

Efficient and Reliable Integration Methods for Particle Tracing in Unsteady Flows on Discrete Meshes

Christian Teitzel, Roberto Grosso and Thomas Ertl

Computer Graphics Group, University of Erlangen
Am Weichselgarten 9, 91058 Erlangen, Germany

Abstract. In real applications the velocity field of a flow is not available in analytical but in discrete form. One goal of this paper is to analyze particle integration methods for discretized data defined on meshes with regard to numerical efficiency and accuracy. Careful error analysis of the particle tracing process relates the error of velocity interpolation in space and time to the error of the numerical integration. Hence, a fast integration routine which provides accuracy similar to that of interpolation is necessary. This leads to a robust integration routine with adaptive step size control and error monitoring. A second aspect of this work is the treatment of stiff problems. Stiffness occurs in flows with strong shear deformations or vorticity. To detect stiffness in a given flow field, the Jacobian of the velocity field is analyzed. Implicit integration methods are used to handle stiff systems of ordinary differential equations.

1 Introduction

Flow visualization tools based upon particle methods continue to be an important topic of research. With respect to numerical integration fourth order Runge-Kutta schemes are widely used within the visualization community, some times without a detailed error analysis. Especially, the interrelation to the ubiquitous linear interpolation of grid values is rarely discussed.

When computing a particle trace, an initial value problem for an ordinary differential equation has to be solved (see Section 2). If the velocity field of the flow is given in an analytical form, integration algorithms of high order are preferable like extrapolation methods or high order Runge-Kutta schemes. However, in real applications vector fields arise that are defined on discrete grids, since these velocity fields are given by numerical simulation or by measurement. For such rough vector fields higher order algorithms like extrapolation methods are useless.

In most particle tracing modules integration methods of the Runge-Kutta type of at least order four are used, e.g. the NAG-Advect modules of the IRIS Explorer use an adaptive RK4(5) scheme. Also Stalling and Hege use an adaptive RK4(3) algorithm in their Fast-LIC module [10].

In Section 2 we establish a relation between the errors introduced by the linear interpolation of the velocity on time and space domain grids and the errors

introduced by the numerical integration. This leads to a robust integration routine with adaptive step size control of type RK3(2), which is not more accurate than necessary but significantly more efficient than the adaptive Runge-Kutta methods of higher order and the classical RK4 with fixed step size or the step doubling approach [1] or some other heuristic step sized adaption techniques [7].

In Section 3 the problem of stiffness is discussed. Mechanisms of detecting stiffness in a given flow field are analyzed and it is discussed in which data sets stiffness can be expected. Finally, implicit integration methods are suggested to cope with stiff systems of ordinary differential equations.

2 Integration Methods for Discrete Data

This section deals with the fast and accurate computation of particle traces in steady and unsteady flows. Lagrange visualization techniques of vector fields are based upon the numerical solution of an initial value problem for the following ordinary differential equation:

$$\frac{dx}{dt} = v(x, t) \quad , \quad x(t_0) = x_0 \quad (1)$$

where v denotes the velocity vector field, x the position, t the time variable and x_0 the start value at the initial time t_0 .

Usually, the numerical solution for a particle trace is given on discrete grids $x_\Delta(t_i)$ at discrete time steps $t_i \in \Delta = \{t_0, t_1, \dots, t_n\}$. The accuracy of a particle tracing algorithm is limited by the discretization error of the velocity field and the error of the numerical integration. If an integration method of order p is used, then the discretization error is characterized by

$$\|x(t_i) - x_\Delta(t_i)\| \leq C\tau^p \quad (2)$$

where C is a positive constant, τ the maximal step size in the grid Δ and $t_i \in \Delta$.

Based upon this information, it is not possible to determine a good time discretization a priori and therefore equidistant time steps are usually chosen with $t_i = t_{i-1} + \tau$. However, it cannot be expected that grids equidistant in time direction solve problems with completely different characteristics appropriately. Even within a single flow, a different behavior of the flow field in different regions of the computing domain can be expected.

A reliable and efficient algorithm should be able to construct a problem adapted grid Δ . In this way the two goals, better performance and higher accuracy, can be achieved simultaneously. Hence, we are interested in an integration method which provides the desired accuracy with the minimal possible cost, including the generation of the approximating grid. This leads to adaptive integration algorithms.

In order to select an integration scheme, single-step or multi-stage methods, and multi-step methods are considered. In a single-step method like Runge-Kutta, the position $x(t_n)$ depends only on $x(t_{n-1})$. In a multi-step method, the

results of a fixed number of previous time steps are used to compute the new position $x(t_n)$. The order of the method depends on how many previous time steps are used to calculate a new x . Multi-stage schemes are single-step methods with multiple function evaluations per time step, e.g. Runge-Kutta methods of at least order two.

Here we have to take into account that the overhead of a multi-step method for the computation of a problem adapted grid $x_\Delta(t_i)$ results in a higher cost than in the case of a single-step method. Because of this fact only multi-stage methods are considered in this text.

For adaptive integration algorithms the right-hand side of equation (1) has to be evaluated not only at arbitrary positions in space but also in time. In order to calculate the velocity v , tri-linear interpolation in space and linear interpolation in time is used. Higher order interpolation schemes in time would require to keep many complete time steps in memory. This would be a problem because of the huge data sets which usually arise from numerical simulations.

The interpolation scheme causes an approximation error of the velocity field. This error limits the maximal possible accuracy that can be reached by numerical integration of the particle trajectories. The error originating from the discretization of the velocity and the error resulting from the numerical integration are separately considered in order to estimate them.

If the discretization constants are h_t in time, the size of the time step given by the numerical simulation, and h_x in space, the size of the space discretization, then the error in the approximation of $v(x, t)$ resulting from the linear interpolation is of order $O(h_t^2)$ in time and $O(h_x^2)$ in space. In order to understand the influence of these errors on the integration, the exact trajectory generated by the velocity field $v(x, t)$ is compared with that generated by the vector field $\hat{v}(x, t)$ which is obtained by interpolation. It is assumed that both velocity fields v and \hat{v} satisfy a Lipschitz condition, i.e. there exist positive Lipschitz constants L_v and $L_{\hat{v}}$ so that $\|v(x_1, t) - v(x_2, t)\| \leq L_v \|x_1 - x_2\|$ and $\|\hat{v}(x_1, t) - \hat{v}(x_2, t)\| \leq L_{\hat{v}} \|x_1 - x_2\|$ for all x_1, x_2 and t of the domain of v respective \hat{v} . Then the trajectories are given by the equations

$$\frac{dx}{dt} = v(x, t) \quad \text{and} \quad \frac{d\hat{x}}{dt} = \hat{v}(\hat{x}, t) \quad . \quad (3)$$

If it is assumed that both trajectories have the same starting position $x(t_0) = \hat{x}(t_0) = x_0$, the difference between them at a later time t is:

$$\|x(t) - \hat{x}(t)\| = \left\| \int_0^t (v(x, s) - \hat{v}(\hat{x}, s)) ds \right\| \quad (4)$$

$$\leq (C_t h_t^2 + C_x h_x^2)t + L_{\hat{v}} \int_0^t \|x(s) - \hat{x}(s)\| ds \quad (5)$$

where C_t and C_x are constants. Now applying the Gronwall lemma (see [5]), the following estimate for the global discretization error caused by interpolating the

velocity field is obtained:

$$\|x(t) - \hat{x}(t)\| \leq \frac{C_t h_t^2 + C_x h_x^2}{L_{\hat{v}}} (e^{L_{\hat{v}} t} - 1) \quad . \quad (6)$$

An estimation for the global error of the numerical integration of the exact velocity field v is given by

$$\|x(t) - x_{\Delta}(t)\| \leq \frac{C\tau^p}{L_{\Delta}} (e^{L_{\Delta} t} - 1) \quad (7)$$

where L_{Δ} is the Lipschitz constant of the integration scheme (compare [4, 5]).

If the constants C and L_{Δ} are comparable to $\max(C_t, C_x)$ and $L_{\hat{v}}$ respectively, and if the integration step τ is comparable to the time discretization step h_t and smaller than the cell size ($\|v\|\tau \leq h_x$), then equations (6) and (7) show that an integration scheme of order $p > 2$ should produce an error which is negligible compared to the discretization error. This means that an integration method of order $p \geq 2$ should be accurate enough with regard to the discretization error. To verify this assumption, we have implemented standard and extrapolated Runge-Kutta algorithms up to order 6 and extrapolated midpoint rules up to order 8, and tested these algorithms in several applications (compare Section 5). Furthermore, we have implemented embedded Runge-Kutta methods with error monitoring and adaptive step size control (see Section 4) to create a problem adapted grid Δ . Because of the adapted grid, these algorithms have a better performance and a higher accuracy than the simple Runge-Kutta methods.

3 Stiff Sets of Ordinary Differential Equations

In many practical applications the ordinary differential equations are *stiff*, for instance equations describing chemical reactions or the equations resulting from the semi-discretization of parabolic equations. Stiff systems were observed for the first time in 1952 by Curtis and Hirschfelder, when solving the problem of chemical reaction of a multicomponent system using explicit Runge-Kutta method. In this chemical experiment some components react in a very short time achieving the equilibrium state while the other components change very slowly.

It is important to distinguish between the stability of the ordinary differential equation and the stability of the integration method. In the case of the chemical reaction, explicit Runge-Kutta methods fail even for very small integration steps. On the other hand, the stability properties of the system are very good, i.e. the explicit Runge-Kutta integration method is unstable for stiff systems.

Stiff systems are in some sense characterized by the Jacobian $D_x v$ of v . The concept of stiffness can be mathematically defined but we concentrate on a characterization which is valuable in practical applications. We say that a system of ordinary differential equations is stiff in the *time* interval (t_0, t) if

$$(t - t_0) \|D_x v\| \gg 1 \quad \text{or equivalent} \quad (t - t_0)L \gg 1 \quad (8)$$

where L is the Lipschitz constant of v . If we use the \mathcal{L}^2 matrix norm, then $\|D_x v\|$ is equal to the absolute value of the largest eigenvalue of the matrix. In this section we consider fluid flow data and analyze under which circumstances the particle tracing equation (1) may show stiffness.

If a system of ordinary equations is stiff in some regions, then the corresponding linearized problem inherits this stiffness (compare [5]). Thus, only the linearized problem is considered. Furthermore, because the time dependent part is linear in t , it is not relevant for the considerations in this section. Thus, we only analyze the part of the ordinary differential equation that contains the Jacobian of the velocity:

$$\frac{dx}{dt} = D_x v \cdot x \quad . \quad (9)$$

In order to determine whether particle tracing for a given flow is stiff or not, the eigenvalues of the Jacobian of the velocity field have to be analyzed. The Jacobian can be decomposed into three parts: $D_x v = \Sigma + \Theta Id + \Omega$ where Σ represents the shear of the fluid flow and is a symmetric matrix, Θ is the expansion, Id the identity matrix, and Ω the rotation, which is an skew-symmetric matrix. The rotation matrix is of the form

$$\Omega = \frac{1}{2} \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix} \quad (10)$$

where the vector ω is the rotation of the velocity, i.e. $\omega = \nabla \times v$.

If the shear of the flow field is large, e.g. near walls or within turbulence, the eigenvalues of Σ which correspond to the shear directions are large. It follows that the norm of the Jacobian is large and satisfies condition (8). In such situations it is recommended to switch to a special integration method, called implicit. We are coming back to this problem in the next section.

The norm of the matrix Ω can be easily computed due to its special form:

$$\|\Omega\| = \frac{1}{2}|\omega| = \frac{1}{2}\sqrt{\omega_1^2 + \omega_2^2 + \omega_3^2} \quad (11)$$

which is the vorticity of the flow. A typical example, where vorticity is relevant, is a circular flow. In this case particle tracing becomes a stiff problem. Additionally, such a circular flow shows also strong shear deformations.

4 Multi-Stage Methods

Due to the disadvantages of multi-step algorithms mentioned above, only multi-stage or single-step methods are considered. A detailed description of integration methods is found in [5, 9].

The well-known Runge-Kutta schemes (RK p) are the most famous explicit multi-stage algorithms. The letter p denotes the order of the integration scheme. The simplest one is Euler's method (RK1) and an example of RK2 is Heun's method. A different class of single-step algorithms are the extrapolation schemes,

for instance the extrapolation of Euler's method (RK1X p) or the extrapolation of the midpoint rule (MPX p). The order of RK1X p rises by one after each extrapolation step and the order of MPX p increases by two after each step.

An improvement of explicit single-step algorithms are adaptive explicit single-step methods. The general idea of adaptive algorithms is to compute two trajectories with different integration schemes for each time step. If the difference between the endpoints of these traces is larger than a given tolerance, the computation is repeated with a smaller integration step size.

As adaptive schemes, embedded Runge-Kutta schemes are well suited. Here, the different integration methods are Runge-Kutta algorithms of different order and *embedded* means that the coefficients for the evaluation of the velocity field are all equal up to the lower order. Embedded Runge-Kutta methods are denoted RK $p(q)$ where p is the integration order and q is the order for the error estimation. We have implemented RK2(1), RK3(2) and RK4(3). These algorithms are represented with the help of the Butcher schemes, where the coefficients of the method can easily be seen (compare [2]):

$$\begin{array}{c|cc}
 0 & & \\
 1 & 1 & \\
 \hline
 & 1/2 & 1/2 \\
 & 1 & 0
 \end{array}
 \qquad
 \begin{array}{c|ccc}
 0 & & & \\
 1 & 1 & & \\
 \hline
 1/2 & 1/4 & 1/4 & \\
 & 1/6 & 1/6 & 2/3 \\
 \hline
 & 1/2 & 1/2 & 0
 \end{array}
 \qquad
 \begin{array}{c|cccc}
 0 & & & & \\
 1/2 & 1/2 & & & \\
 1/2 & 0 & 1/2 & & \\
 1 & 0 & 0 & 1 & \\
 \hline
 1 & 1/6 & 1/3 & 1/3 & 1/6 \\
 & 1/6 & 1/3 & 1/3 & 1/6 & 0 \\
 \hline
 & 1/6 & 1/3 & 1/3 & 0 & 1/6
 \end{array}$$

Adaptive Runge-Kutta algorithms RK2(1), RK3(2) and RK4(3).

Now we come back to stiff sets of ordinary differential equations. For a proper integration of such equations, linear implicit multi-stage methods are introduced. To see how implicit algorithms work, we consider equation (1) and write down the explicit Euler scheme for integrating this equation with step size τ :

$$x(t_n) = x(t_{n-1}) + \tau v(x(t_{n-1}), t_{n-1}) \quad . \quad (12)$$

The method is called explicit because the new value $x(t_n)$ is given explicitly in terms of the old value $x(t_{n-1})$. Whereas, the implicit Euler scheme is:

$$x(t_n) = x(t_{n-1}) + \tau v(x(t_n), t_n) \quad . \quad (13)$$

In general this is some nasty set of nonlinear equations that has to be solved iteratively at each step. In order to make implementation easier and to improve performance, the problem can be linearized. This leads to the linear implicit Euler method:

$$x(t_n) = x(t_{n-1}) + \tau (Id - \tau D_x v(x(t_{n-1})))^{-1} \cdot v(x(t_{n-1}), t_{n-1}) \quad (14)$$

where $D_x v$ denotes the Jacobian of the vector field v at the point $x(t_{n-1})$. Since at each time step a matrix has to be inverted, linear implicit methods are slower

than explicit ones. Therefore, linear implicit algorithms should be used only if the system of differential equations is stiff. Also remark that a linear implicit scheme converges to its corresponding explicit scheme for $\tau \rightarrow 0$.

We have implemented the linear implicit Euler method (LIRK1), the linear implicit midpoint rule (LIMP) and extrapolation schemes of both algorithms, abbreviated LIRK1X p and LIMPX p respectively. Again the order of LIMPX p increases by two after each extrapolation step.

An improvement of implicit algorithms are again adaptive linear implicit multi-stage methods. Kaps and Rentrop showed in [6] that the smallest order for which embedded adaptive linear implicit Runge-Kutta methods are possible is order 4(3). We have implemented such an integration scheme and abbreviated it LIRK4(3).

We have implemented all integration algorithms mentioned in this section within the framework of the IRIS Explorer visualization environment.

5 Comparison of Integration Methods

In many applications vector fields arise that are very rough, so that higher order algorithms like extrapolation methods do not work well. Furthermore, due to the fact that the vector fields are strongly varying, the integration steps have to be very short. Thus, extrapolation schemes become very expensive. The explicit Euler method does not work well either. The first order integration method is not able to follow particles properly in turbulent flows.

Therefore we compare the performance of adaptive and non-adaptive Runge-Kutta methods only. The efficiency of the algorithms is measured by the number of evaluations of the velocity field. Also the number of calculated particles is listed in the following table. The evaluation of the velocity field corresponds to the point location and velocity interpolation steps of the particle tracing algorithm, which are the most time consuming operations [7]. For the test three data sets were used with different characteristics. The first one corresponds to the simulation of a vortex breakdown where the gradients of the velocity field are strongly varying. The second data set corresponds to the simulation of the air flow in a clean air room. The third data set is a laminar cross-cylinder flow. The velocity field of this data set is very smooth.

		RK2	RK3	RK4	RK2(1)	RK3(2)	RK4(3)
vortex	integr. steps	3694	1849	1233	691	239	166
	v -evaluations	7387	5545	4929	2582	1957	2211
air flow	integr. steps	5000	4147	3290	146	53	46
	v -evaluations	9999	12439	13157	636	225	235
laminar flow	integr. steps	291	146	97	94	34	29
	v -evaluations	581	436	387	346	180	189

In these tests the step sizes of all methods and the tolerances of the adaptive algorithms are chosen so that the resulting trajectories look identical and so that scaling down the step size of an integration scheme would not change its

resulting trace. The speed differences are relative small in laminar flows, since all algorithms can integrate with a large step size. The speedup of adaptive methods is much greater in turbulent flow fields, because the non-adaptive methods have to use a small step size everywhere.

Furthermore, the tests confirm our consideration that RK3(2) is faster than RK4(3), if the integration is not more accurate than necessary. On the other hand RK2(1) is a lot slower than RK3(2). The scheme RK2(1) uses RK2 and Euler’s method to determine the step size. Usually, the difference between the trajectory of the Euler scheme and that of the RK2 algorithm is relative large, and thus a relative small step size is chosen (compare Section 4). Hence, the RK2(1) method is significantly slower than the RK3(2) algorithm.

Now we compare the adaptive Runge-Kutta methods of our IRIS Explorer particle tracing module with the NAG-Advect-Simple module that uses an RK4(5) algorithm. The vortex data set is used to compare the computational time of the algorithms. The computational time of the NAG-Advect module is set to 100%.

NAG-Advect	RK4(3)	RK3(2)	RK2(1)
100%	86%	82%	109%

Relating to stiff problems, two example data sets where stiffness occurs are shown and described in Figure 1 and 2. These two figures are also displayed as colored images in the Appendix.

Figure 1 shows a flow in a floating-zone furnace for crystal growing. Because of the rotational symmetry, the stream bands should be closed. On the left hand side the stream bands are computed by the linear implicit Euler scheme and on the right hand side the classical Runge-Kutta method of order four is used. In this example both algorithms use the same step size and the same number of steps. On the left hand side the bands are closed but on the other side they are not. This is a typical data set where explicit methods fail and implicit algorithms have an advantage. The black planes define the starting positions for the particles.

Figure 2 shows a test data set of a cylindrical air flow caused by different temperatures on the walls of the box. Because of symmetry all trajectories should be closed. Here we have used both explicit and linear implicit RK4(3). At first glance both methods close their particle traces but after some 50 circulations the trajectories of the explicit Runge-Kutta scheme become thick (grey circles), whereas the traces of the implicit algorithm remain closed (black circles).

In both examples the particle tracing using the implicit algorithm took about twice as long as that using the explicit equivalent. Linear implicit integration methods are slower than their explicit counterparts but in stiff cases they are very helpful.

6 Conclusion

As a result of careful analysis of numerical efficiency and accuracy of different integration methods on discrete data, it is shown that an adaptive RK3(2)

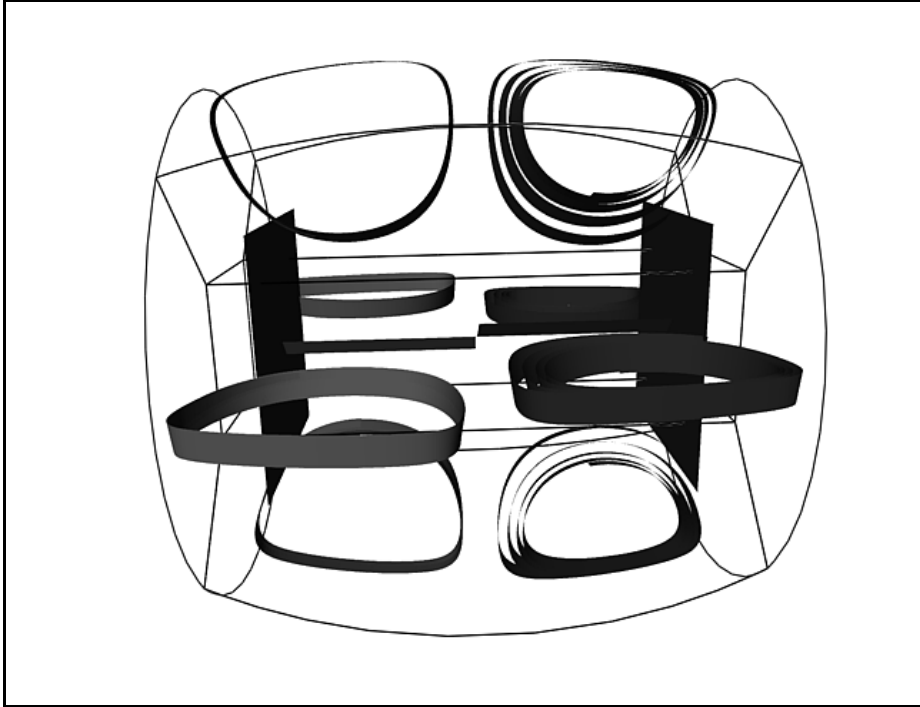


Fig. 1. Flow in a floating-zone furnace for crystal growing. On the left hand side the stream bands are computed by the linear implicit Euler scheme and on the right hand side the classical Runge-Kutta method of order four is used.

scheme is accurate enough in relation to the interpolation error and significantly more efficient than higher order integration algorithms. Furthermore, we have implemented implicit integration methods to handle stiff systems of ordinary differential equations. These algorithms are slower than explicit methods but in stiff data sets when explicit methods fail, they can create proper trajectories.

Based upon these results we have implemented a particle tracer as a module within the IRIS Explorer visualization environment.

Due to the fact that implicit integration is very expensive and that, in general, systems are stiff only in some localized regions, we intend to implement *partitioned* methods as a future extension. These are adaptive methods with a mechanism to switch between explicit and implicit integration depending on the eigenvalues of the Jacobian at the current point. If $\tau L \leq C$, the explicit integration is used else the implicit one. Here C is a method dependent constant (compare [3]) and L is the Lipschitz constant of the velocity field. If the currently used method is the implicit one, L can be approximated by the norm of the Jacobian (see Section 3) else L can be estimated by intermediate values of

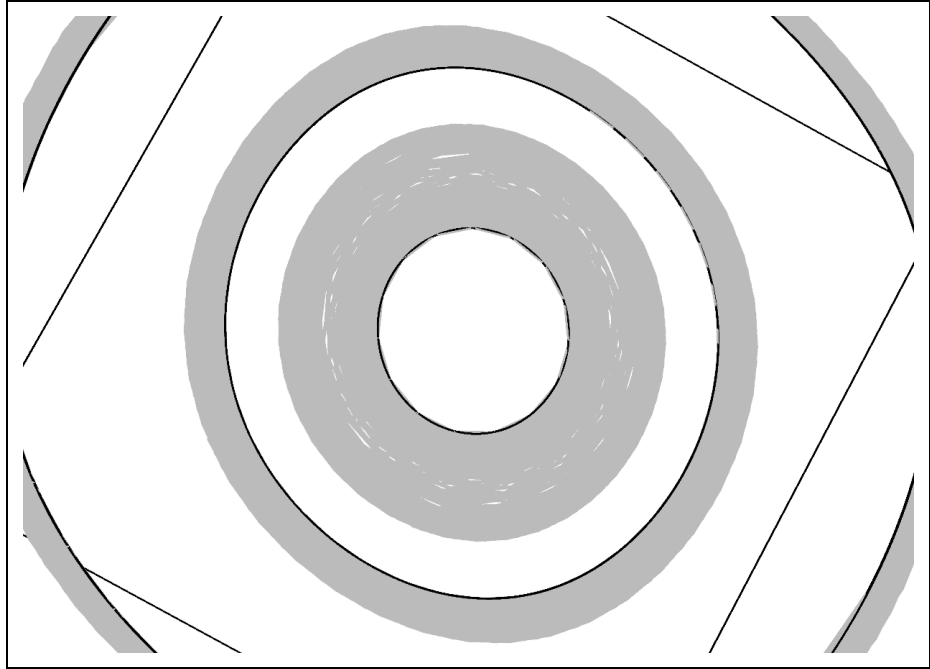


Fig. 2. Test data set of an cylindrical air flow caused by different temperatures on the walls of the box. Here we have used both explicit and linear implicit RK4(3). The trajectories of the implicit Runge-Kutta scheme remain closed (black circles), whereas the traces of the explicit method become thick (grey circles).

the explicit multi-stage method in the following way:

$$L \geq \frac{\|v(x^{(i)}(t_n)) - v(x^{(j)}(t_n))\|}{\|x^{(i)}(t_n) - x^{(j)}(t_n)\|} . \quad (15)$$

7 Acknowledgments

We are grateful to H. Weimann from the Lehrstuhl für Werkstoffwissenschaften VI of the University of Erlangen for making the data set of a floating-zone furnace for crystal growing available to us. Also we would like to thank M. Breuer, S. Enger and K. Wechsler from the Lehrstuhl für Strömungsmechanik of the University of Erlangen for the other data sets and for their valuable remarks. Finally, we wish to thank the anonymous reviewers of this paper for their helpful comments.

References

1. B. Becker, D. A. Lane, and N. L. Max. Unsteady Flow Volumes. In G.M. Nielson and Silver D., editors, *Visualization '95*, pages 329–335, Los Alamitos, CA, 1995. IEEE Computer Society, IEEE Computer Society Press.
2. J. C. Butcher. Coefficients for the study of Runge-Kutta integration processes. *J. Austral. Math. Soc.*, 3:185–201, 1963.
3. J. C. Butcher. Order, stepsize and stiffness switching. *Computing*, 44:209–220, 1990.
4. D. L. Darmofal and R. Haimes. An Analysis of 3-D Particle Path Integration Algorithms. In *Proceedings of the 1995 AIAA CFD Meeting*, 1995.
5. P. Deuffhard and F. Bornemann. *Numerische Mathematik II: Integration gewöhnlicher Differentialgleichungen*. Walter de Gruyter, Berlin, New York, 1994.
6. P. Kaps and P. Rentrop. *Numerische Mathematik*, 33:55–68, 1979.
7. D. N. Kenwright and D. A. Lane. Optimization of Time-Dependent Particle Tracing Using Tetrahedral decomposition. In G. M. Nielson and Silver D., editors, *Visualization '95*, pages 321–328, Los Alamitos, CA, 1995. IEEE Computer Society, IEEE Computer Society Press.
8. D. N. Kenwright and G. D. Mallinson. A 3-D Streamline Tracking Algorithm Using Dual Stream Functions. In A. E. Kaufman and G. M. Nielson, editors, *Visualization '92*, pages 62–68, Los Alamitos, CA, October 1992. IEEE Computer Society, IEEE Computer Society Press.
9. William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, New York, Victoria, second edition, 1992.
10. D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *Computer Graphics Proceedings*, Annual Conference Series, pages 249–256, Los Angeles, California, August 1995. ACM SIGGRAPH, Addison-Wesley Publishing Company, Inc.