Computing 58, 173–185 (1997)

Computing

Multirate Extrapolation Methods for Differential Equations with Different Time Scales

Ch. Engstler and **Ch. Lubich**, Tübingen

Received October 30, 1995; revised July 26, 1996

Abstract — Zusammenfassung

Multirate Extrapolation Methods for Differential Equations with Different Time Scales. A multirate extrapolation method is developed for the integration of differential equations whose components evolve at different time scales. Numerical work is focused on fast components. The partitioning into different levels of slow to fast components is obtained automatically during the extrapolation process. The method has been implemented in the Fortran code MURX.

AMS Subject Classification: 65L05.

Key words: Ordinary differential equations, multirate methods, multiple time stepping, extrapolation.

Extrapolation mit komponentenweise verschiedenen Ordnungen für Differentialgleichungen mit unterschiedlichen Zeitskalen. Für die numerische Integration von Differentialgleichungen, deren Komponenten sich unterschiedlich rasch verändern, wird ein Extrapolationsverfahren vorgeschlagen, das verschiedene Ordnungen für verschiedene Komponenten adaptiv wählt. Der Rechenaufwand für langsame Lösungskomponenten wird dabei erheblich herabgesetzt. Die Aufteilung in verschiedene Stufen langsamer und schneller Komponenten ergibt sich ohne weiteres Zutun während des Extrapolationsvorganges. Das Verfahren wurde im Fortran-Programm MURX implementiert.

1. Introduction

When integrating systems of differential equations whose components evolve and persist at different time scales, one would like to use numerical methods that do not expend unnecessary numerical work on slowly changing solution components. This paper concerns problems where a small number of fast changing components restricts the stepsize of numerical integrators and, unlike stiff problems, both fast and slow components are present throughout the integration interval. The partition into fast and slow components may vary with time (e.g., in steep moving wavefronts) and need not be sharp so that various scales from slow to fast components exist.

The first article to treat problems of this type appears to be a paper by Rice [10], who proposes 'split' Runge-Kutta methods (which would nowadays be called 'multirate' Runge-Kutta methods) that use different time steps to integrate fast and slow solution components. Such an approach has been developed further by Gear and Wells [3], Günther and Rentrop [4], and Skelboe and Andersen [11].

In a closely related approach, known as 'multiple time stepping', see Biesiadecki and Skeel [1], one does not split solution components, but instead the right-hand side function as a sum of fast and slowly changing functions which are then evaluated with different rates.

A difficulty with the existing multirate techniques is that they assume a clear-cut partition of the system into fast and slow components, which has to be known before performing a macro-step. A 'slowest first' or 'fastest first' strategy [3] has to be employed, and the potential of such an approach depends on the coupling between components.

In the present paper, we propose a multirate method based on Richardson extrapolation. The basic idea is to stop building up the extrapolation tableau for components that have been recognized to be already sufficiently accurate. Since local error estimates are available at all extrapolation levels, this provides an inexpensive strategy for dynamic partitioning into several classes of slow to fast components. In contrast to previous multirate methods, slow and fast components are integrated simultaneously over a macro-step. However, slow components are inactivated at early extrapolation levels, and the refinement of the solution by high-order Richardson extrapolation is restricted to the faster components. The method proposed here has been implemented in MURX, a multirate extrapolation code written in Fortran.

In Section 2 we begin by recalling classical Richardson extrapolation. Multirating for the extrapolated Euler scheme is described in Sections 3 and 4, and a defect control mechanism to verify the inactivation strategy in Section 5. A simple extension of the techniques to the situation of 'multiple time stepping' for split right-hand sides is given in Section 6. Finally, we present in Section 7 numerical results of MURX applied to an example from astrophysics.

2. Classical Extrapolation

When the initial value problem

$$y' = f(t, y), \qquad y(t_0) = y_0$$

is discretized by the explicit Euler method with step size h, this gives at $t_{n+1} = t_n + h$ a first-order approximation to $y(t_{n+1})$ by

$$y_{n+1} = y_n + h f(t_n, y_n), \qquad n \geq 0,$$

which we denote

$$y(t, h) = y_n \qquad \text{for } t = t_0 + nh.$$

The error has an asymptotic h-expansion [5, Ch. II.8]

$$y(t, h) - y(t) = he_1(t) + h^2 e_2(t) + \ldots + h^N e_N(t) + O(h^{N+1}),$$

with $e_j(t_0) = 0$ for all j. The error terms are successively eliminated by Richardson extrapolation. Given a macro-step size H, one constructs Euler approximations to $y(t_0 + H)$ with step sizes $h_j = H/n_j$, where $\{n_j\}$ is the step number sequence, e.g., $\{n_j\} = \{1, 2, 3, 4, 5, 6, 8, 10, 12, 16, 20, 24, \ldots\}$. The formulas

$$T_{j,1} = y(t_0 + H, h_j)$$

$$T_{j,k+1} = T_{j,k} + \frac{T_{j,k} - T_{j-1,k}}{(n_j/n_{j-k}) - 1} \qquad \text{for } k+1 \le j$$

define the extrapolation tableau

$$\begin{array}{llll} T_{11} & & & \\ T_{21} & T_{22} & & \\ T_{31} & T_{32} & T_{33} & \\ \vdots & \vdots & \vdots & \ddots \end{array}$$

The error of T_{jk} is of the form

$$T_{jk} - y(t_0 + H) = c_{jk}^{(k)} e_k(t_0 + h) H^k + \ldots + c_{jk}^{(N)} e_N(t_0 + H) H^N + O(H^{N+1}),$$

where the coefficients $c_{jk}^{(l)}$ depend only on the step number sequence. An important property of the extrapolation tableau is that numerical error estimators are available at all levels: The subdiagonal difference $T_{k,k} - T_{k,k-1}$ is taken as an approximation to the error at level k. This is used for step size and order selection [2,5]. Another very useful option of extrapolation methods is their continuous solution approximation: Here, the idea is to first compute accurate approximations to solution derivatives of sufficiently high order at both endpoints of the interval $[t_0, t_0 + H]$ by extrapolation of divided differences of Euler approximations, and then to construct a polynomial

$$P_k(t) \approx y(t)$$

having these endpoint values and derivatives. Approximation properties of this approach have been studied in [6].

3. Multirate Extrapolation: Strategies at the Second Level

We will see that multirating is easy and natural within extrapolation methods. Let us assume that we have already computed the entries T_{11}, T_{21}, T_{22} of the extrapolation tableau.

We use the following simple strategy: If the ith component of the error estimator is below the given tolerance,

$$|T_{22}^i - T_{21}^i| < 0.1 \cdot \text{tol}, \tag{3.1}$$

then we accept T_{22}^i as solution approximation,

$$T_{22}^i \approx y^i(t_0 + H),$$

and make component i *inactive*, meaning that we are not willing to make any further evaluations of f^i, the ith component of the right-hand side function.

Remark. The safety factor 0.1 in (3.1) is more or less arbitrary. In the actual implementation, (3.1) would use a scaled norm: For

$$\mathrm{err}^i = |T_{22}^i - T_{21}^i|/w^i \qquad \text{with } w^i = \mathrm{rtol} \cdot \max\bigl(|y_0^i|, |T_{22}^i|\bigr) + \mathrm{atol}^i$$

one requires $\mathrm{err}^i < 0.1$. To reduce the risk of accidental cancellation of error terms in $T_{22}^i - T_{21}^i$, one might check (3.1) for small groups of components rather than single components, e.g., for pairs of odd/even components or physically reasonable groups such as the coordinates of a mass point.

At the next level of extrapolation, we then have to compute T_{31} for the remaining (= *active*) components. We split the solution vector and the function into their active and inactive components,

$$y = \begin{pmatrix} y^A \\ y^I \end{pmatrix}, \quad f = \begin{pmatrix} f^A \\ f^I \end{pmatrix},$$

where f^I is not to be evaluated any more. To compute T_{31}^A, we would have to perform Euler steps with $h = H/3$:

$$\begin{aligned} y_1^A &= y_0^A + h f^A(t_0, y_0) \\ y_2^A &= y_1^A + h f^A\bigl(t_1, y_1^A, \underline{y_1^I}\bigr) \\ T_{31}^A &= y_3^A = y_2^A + h f^A\bigl(t_2, y_2^A, \underline{y_2^I}\bigr). \end{aligned} \tag{3.2}$$

At this stage we have the problem of how to obtain the values y_1^I and y_2^I. There are at least two possibilities.

First strategy: Approximation to the *exact* solution at intermediate steps.

Here, one takes as y_1^I and y_2^I the values obtained from the continuous solution approximation of the extrapolation method at $t_0 + h$ and $t_0 + 2h$:

$$P_2^I(t_0 + h) \approx y_1^I, \quad P_2^I(t_0 + 2h) \approx y_2^I,$$

where $P_2^I(t)$ is the quadratic polynomial with $P_2^I(t_0) = y_0^I$, $(d/dt)P_2^I(t_0) = f^I(t_0, y_0)$, $P_2^I(t_0 + H) = T_{22}^I$. Then, we have the error bound

$$|P_2^I(t) - y^I(t)| \le C^I \cdot H^3 \qquad \text{for } t \in [t_0, t_0 + H],$$

where the constant C^I depends on d^3y^I/dt^3 and the error of T_{22}^I which is small provided that (3.1) was a reasonable error estimate. Since the error of T_{22}^I depends itself mainly on d^3y^I/dt^3, we can expect that in generic situations (that is, unless some cancellations of error terms have occurred in (3.1)) the error $P_2^I(t) - y^I(t)$ is of the same size as the error of T_{22}^I, and therefore sufficiently small. Instead of giving a more detailed analysis, we refer to the error bounds in [6]. For the computation of the active components T_{41}^A, T_{51}^A, etc., in the higher extrapolation levels, one would again use $P_2^I(t)$ to approximate the required

values of the inactive solution components. There is, however, a difficulty. Since this approach amounts to building up the extrapolation tableau for the differential equation

$$\frac{d}{dt}y^A = f^A\big(t, y^A, P_2^I(t)\big)$$

from the third level onwards, we must not use the previously computed values T_{21}^A, T_{22}^A for extrapolation, since they have a different asymptotic expansion that corresponds to the original differential equation $y' = f(t, y)$. A remedy would be to recompute T_{21}^A and T_{22}^A for the modified differential equation, which necessitates however re-evaluation of f^A. The situation aggravates at higher levels, where one would have to recompute the whole extrapolation tableau up to the current level. This is avoided in the second approach.

Favoured strategy: Approximation to the *Euler steps*.

In (3.2), the correct underscored values would be Euler approximations, which we now try to approximate without, however, using the inactivated function components f^I. We start from the asymptotic expansion

$$y^I(t,h) = y^I(t) + he_1^I(t) + \ldots.$$

We have already an accurate approximation $P_2^I(t)$ to $y^I(t)$. To approximate $e_1^I(t)$, we note that up to terms of size $C^I \cdot H^3$ with a constant C^I of the same type as previously, we have for $h = H/2$ and $t = t_0 + H$

$$T_{21}^I \approx T_{22}^I + H/2 \cdot e_1^I(t_0 + H),$$

and for $h = H/2$ and $t = t_0 + H/2$

$$y^I(t_0 + H/2, H/2) \approx P_2^I(t_0 + H/2) + H/2 \cdot e_1^I(t_0 + H/2).$$

We now replace $e_1^I(t)$ by the quadratic polynomial $q_2^I(t)$ which vanishes at t_0 and for which the above two relations become an equality:

$$q_2^I(t_0 + \theta H) = (4a - b)\theta + (2b - 4a)\theta^2 \approx e_1^I(t_0 + \theta H),$$

where $a = 2/H \cdot (y^I(t_0 + H/2, H/2) - P_2^I(t_0 + H/2))$ and $b = 2/H \cdot (T_{21}^I - T_{22}^I)$. By construction we have

$$|q_2^I(t) - e_1^I(t)| \le C^I \cdot H^3,$$

where C^I is nearly a constant of the previously encountered type, except that it now depends also on derivatives of $e_1^I(t)$. (Again, only the inactive — or slow — components enter into C^I.) This gives us the approximate Euler steps

$$P_2^I(t) + hq_2^I(t) \approx y^I(t,h), \tag{3.3}$$

which are to be used with $h = H/3$ and $t - t_0 = H/3$, $2H/3$ in (3.2), and with $h = H/4, H/5, \ldots$ at higher levels of the extrapolation tableau. Provided that (3.3) is sufficiently accurate, we work with the correct asymptotic expansions in continuing the extrapolation tableau.

4. Multirate Extrapolation at the First and Higher Levels

From the second step onwards, we can inactivate slow components already at the first level: A component is made inactive if the difference between the first Euler step and the linear extrapolation of two successive solution values is below the required tolerance. We then take the linear interpolation

$$P_1(t) = y_0 + (t - t_0) f(t_0, y_0) \approx y(t,h)$$

of the inactive components to obtain the required Euler steps at higher extrapolation levels.

We now turn to inactivation at higher levels: A component i is inactivated at level k, if it is still active at level $k-1$ and if

$$|T^i_{k,k} - T^i_{k,k-1}| < 0.1 \text{ tol.} \tag{4.1}$$

We denote by $y^{[k]}$ the vector of all y-components that are inactivated at level k. For these components we approximate the Euler steps at levels higher than k by

$$P^{[k]}_k(t) + hq^{[k]}_k(t) + \ldots + h^{k-1} q^{[k]}_2(t) \approx y^{[k]}(t,h), \tag{4.2}$$

where $P^{[k]}_k(t) \approx y^{[k]}(t)$ is the continuous solution approximation of [6], and $q^{[k]}_j(t) \approx e^{[k]}_{k-j+1}(t)$ are polynomials of degree j that are determined as follows: For $t = t_0$, we set $q^{[k]}_j(t_0) = e^{[k]}_{k-j+1}(t_0) = 0$. For $t = t_0 + H$, we obtain

$$\epsilon_{k-j+1} := q^{[k]}_j(t_0 + H)$$

by requiring that

$$T^{[k]}_{i,1} = T^{[k]}_{k,k} + h_i \epsilon_1 + \ldots + h_i^{k-1} \epsilon_{k-1}$$

for $i = 2, \ldots, K$, which gives a Vandermonde system for the ϵ_j. One could instead use a similar system where the kth row instead of the first column of the extrapolation tableau is used, which is advantageous for the implementation. We now consider the values $q^{[k]}_j(t_0 + ih_j)$ for $j = 2, \ldots, k$ and $i = 1, \ldots, j-1$ as the remaining unknowns for the determination of $a^{[k]}_j(t)$. These are obtained by requiring equality in (4.2) for $t = t_0 + ih_j$ ($j = 2, \ldots, k$ and $i = 1, \ldots, j-1$) which are the time steps where the Euler steps have already been computed. This gives a linear system of $(k-1)k/2$ equations in as many unknowns. For $k = 3$ and $k = 4$ we have solved this system. (We did not try for $k = 5$, because there the expense of computing the polynomials would be offset only by very expensive function evaluations.) For $k = 3$ a calculation yields

$$\begin{pmatrix} H\, q_3(t_0 + H/3) \\ H\, q_3(t_0 + 2H/3) \\ H^2\, q_2(t_0 + H/2) \end{pmatrix} = \begin{pmatrix} 6 & 3 & -32/9 \\ 3 & 6 & -32/9 \\ -81/8 & -81/8 & 12 \end{pmatrix} \begin{pmatrix} \delta_1 + H\,\epsilon_1/32 \\ \delta_2 + H^2\,\epsilon_2/81 \\ \delta_3 - 2H^2\,\epsilon_2/81 \end{pmatrix}$$

with

$$\begin{aligned} \delta_1 &= y(t_0 + H/3, H/3) - P_3(t_0 + H/3) \\ \delta_2 &= y(t_0 + 2H/3, H/3) - P_3(t_0 + 2H/3) \\ \delta_3 &= y(t_0 + H/2, H/2) - P_3(t_0 + H/2). \end{aligned}$$

For ease of notation, we have omitted the omnipresent superscript $[k]$. With this construction, we get via (4.2) an approximation to $y^{[k]}(t,h)$. Its error is $O(H^{k+1})$ when the Euler steps up to level k have been computed exactly. It has been observed numerically that errors in the Euler steps, due to previously inactive components, are propagated in a harmless way. For improved accuracy, the degree j of $q_j(t)$ is one higher than absolutely necessary.

At levels higher than k, the required Euler step components $y^{[k]}(t,h)$ are replaced by the polynomials of (4.2). These are inexpensive to evaluate, at least for small k. Moreover, all the required values at all levels can be computed in parallel.

5. Failure of the Inactivation Strategy, Defect Control

In some contrived situations, the inactivation strategy (4.1) can fail spectacularly. Consider the linear example with a shift matrix

$$y' = \begin{pmatrix} 0 & & & 0 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & 1 & 0 \end{pmatrix} y, \quad y(0) = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

When only one or two Euler steps are applied to this system, then all components except the first three are still 0, independently of the step size. Consequently, $T_{22}^i - T_{21}^i = 0$ for $i > 3$, and hence components $4, 5, \ldots$ are inactivated. The wrong values $T_{22}^i = 0$ are accepted.

To prevent the program from delivering wrong results without ringing alarm, the remedy is to build in a defect control which can be obtained at negligible additional cost. Here, one compares the function evaluated at the endpoint of the step (which would anyway be computed for the next step) with the extrapolated backward differences of the Euler steps (which are anyway computed for the continuous solution approximation). If these two values differ intolerably, then the step is rejected and the incriminated components are locked for the inactivation strategy.

We remark, however, that in our numerical experiments we encountered a failure of the inactivation strategy only once for a special choice of initial values.

6. Splitting the Function

A technique that has come up in molecular dynamics is to split the right-hand side function of the differential equation

$$y' = f_1(t,y) + f_2(t,y)$$

where, for example, f_1 is cheap to evaluate and rapidly varying, whereas f_2 is expensive but fortunately slowly changing, so that it is tempting to evaluate f_2 less often than f_1. The previously described approach extends to this situation in a very simple way. We introduce $z_i = f_i(t, y)$ and rewrite the equation as a differential-algebraic system

$$y' = z_1 + z_2$$
$$z_1 = f_1(t, y)$$
$$z_2 = f_2(t, y)$$

with 'fast' components z_1 and 'slow' components z_2, or in more general terms,

$$\begin{aligned} y' &= F(t, y, z) \\ z &= G(t, y). \end{aligned} \tag{6.1}$$

Euler's scheme extends straightforwardly to such a system,

$$y_{n+1} = y_n + hF(t_n, y_n, z_n)$$
$$z_{n+1} = G(t_{n+1}, y_{n+1}).$$

Multirate extrapolation can be done on this scheme (in both y and z) in the same way as above. Again, the partitioning into slow and fast and various shades of intermediate components is obtained automatically, controlled by the given error tolerance.

7. Numerical Experiments

We have implemented the multirate method in a Fortran code MURX (**MU**lti-**R**ate e**X**trapolation). The code MURX together with the subroutines for the example described below is available via anonymous ftp at na.uni-tuebingen.de in the directory /pub/codes. The order and step-size selection strategies were adapted from the code ELEX of [6], which implements the classical extrapolated Euler method. In the multirate code, only the active components at each level affect the order and step size control. The code MURX has been written to be applicable to problems of the form (6.1). The calling sequence of MURX is like that of a standard ODE integrator:

subroutine murx(n, m, fcn, gcn, t, y, tend, h, rtol, atol, itol, ...)

where the given arguments have their usual meaning, and the dots stand for additional control and workspace parameters. The integers n and m are the dimensions of the vectors y and z in (6.1). fcn and gcn are user-supplied subroutines (external) required in the following format:

subroutine fcn(n, m, t, y, z, f, activey, ...)

subroutine gcn(n, m, t, y, g, activez, ...)

These subroutines should return in f and g those components of $F(t, y, z)$ and $G(t, y)$, respectively, that are declared active by the logical arrays activey and

activez. The arguments indicated by dots communicate workspace and problem parameters. The choice $m = 0$ for an unsplit right-hand side is possible, in which case gcn should be included as a dummy subroutine.

We have performed numerical experiments with a problem that originates in astrophysics, in the computation of the mass distribution in accretion disks [8]. Such a disk forms around a primary star which attracts mass from a secondary star. Motion is described by the Navier-Stokes equations which here are discretized in space by the smoothed particle hydrodynamics (SPH) method [7], leading to a large system of ordinary differential equations whose components represent positions and velocities of "smoothed particles". These move around the star in a Kepler-like motion which is perturbed by viscosity effects. Particles near the center move much faster than distant ones.

The equations of motion of a two-dimensional model problem [8] take the form

$$\frac{dx_{i\alpha}}{dt} = v_{i\alpha}$$

$$\frac{dv_{i\alpha}}{dt} = -GMr_i^{-3/2}x_{i\alpha} + z_{i\alpha}$$

where the viscosity terms $z_{i\alpha}$ are given as

$$z_{i\alpha} = \sum_j \frac{m\nu}{\rho_i \rho_j} \sum_{\beta=1}^{2} (\rho_j \sigma_{j\alpha\beta} + \rho_i \sigma_{i\alpha\beta}) W_{ij,\beta}.$$

Here $\alpha = 1, 2$, and i runs through all particles. Further, $x_i = (x_{i1}, x_{i2})$ is the position of the ith particle, $r_i = |x_i| = \sqrt{x_{i1}^2 + x_{i2}^2}$ is its distance to the center, and $v_i = (v_{i1}, v_{i2})$ is its velocity. The mass of a particle is denoted by m, and M is the mass of the central star. G is the constant of gravity, and ν is a viscosity coefficient. Further, we have the densities

$$\rho_i = m \sum_j W_{ij} \qquad \text{with } W_{ij} = W(|x_i - x_j|),$$

where W is the smoothing kernel of the SPH method. This is a cubic spline which vanishes outside the smoothing length l and whose integral satisfies a normalization condition [7, 8]. We have set

$$W_{ij,\alpha} = \frac{\partial}{\partial x_{i\alpha}} W(|x_i - x_j|) = W'(r_{ij}) \cdot (x_{i\alpha} - x_{j\alpha}) / r_{ij}$$

with $r_{ij} = |x_i - x_j|$. Finally, the viscosity term depends on

$$\sigma_{i\alpha\beta} = \sum_k \frac{m}{\rho_k} \left([v_{k\alpha} - v_{i\alpha}] W_{ik,\beta} + [v_{k\beta} - v_{i\beta}] W_{ik,\alpha} - \frac{2}{3} \delta_{\alpha\beta} \sum_{\gamma=1}^{2} [v_{k\gamma} - v_{i\gamma}] W_{ik,\gamma} \right),$$

where $\delta_{\alpha\beta}$ is Kronecker's delta. In the above formulas, the sums over j and k extend over all particles within the smoothing length from particle i.

Our numerical experiments used data from [8]. As starting values we used a configuration with 20000 particles as shown in the first picture in Fig. 1, which evolved from a dissolving initial annulus. The problem data and initial values are available as Fortran subroutines together with MURX.

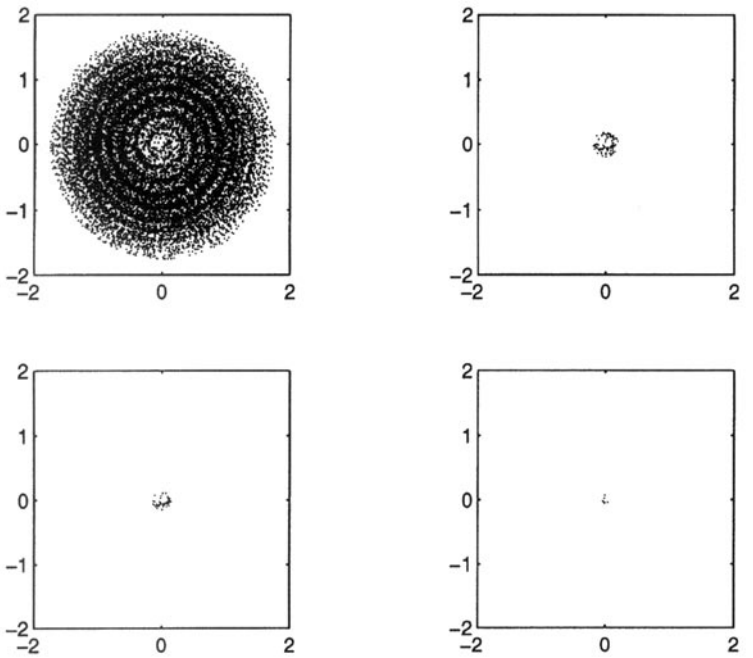

Figure 1. Active particles at different extrapolation levels, tol = 10^{-4}

We applied MURX to this problem in two different ways: The first variant uses the differential equation directly without a splitting of the right-hand side, so that no z-variables are present. In the second variant, we used the viscosity terms $z_{i\alpha}$ as the z-variables of (6.1). Here, we also prevented the code from inactivating y-variables, because $F(t, y, z)$ is very cheap to evaluate for this problem. In the following we refer to the first version as MURX(y), and to the split version as MURX(y, z).

Figures 1 and 2 show how MURX(y) works at this problem for two different local error tolerances. The pictures show the active particles at the first, second, third and fourth extrapolation level. Here, a particle is considered as active if at least one of its velocity components is active. Figure 1 shows the typical behaviour for a moderate tolerance (rtol = atol = 10^{-4}), where a large portion of components is inactivated already at early extrapolation levels. For more stringent tolerances (rtol = atol = 10^{-8} in Fig. 2) the inactivation is shifted to higher levels.

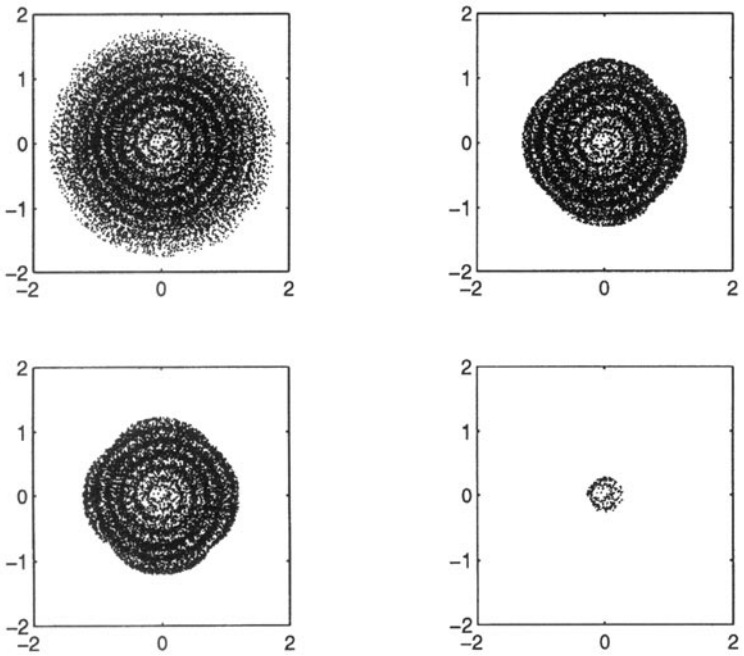

Figure 2. Active particles at different extrapolation levels, tol = 10^{-8}

In Figs. 3 and 4 we show the numbers of function evaluations and computing times versus the attained accuracy, measured in the Euclidean norm scaled by the square root of the dimension, at the end of an integration interval of length 200. We compare the results for the two types of MURX, for ELEX, the extrapolated Euler method as implemented by Hairer and Ostermann [6], and for the Runge-Kutta code DOP853 from [5], which is based on the eighth-order method of Dormand and Prince [9]. The code DOP853 is known to perform very efficiently for all tolerances, see the numerical comparisons in Section II.10 of [5], and has therefore been included as a reference general-purpose code. The codes were used with tolerances atol = rtol = $10^{-3}, \ldots, 10^{-10}$. For MURX, we give the number of weighted function evaluations, where the percentage of active components is counted in each function evaluation. In MURX(y) we count the numbers of weighted f-evaluations, whereas for the split version MURX(y, z) we count only the weighted G-evaluations, which are much more expensive that the F-evaluations. Figure 3 shows a clear advantage of MURX at all tolerances. However, due to the relatively high cost of interpolations at higher levels as compared to the cost of function evaluations in this problem, the situation looks different for the comparison of computing times given in Fig. 4. MURX(y, z) is still advantageous, but MURX(y) is less efficient than DOP853 for accuracies less than 10^{-4}. The behaviour is partly explained by the inactivation at later extrapolation levels, as shown in Fig. 2 as compared to Fig. 1. The cross-over point of MURX and DOP853 depends strongly on the relative costs of function evaluations and interpolation. For the limit of very expensive

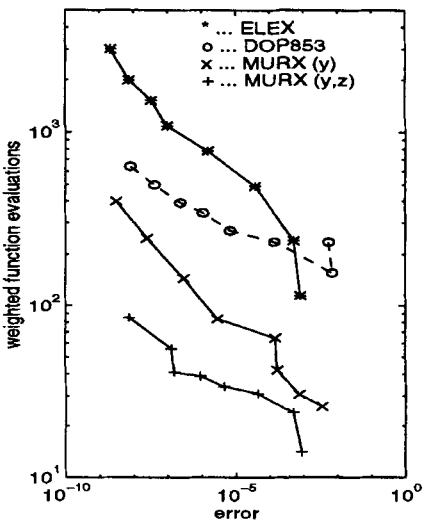

Fig 3. Function evaluations

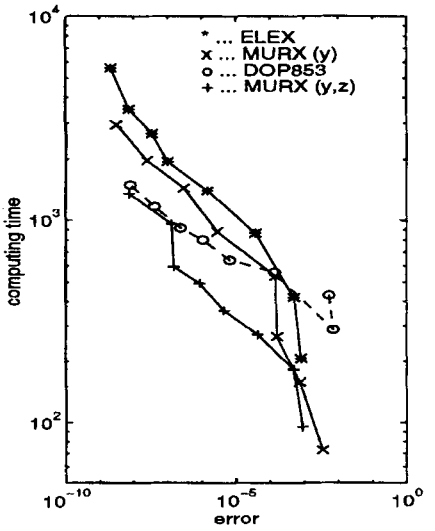

Fig 4. Computing times

function evaluations, Fig. 3 shows the potential of gains in computing time by MURX.

Acknowledgements

We are grateful to F. Ott for providing data and helpful information concerning the accretion disk example. We thank J. Butcher for pointing out reference [10].

References

[1] Biesiadecki, J. J., Skeel, R. D.: Dangers of multiple time step methods. J. Comp. Phys. *109*, 318–328 (1993).
[2] Deuflhard P.: Order and stepsize control in extrapolation methods. Numer. Math. *41*, 399–422 (1983).
[3] Gear, C. W., Wells, R. R.: Multirate linear multistep methods. BIT *24*, 484–502 (1984).
[4] Günther, M., Rentrop, P.: Multirate ROW-methods and latency of electric circuits. Appl. Numer. Math. *13*, 83–102 (1993).
[5] Hairer, E., Nørsett, S. P., Wanner, G.: Solving ordinary differential equations I. Nonstiff problems, 2nd revised ed. Springer Series in Computational Mathematics 8, Berlin Heidelberg New York Tokyo: Springer, 1993.
[6] Hairer, E., Ostermann, A.: Dense output for extrapolation methods. Numer. Math. *58*, 419–439 (1990).
[7] Monaghan, J. J.: Smoothed particle hydrodynamcis. Ann. Rev. Astron. Astrophys. *30*, 543–574 (1992).
[8] Ott, F.: Smoothed particle hydrodynamics. Grundlagen und Tests eines speziellen Ansatzes für viskose Wechselwirkungen. Diploma thesis, Inst. f. Astronomie und Astrophysik, Univ. Tübingen 1995.
[9] Prince, P. J., Dormand, J. R.: Practical Runge-Kutta processes. SIAM J. Sci. Stat. Comput. *10*, 977–989 (1989).

[10] Rice, R. C.: Split Runge-Kutta methods for simultaneous equations. J. Res. Natl. Bur. Standards. *64B*, 151–170 (1960).
[11] Skelboe, S., Andersen, P. U.: Stability properties of backward Euler multirate formulas. SIAM J. Sci. Stat. Comp. *10*, 1000–1009 (1989).

Ch. Engstler, Ch. Lubich
Mathematisches Institut
Universität Tübingen
Auf der Morgenstelle 10
D-72076 Tübingen
Germany
engstler@na.uni-tuebingen.de
lubich@na.uni-tuebingen.de