S = 0$ the method is reduced to a classical Runge–Kutta method. If this underlying classical method is explicit the stage vectors $u_{m+1}^{(i)}$ can be computed subsequently. Because the matrix functions are rational functions one has to solve q_i systems of linear equations in the i–th stage. For a parallel execution of the stages implicit underlying methods (with an index i in (5) running up to $s+1$) are more interesting. In this case we have to compute the stage vectors $u_{m+1}^{(i)}$ from a nonlinear system of equations by a modified Newton method.

If this Newton method uses a matrix T_N to approximate the Jacobian f_y of the right hand side f in (1) we get

$$\begin{aligned}(I_{s+1}\otimes I_n - h\mathbf{A}(hT_S)\,(I_{s+1}\otimes T_N - I_{s+1}\otimes T_S))\;(U_{m+1}^{(k+1)} - U_{m+1}^{(k)}) = \\ -U_{m+1}^{(k)} + \mathbf{R}(hT_S)u_m + h\mathbf{A}(hT_S)\left(F(U_{m+1}^{(k)}) - (I_{s+1}\otimes T_S)U_{m+1}^{(k)}\right).\end{aligned} \tag{8}$$

This formula essentially simplifies, however, if we use the same matrix $T = T_S = T_N$ to approximate f_y for the stiff ODE-solver and for the modified Newton method. We get

$$U_{m+1}^{(k+1)} = \mathbf{R}(hT)u_m + h\mathbf{A}(hT)\left(F(U_{m+1}^{(k)}) - (I_{s+1}\otimes T)U_{m+1}^{(k)}\right). \tag{9}$$

We see that the same formulas are valid for the iterative solution of (6) with functional iteration or with modified Newton iteration, if the approximations to the Jacobian in the Newton-method and in the linearly-implicit method are the same. In (9) the stages are decoupled and can be executed parallel (one LU–decomposition and q_i back substitutions).

To get an impression of the relation to implicit Runge-Kutta methods, we define coefficients $w_{i,k}$ and $a_{i,j}^{(k)}$ by the following Taylor-series:

$$\begin{aligned} R_0^{(i)}(z) &= \sum_{k=0}^{\infty} w_{i,k} z^k && (10)\\ A_{ij}(z) &= \sum_{k=0}^{\infty} \frac{1}{(k+1)!} a_{ij}^{(k+1)} \left(\frac{z}{c_i}\right)^k . \end{aligned}$$

For matrix notation we summarize the coefficients $a_{ij}^{(k)}$ in a matrix $A^{(k)}$ and get

$$\mathbf{A}(hT) = \sum_{k=0}^{\infty} \frac{1}{(k+1)!} h^k (A^{(k+1)} \otimes T^k). \tag{11}$$

If we now compare (4) with (9), (10) and (11) we see that an implicit Runge-Kutta method and a LIRKM based on the same implicit Runge-Kutta method would be identical, if

$$A^{(k)} = k!\; A^k, \tag{12}$$

$$w_{i,k} c_i^k k! = \sum_{j=1}^{s+1} a_{ij}^{(k)}, \tag{13}$$

$k = 1, \ldots, \infty$ would hold for the matrix functions of the LIRKM. This is true for each iteration as well as for the limit $U_{m+1} = U_{m+1}^{(k+1)} = U_{m+1}^{(k)}$ of the iteration. Because only q_i coefficients are available in $R_0^{(i)}(z)$ and $A_{ij}(z)$, the conditions (12) and (13) can only be fulfilled up to $k = q_i$. But we see that a LIRKM with coefficients $a_{ij}^{(k)}, k = 1, \ldots, q_i$ due to (12) is 'similar' to implicit Runge-Kutta methods. More details and consistency considerations can be found in [2].

2.2 PSODE

For stiff systems van der Houwen and Sommeijer [6] start from the implicit Runge–Kutta method (2) and introduce an arbitrary matrix D to get

$$U_{m+1} - h\ (D \otimes I_n)\ F(U_{m+1}) = e \otimes u_m + h\ ((A-D) \otimes I_n) F(U_{m+1}).$$

The solution of this problem is determined iteratively by an appropriate predictor and the (outer) iteration

$$U_{m+1}^{(k+1)} - h\ (D \otimes I_n)\ F(U_{m+1}^{(k+1)}) = e \otimes u_m + h\ ((A-D) \otimes I_n) F(U_{m+1}^{(k)}),$$

which is performed until convergence. Each iteration step requires the solution of a nonlinear system of equations, usually by a modified Newton method. With the Jacobian f_y of the right hand side f in (1) and $F_U = diag(f_y, \ldots, f_y)$ we get the (inner) iteration

$$\begin{aligned}(I - h\ (D \otimes I_n)\ F_U) U_{m+1}^{(k+1),new} &= e \otimes u_m \\ &+ h\ (D \otimes I_n) \left(F(U_{m+1}^{(k+1),old}) - F_U\ U_{m+1}^{(k+1),old} \right) \\ &+ h\ ((A-D) \otimes I_n)\ F(U_{m+1}^{(k)}).\end{aligned}$$

The convergence of the outer iteration is studied in [6] by the test equation $y' = \lambda y$. With $z = h\lambda$ it holds

$$U_{m+1} - U_{m+1}^{(k+1)} = C(z)(U_{m+1} - U_{m+1}^{(k)})$$

with

$$C(z) = (I - zD)^{-1} z(A - D).$$

We see:

- For $D = A$ we get an usual implicit Runge–Kutta method which can not be executed on several processors without parallel linear equations solvers because the stages are coupled. But for this choice good convergence properties are expected ($C(z) \equiv 0$), because it corresponds to the solution of (2) by a modified Newton method.
- For $D = 0$ we get functional iteration which is not suitable for stiff problems.
- Finally $D = diag(d_1, \ldots, d_s, 0)$ is considered in [6] to decouple the stages for the inner iteration. Thus we can compute each stage parallel by a different processor (one system of linear equations) but we have to expect worse convergence than for the case $D = A$.

For further details we refer to [6].

3 Implementations and Numerical results

3.1 The Implementation of the methods

All methods used for the tests are based on the implicit 3–stage Radau IIA method of order 5. The numerical tests were performed on a Parsytec–computer with distributed memory. Because of the three stages we use three processors. For the computation of the Jacobian each processsor computes one of three stripes of the Jacobian, followed by an interchange of these parts to have the entire Jacobian available on each processor. The function values after each iteration are computed parallel at all three processors also, followed by an interchange of the function values. And of course the computation of the stage vectors $u_{m+1}^{(i)}$, especially the LU–decompositions and the back substitutions are done parallel.

PSODE

The original code PSODE was developed for a *shared* memory machine and had to be adapted to the handling of the parallel processors of the *distributed* memory of the Parsytec–computer. The explicit information interchange that had to be introduced may change the behaviour of the code compared to an shared memory machine. The inner iteration is performed only ones. The outer iteration is performed at least three times. The last two iterations are used to estimate the iteration error.

To make all three codes comparable the computation of the Jacobians has been changed. We do not use the exact Jacobian like the original code but a finite difference approximation like the other codes. Furthermore we did not check for an overflow of the solution because the solution is not known in general.

The three LU–decompositions of the matrices $I - d_i f_y,\ i = 1, 2, 3$, are computed parallel. Thus the numerical expense is only one LU–decomposition per integration step and one back substitution per iteration step. However, the convergence for a diagonal matrix D is in general less efficient than it would be for the full Runge–Kutta matrix A. We can expect more iterations than RADAU5 but a smaller expense for each iteration.

RADAU5

The implementation of the implicit Runge–Kutta method is based on the code RADAU5 of [5]. This code was *not* developed for parallel environments. The numerical expense for the system of linear equations is reduced by transforming the coefficients matrix $(I - h\ (A \otimes I_n)\ f_y)$ to a block diagonal matrix with one real $n \times n$ block for the real eigenvalue of A and one complex $n \times n$ block resp. one real $2n \times 2n$ block for the two complex eigenvalues of A. The expense for this transformation is of order n only. These two systems of linear equations are independent of each other and a parallel computation seems to be useful. A further splitting of the linear equations systems to more than two processors was not done because we did not want to parallelise the systems of linear equations. The handling of the parallel processors of the Parsytec–computer had to be inserted. The numerical expense is one complex LU–decomposition per integration step and one complex back substitution per iteration. We expect good convergence of the iteration because a modified Newton method is used but the iterations are more expensive than PSODE because of the complex linear equations systems.

The code only requires one iteration to estimate the iteration error. The truncation error for the stepsize control is estimated from the internal approximations $u_{m+1}^{(i)}$ by using special properties of Radau IIA methods.

Experiments in [5] showed that the original sequential RADAU5–code is efficient and very accurate.

Linearly–implicit Runge–Kutta method

The linearly–implicit Runge–Kutta method is based as the methods above on the 3–stage Radau IIA method of order 5. To get the classical order of consistency 5 for the linearly–implicit method the rational functions have to be approximations to the exponential function of order at least 5 in the last stage. We choose L–acceptable rational functions with only one multiple zero in the numerator. Then the polynomial in the numerator has to be at least of order 5, which requires 5 back substitutions per iteration. Because the computations of the three stages are done parallel the usual savings in the number of back substitutions in the internal stages have no effect. Thus we use the L-acceptable rational functions of order 5 in all stages. The LU–decompositions are done parallel. Accordingly the numerical expense is one LU–decomposition per integration step and 5 back substitutions per iteration. We need more vector operations than the other methods. Because (9) is a modified Newton method we expect a good convergence.

Compared to PSODE we have a more expensive iteration, but a better convergence (less number of iterations), compared to RADAU5 we have a comparable convergence, but the expensive iteration can be compensated only for large problems by the less expensive LU-decomposition.

The stopping criterion for the iteration and the stepsize control we used from RADAU5, because PSODE requires at least two iterations which may be too much for our expensive iteration. Numerical tests with a sequential implementation of a linearly–implicit methods in [1] showed that these methods are efficient for large and medium tolerances and slowly varying Jacobian.

3.2 Numerical Results

Examples

For the numerical tests we first used the problems F5 of Gottwald and Wanner [4] (8 equations, with the exception of the end of the integration intervall identical to the HIRES-problem in [5]), an example of Watkins/Hansonsmith [10] with 20 equations and 4 stiff components and an reactor equation of Strehmel/Weiner [9], S. 216 (8 equations). For our purpose, however, we did not consider the stiffness, partitioning or other properties of the problems.

Tests were performed with error tolerances of 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, and 10^{-8}. The Figures give the computing time versus the error of the methods at the end of the integration intervall.

The test results are summarized in the following items:

- RADAU5 is the most efficient code for all examples of the test set
- PSODE is less efficient, which may be caused by the adaption of the code to the distributed architecture
- LIRKM is arranged between the other codes
- LIRKM does not reach the accuracy of the other codes because the Newton-iterations and the solutions are only up to h^{q_i} identical to the Newton-iterations and the solutions of the underlying implicit Runge-Kutta method

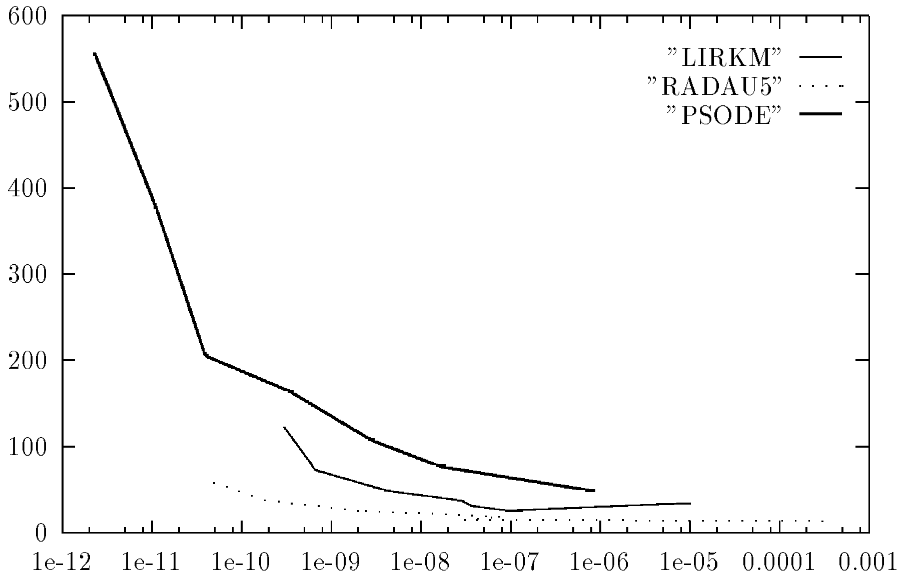

Figure 1: computing time versus accuracy at the end of the integration intervall for the example F5 of Gottwald/Wanner

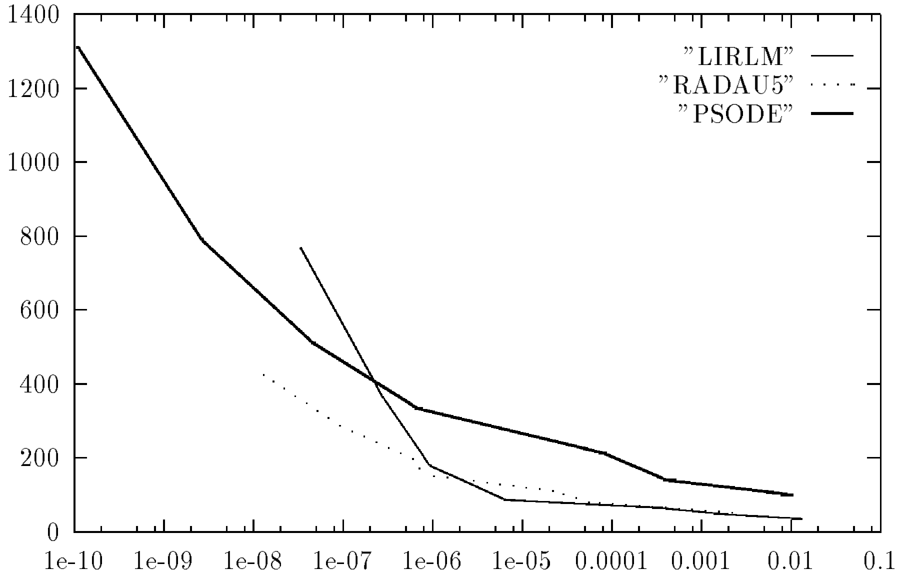

Figure 2: computing time versus accuracy at the end of the integration intervall for the example of Watkins/Hansonsmith

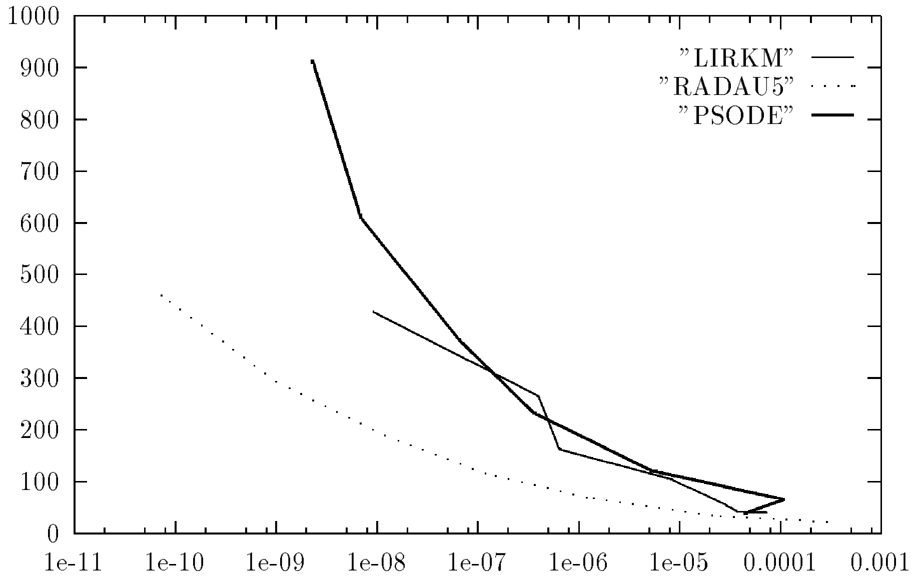

Figure 3: computing time versus accuracy at the end of the integration intervall for the reactor example

- only for the large W4-example (20 equations) LIRKM is comparable to RADAU5 for large tolerances; because for large dimensions the expense of the complex LU-decompositions compared to the real ones is more influential; for small dimensions the expensive iteration of LIRKM reduces the efficiency

To point out the dependance of the efficiency from the dimension of the problem we constructed two nonlinear examples with variable dimension. First we used the quadratic (Q-)example

$$y_i' = \sum_{j=1}^{i} a_{ij} y_j^2, \; i = 1, \ldots, n \tag{14}$$

with $t \in [t_0, t_e]$, $y_i(t_0) = y_{i,0}, i = 1, \ldots, n$. The exact solutions of the first equation of the system is $y_1(t) = -\frac{1}{a_{11}t + c_1}$. With the initial conditions we get $y_1(t_0) = -\frac{1}{a_{11}t_0 + c_1} = y_{1,0}$, i. e. $c_1 = -\frac{1}{y_{1,0}} - a_{11}t_0$. Thus the exact solution is $y_1(t) = -\frac{y_{1,0}}{a_{11}(t - t_0)y_{1,0} - 1}$. In the following we want to use a_{11} to get $c_1 = 0$ for simplicity. We have to set $a_{11} = -\frac{1}{y_{1,0}t_0}$ and get the solution $y_1(t) = \frac{t_0}{t} y_{1,0}$ for the first equation.

For the Jacobian f_y the (1,1)-element is $2a_{11}y_1(t) = -\frac{2}{t}$. Because the Jacobian is lower triangular for the first eigenvalue $\lambda_1(t)$ of the Jacobian we also get $\lambda_1(t) = -\frac{2}{t}$ and for the initial point $t = t_0$ it holds $\lambda_1(t_0) = -\frac{2}{t_0}$.

For the solution components $y_2, \ldots, y_n$ we again want to use the coefficients a_{ij} to get simple solutions of the form $y_i(t) = \frac{const}{t}$. Nevertheless the coeffi-

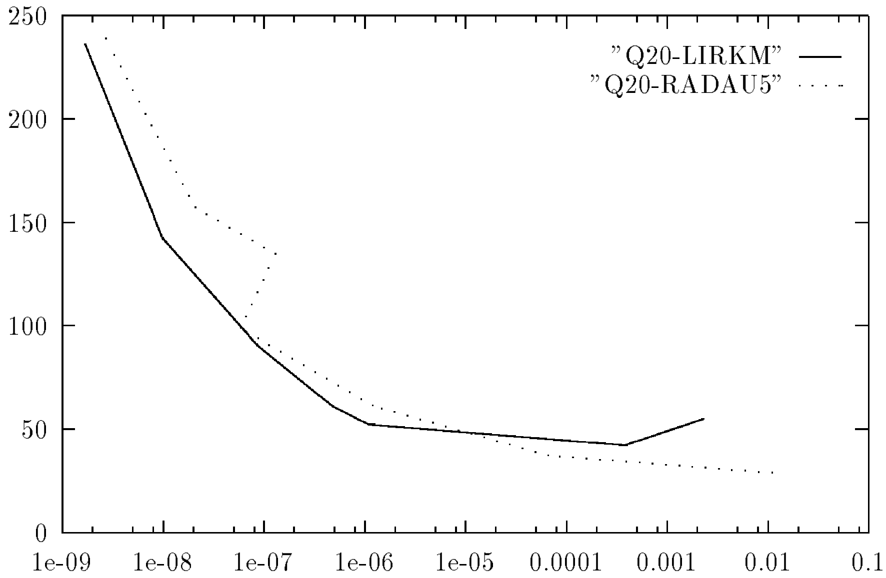

Figure 4: computing time versus accuracy at the end of the integration intervall for the Q-example (14) with n=20

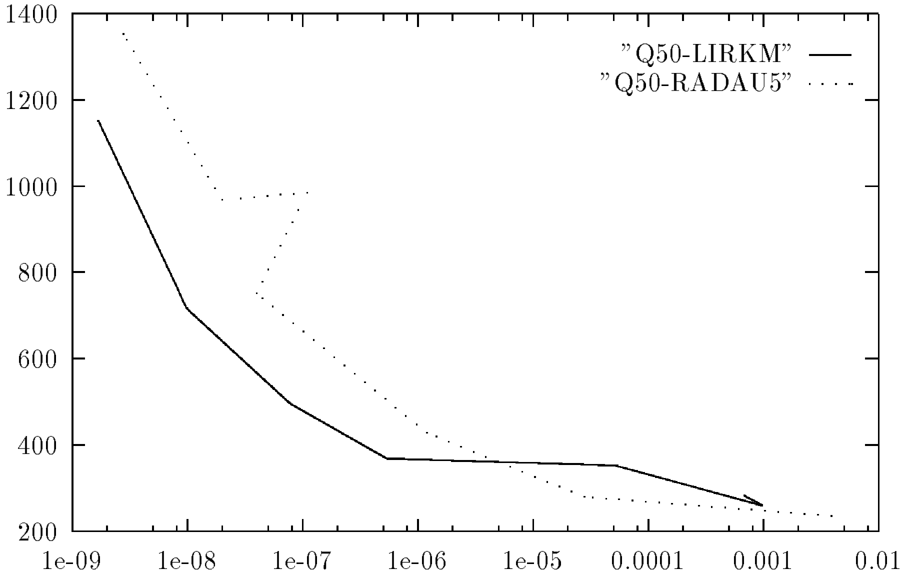

Figure 5: computing time versus accuracy at the end of the integration intervall for the Q-example (14) with n=50

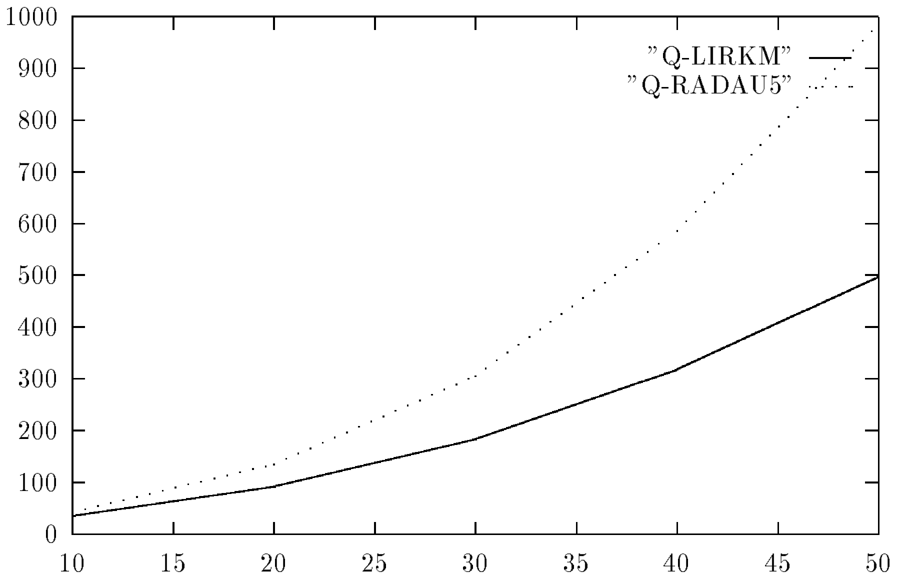

Figure 6: computing time versus number of equations with the Q-example (14) for TOL=1.e-6

cients a_{ij} are not determined uniquely, i. e. we can first choose initial eigenvalues $\lambda_2(t_0), \dots, \lambda_n(t_0)$ and then determine a_{ij}. For a given maximal initial eigenvalue λ_{max} we set

$$\lambda_i(t_0) = \lambda_1(t_0) + \frac{i-1}{n-1}(\lambda_{max} - \lambda_1(t_0)),\ i = 2, \dots, n. \tag{15}$$

Then for the diagonal elements of the coefficients matrix we get from $\lambda_i(t_0) = 2a_{ii}y_i(t_0)$ the expression

$$a_{ii} = \frac{\lambda_i(t_0)}{2y_{i,0}},\ i = 2, \dots, n. \tag{16}$$

The non-diagonal elements we want to be equally, i. e. we have to choose

$$a_{ij} = -\frac{y_{i,0} + a_{ii}y_{i,0}^2 t_0}{t_0 \sum_{j=1}^{i-1} y_{j,0}^2},\ i = 2, \dots, n,\ j = 1, \dots, i-1. \tag{17}$$

For our computations with $n = 10, 20, 30, 40, 50$ we used $t_0 = 0.01,\ t_e = 1,\ \lambda_{max} = -10^8,\ y_{i,0} = 100,\ i = 1, \dots, n$. Thus we get the exact solutions $y_i(t) = \frac{1}{t}$ with $y_i(t_0) = 100$ and $y_i(t_e) = 1$. For the first eigenvalue it holds $\lambda_1(t) = -\frac{2}{t}$, i. e. $\lambda_1(t_0) = -200$ and $\lambda_1(t_e) = -2$. For λ_n we have $\lambda_n(t) = \frac{t_0}{t}\lambda_n(t_0)$, i. e. with our initial values we get $\lambda_n(t_0) = -10^8, \lambda_n(t_e) = -10^6$.

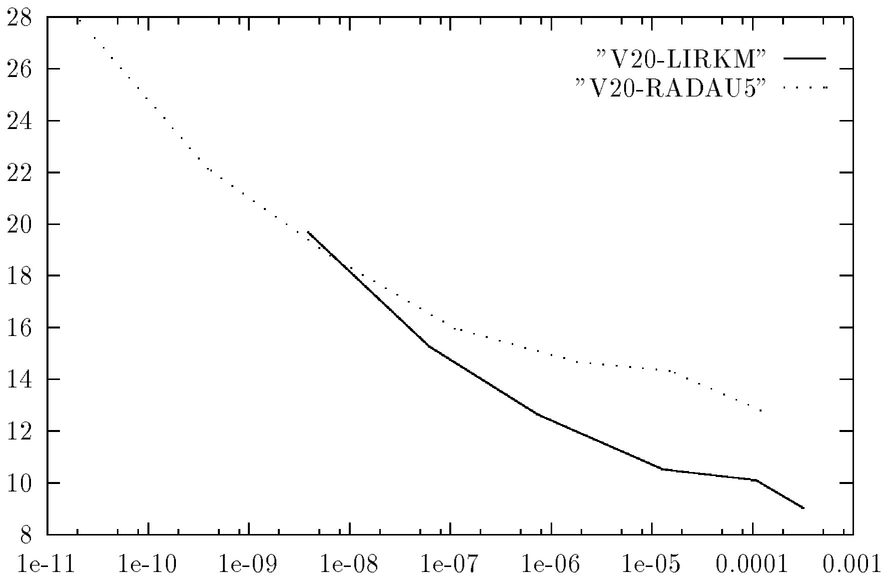

Figure 7: computing time versus accuracy at the end of the integration intervall for the V-example (18) with n=20

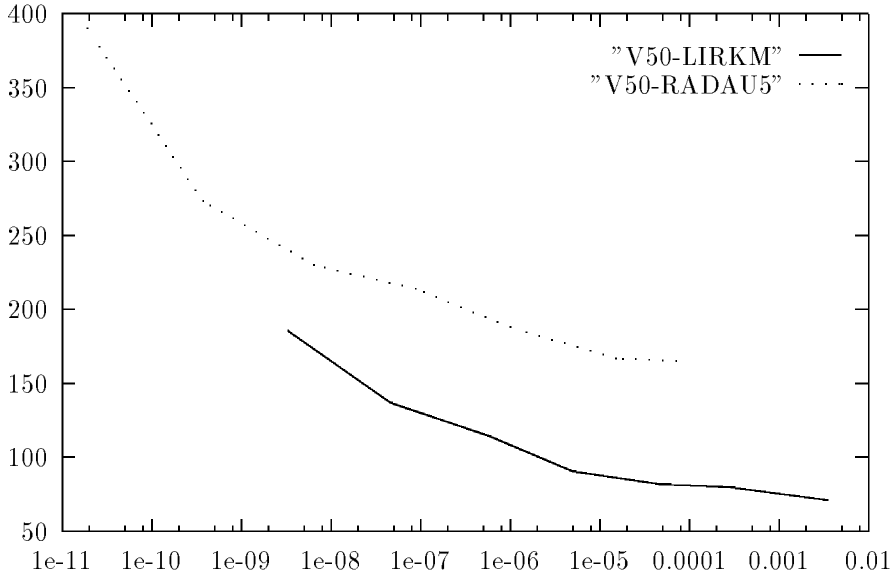

Figure 8: computing time versus accuracy at the end of the integration intervall for the V-example (18) with n=50

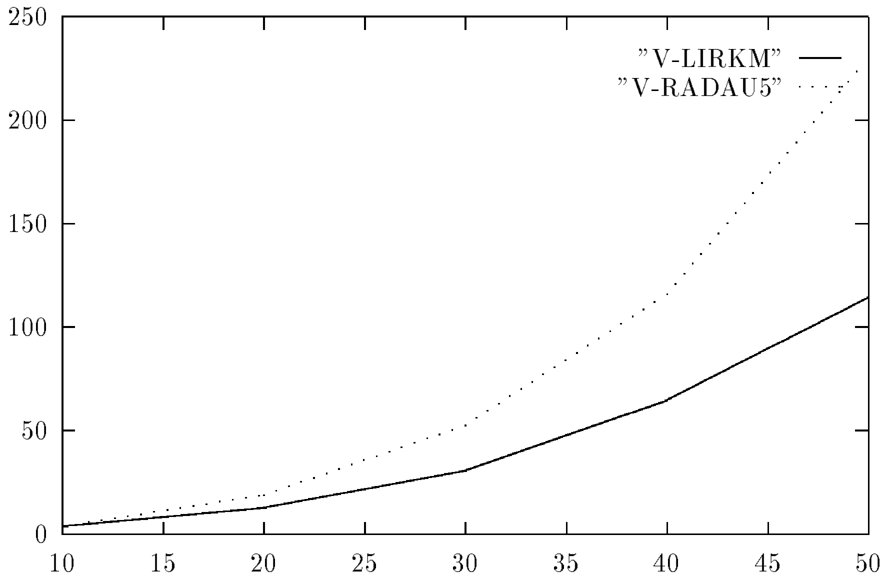

Figure 9: computing time versus number of equations with the V-example (18) for TOL=1.e-6

A second nonlinear example of variable dimension is the (V-)example

$$\begin{aligned} y_1' &= -0.1y_1 \\ y_i' &= (y_{i-1}y_{i+2} - y_{i+1}y_i)^2 - 0.1iy_i, \ i = 2, \ldots, n-2 \\ y_{n-1}' &= -0.1(n-1)y_{n-1} \\ y_n' &= -0.1ny_n \end{aligned} \tag{18}$$

(see [11]) with initial values and exact solution $y_i(t) = e^{-0.1it}$. The purpose of this example was also to show how the efficiency depends on the dimension. Because the standard LU-decomposition would take advantage of the banded structure we modified the LU-decomposition for this example to ignore zeros in the Jacobian and to do all arithmetical operations. Again we used the dimension $n = 10, 20, 30, 40, 50$.

Summary

The numerical test show that for problems with slowly varying Jacobian the parallel LIRKM-Implementation is comparable with the parallel implementation of RADAU5 and PSODE for large and medium tolerances ($\geq 10^{-6}$). For large problems ($n > 10$) LIRKM becomes superior to PSODE and RADAU5.

References

[1] Bruder, J.; Strehmel, K.; Weiner, R. (1988): Partitioned Adaptive Runge-Kutta Methods for the solutions of Nonstiff and Stiff Systems, Numer. Math. 52, 621-638

[2] Bruder, J. (1993): Linearly–implicit Runge–Kutta methods based on implicit Runge–Kutta methods, Applied Numerical Mathematics 13, 33–40

[3] Burrage, K. (1993): Parallel methods for initial value problems, Applied Numerical Mathematics 11, 5–25

[4] Gottwald, B.A.; Wanner, G. (1981): A reliable Rosenbrock integrator for stiff differential equations, Computing 26, 355-360

[5] Hairer, E.; Wanner, G. (1991): Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems, Springer-Verlag Berlin Heidelberg

[6] van der Houwen, P.J.; Sommeijer,B.P. (1993): Analysis of parallel diagonally implicit iteration of Runge–Kutta methods, Applied Numerical Mathematics 11, 169–188

[7] Keeling, S. L. (1989): On implicit Runge–Kutta methods with a stability function having distinct real poles, BIT 29, 91–109

[8] Strehmel, K.; Weiner, R. (1982): Behandlung steifer Anfangswertprobleme gewöhnlicher Differentialgleichungen mit adaptiven Runge-Kutta Methoden, Computing 29, 153-165

[9] Strehmel, K.; Weiner, R. (1992): Linear-implizite Runge-Kutta-Methoden und ihre Anwendungen, Teubner-Texte zur Mathematik, Band 127

[10] Watkins, D.S.; Hansonsmith, R.W. (1983): The numerical solution of seperably stiff systems by precise partitioning, ACM-TOMS, Vol. 9, No.3, 293-301

[11] Weiner, R.; Schmitt, B.A. (1995): Consistency of Krylow-W-methods in initial value problems, Report No. 14, Fachbereich Mathematik und Informatik, Martin-Luther-Universität Halle-Wittenberg