An Analysis of 3-D Particle Path Integration Algorithms

D.L. Darmofal
Research Fellow
Aerospace Engineering
U. of Michigan
Ann Arbor, MI 48109

R. Haimes
Principal Research Engineer
Dept. of Aeronautics & Astronautics
MIT
Cambridge, MA 02139

Abstract

Several techniques for the numerical integration of particle paths in steady and unsteady vector (velocity) fields are analyzed. Most of the analysis applies to unsteady vector fields, however, some results apply to steady vector field integration. Multi-step, multi-stage, and some hybrid schemes are considered. It is shown that due to initialization errors, many unsteady particle path integration schemes are limited to third order accuracy in time. Multi-stage schemes require at least three times more internal data storage than multi-step schemes of equal order. However, for timesteps within the stability bounds, multi-stage schemes are generally more accurate. A linearized analysis shows that the stability of these integration algorithms are determined by the eigenvalues of the local velocity tensor. Thus, the accuracy and stability of the methods are interpreted with concepts typically used in critical point theory. This paper shows how integration schemes can lead to erroneous classification of critical points when the timestep is finite and fixed. For steady velocity fields, we demonstrate that timesteps outside of the relative stability region can lead to similar integration errors. From this analysis, guidelines for accurate timestep sizing are suggested for both steady and unsteady flows. In particular, using simulation data for the unsteady flow around a tapered cylinder, we show that accurate particle path integration requires timesteps which are at most on the order of the physical timescale of the flow.

I. Introduction

In this paper, we analyze several numerical particle path integration schemes for three-dimensional, unsteady data. Our discussions are limited to particles which follow the local vector field (i.e., massless), however, these algorithms can be extended to include forces acting on particles with mass. The problem which we wish to solve is

$$\frac{d\vec{x}}{dt} = \vec{u}(\vec{x}, t),$$

where $\vec{x}$ is the particle location, $\vec{u}$ is the particle velocity, and t is the current time. In our particular application, the particle velocity data is generated from a numerical simulation; therefore, the velocity data is only available at some set of discrete times, t^n, for $n = 0, 1, 2, \ldots, N$. The particle paths may be calculated in a post-processing or concurrent (co-processing) manner. For post-processing, the time planes of velocity data are stored at some set number of iterations typically determined by the amount of disk storage available. For concurrent processing, the vector field can be updated every iteration of the flow algorithm or after some set number of iterations. In either case, the velocity field, $\vec{u}$, is only available at some set of discrete instances in time. A consequence of the temporally discrete nature of the velocity data is that the timestep cannot be set by the integration algorithm. This is unlike instantaneous streamline integration (or steady velocity fields) for which the timestep can be adaptively varied to account for rapid trajectory changes thereby increasing accuracy. The goal of this paper is to determine the factors which impact the accuracy, efficiency, and memory requirements of particle integration schemes. Although the majority of this paper focuses on unsteady data integration, some discussion of the steady data or streamline integration case is also given.

Most related work is concerned with integration of streamlines in steady flows[1, 2, 3, 4, 5]. However, some work on unsteady flows has also been done. Darmofal & Haimes [6] suggest a particular algorithm for higher order accurate particle path calculations. Also, Shirayama[7] has discussed the effects of local truncation errors on a particular class of two-dimensional linear velocity fields. However, unlike this work, Shirayama only considered schemes of second order temporal accuracy.

II. General considerations

Since the timestep is generally not controllable by the particle integration algorithm in unsteady flows, a major portion of this analysis focuses on quantifying the effects of finite timestep size on the integration accuracy. Also, the particle path integration schemes may be subject to numerical instabilities because of large timesteps. Thus, it may be necessary to use an implicit integration scheme to obtain reasonable (although not necessarily accurate) trajectories for a wide range of flow fields and timestep combinations.

In addition, each time plane of velocity data from numerical simulations can generally be assumed to require large amounts of computer memory for storage. Thus, algorithms which need several time planes of velocity data can be extremely memory intensive either using large amounts of internal memory (limiting the size of the problem) or frequently reading from disk (impacting the computational efficiency).

The velocity data is usually only available at discrete points in space. Thus, the velocity field must be spatially interpolated from nodal values to the desired spatial location. However, in this study, we only address the time accuracy of the particle path integrations and our model problems have analytic velocity distributions in space. For spatially discrete velocity fields, we use a trilinear interpolation scheme. The reader may consult [6] for details. Similar interpolation schemes are also described by Shirayama[7].

Finally, current computational algorithms (used in Computational Fluid Dynamics) are often second order accurate in time. If we are using these integration schemes to either debug solvers or to gain greater insight into the flow physics about complex geometries, a reasonable constraint on the particle integration accuracy is that it should not introduce any additional errors. Therefore, we wish to construct particle integration schemes of at least third order so that the error is smaller than the flow simulation error in the limit as the timestep approaches zero.

III. Algorithm descriptions

The algorithms we consider may be divided into multi-step and multi-stage schemes. In either case, we assume that the velocity field is only available at equally spaced time intervals such that $t^n = nk$ where k is the timestep and n is the current iteration index. Almost every algorithm we discuss is easily extendible to non-constant timesteps. One typical example of a non-constant timestep algorithm is given in Appendix B. Furthermore, we assume that the velocity field is defined for any position $\vec{x}$.

We also use the concepts of local and global accuracy. We write a generic integration scheme for timestep k as

$$\vec{x}^{n+1} = S_k\left(\vec{x}^n, \vec{u}^n\right),$$

where S_k may include timeshift operators. The local error, ϵ_L, and the global error, ϵ_G, may be defined by

$$\begin{aligned} \epsilon_L &= ||\vec{x}_e^{n+1} - S_k\left(\vec{x}_e^n, \vec{u}_e^n\right)||, & (1) \\ \epsilon_G &= ||\vec{x}(t) - \vec{x}_e(t)||, & (2) \end{aligned}$$

where $\vec{x}_e$ and $\vec{u}_e$ are the exact particle position and velocity. The local error is the error made at a single timestep while the global error measures the cumulative effects of errors from every timestep including any start-up procedures. The local order of accuracy, p, is defined as

$$\epsilon_L = O(k^{p+1}), \quad \text{as } k \to 0.$$

The global order of accuracy, r, is defined as

$$\epsilon_G = O(k^r), \quad \text{as } k \to 0 \text{ for fixed } t = T_f.$$

Ignoring the start-up effects, a scheme with local error $O(k^{p+1})$ gives a global error of order p because $O(k^{p+1})$ local errors are made for T_f/k iterations. However, if the local error of the start-up procedure is $O(k^i)$, then the global integration error for the scheme is $O(k^r)$ where $r = \min(p, i)$ [8]. As we show, the start-up error plays an important role in the design of particle path integration schemes.

A. Multi-step schemes

A generic multi-step scheme has the form

$$\sum_{i=0}^{s} \alpha_i\, \vec{x}^{n+1-i} = k \sum_{i=0}^{s} \beta_i\, \vec{u}(\vec{x}^{n+1-i}, t^{n+1-i}),$$

where s is the total number of steps, $\alpha_0 = 1$, and either α_{n+1-s} or β_{n+1-s} is non-zero. For implicit schemes, $\beta_0 \neq 0$ and a (typically) nonlinear system of three equations must be solved at every time step for the new particle position $\vec{x}^{n+1}$. We solve these nonlinear equations using a Newton-Raphson technique. Although explicit schemes are simpler to implement than implicit schemes, they are more severely limited by stability restrictions. Coefficients for several multi-step schemes are given in Appendix A.

An advantage of multi-step schemes is that only one velocity field, $\vec{u}^{n+1}$, is needed to calculate the new particle position, $\vec{x}^{n+1}$. The previous particle position and velocity data from n to $n+1-s$ are also needed, however these are only point data (i.e. not field data); for a three-dimensional flow, this adds a maximum of $6s$ additional words of storage per particle path. The storage of this data is generally done in the computer's internal memory for each step of the integration, therefore, we refer to this storage as the internal memory usage. If this data were stored on disk, then the internal memory usage measures the amount of I/O required between disk and the CPU per timestep. For an s-step multi-step scheme, the total internal memory usage is bounded by

$$\text{multi-step memory} \leq N_f + 6sN_p \text{ words}$$

where N_f is the number of words necessary to store the velocity field and N_p is the number of particle trajectories. As mentioned previously, the storage requirements for a time plane of data are considerable; thus, $N_f >> N_p$ and the storage for a multi-step scheme is approximately the storage of a single velocity field.

Higher order accurate multi-step schemes with $s > 1$ require the generation of start-up data for the first $s-1$ iterations. Although the **local** truncation error of an integration scheme after initialization may be of higher order, the presence of start-up errors places a limit on the **global** accuracy of many particle path integration schemes. Typical approaches for generating this initialization data are to use other multi-step schemes with smaller timesteps, or to use a multi-stage scheme. Unfortunately, for this problem, the timesteps are not controllable since they are set by the data generation algorithm. Furthermore, the first iteration can only be a 1-step scheme since previous particle position or velocity data is generally not available. Thus, the local accuracy of the start-up scheme is limited by the highest local accuracy achievable using a 1-step scheme. For stable 1-step schemes, Dahlquist's first stability barrier states that $i \leq 3$. For example, trapezoidal integration (1-step Adams-Moulton) has a local truncation error which is $O(k^3)$. Thus, since the global accuracy of a scheme is $min(p, i)$ and $i \leq 3$, the global error for many algorithms is $O(k^3)$. As we discuss in Section B, a similar situation arises using multi-stage schemes for start-up such that the best global truncation error achievable is again $O(k^3)$.

This start-up error analysis suggests that the best one could hope for is an integration scheme which has a third order global error. However, in some circumstances, it should be possible to improve the start-up error by altering the predicted particle positions after additional timeplanes of data become available. One straightforward technique for doing this would be to simply delay the integration of the particle position by a single iteration. Then, the first update of the particle position would have three data planes available and a higher order accurate, 3-step scheme could be used increasing the start-up accuracy to $i = 4$. Whether or not this integration delay is acceptable will depend on the particular application.

B. Multi-stage schemes

Multi-stage schemes are difficult to write in a single, unified form, therefore, we concentrate on two specific examples from the Runge-Kutta family of multi-stage schemes [9]. First, the 2-stage Runge-Kutta algorithm often called the Heun method is written:

$$\begin{aligned} \vec{a} &:= k\,\vec{u}(\vec{x}^n, t^n), \\ \vec{b} &:= k\,\vec{u}(\vec{x}^n + \vec{a}, t^n + k), \\ \vec{x}^{n+1} &:= \vec{x}^n + \frac{1}{2}(\vec{a} + \vec{b}). \end{aligned} \tag{3}$$

The second scheme is the classic 4-stage Runge-Kutta algorithm which can be written:

$$\begin{aligned} \vec{a} &:= k\,\vec{u}(\vec{x}^n, t^n), \\ \vec{b} &:= k\,\vec{u}(\vec{x}^n + \frac{1}{2}\vec{a}, t^n + \frac{1}{2}k), \\ \vec{c} &:= k\,\vec{u}(\vec{x}^n + \frac{1}{2}\vec{b}, t^n + \frac{1}{2}k), \\ \vec{d} &:= k\,\vec{u}(\vec{x}^n + \vec{c}, t^n + k), \\ \vec{x}^{n+1} &:= \vec{x}^n + \frac{1}{6}(\vec{a} + 2\vec{b} + 2\vec{c} + \vec{d}). \end{aligned} \tag{4}$$

We denote the algorithms given by Equations (3) and (4) as RK2 and RK4, respectively. The local truncation error for RK2 is $p = 2$ and for RK4 is $p = 4$.

A difficulty with multi-stage schemes is that they frequently require velocity data at intermediate times between t^n and t^{n+1}. For example, RK4 requires velocity data at the midpoint (i.e., $t^n + \frac{1}{2}k$). Since velocity data is only available at t^n, the velocity at intermediate times must be interpolated from the previous or the current velocity fields. If the interpolant introduces an error which is $O(k^q)$ at the required quadrature point, then the local accuracy of the multi-stage scheme is $p = \min(q, p_e)$ where p_e is the local truncation error of the multi-stage scheme with an exact interpolant. Therefore, in order to maintain fourth order accuracy in the RK4 scheme, the interpolant must be fourth order ($q = 4$) at the half timestep. For equally-spaced time intervals, the desired fourth order interpolant is

$$\begin{aligned} \vec{u}^{n+\frac{1}{2}}(\vec{x}) &= \frac{5}{16}\vec{u}^{n+1}(\vec{x}) + \frac{15}{16}\vec{u}^{n}(\vec{x}) \\ &- \frac{5}{16}\vec{u}^{n-1}(\vec{x}) + \frac{1}{16}\vec{u}^{n-2}(\vec{x}). \end{aligned}$$

While the velocity is being interpolated in time, the spatial position remains fixed at the desired $\vec{x}$ location. For example, in the second step of the RK4 scheme, the desired location is $\vec{x} = \vec{x}^n + \frac{1}{2}\vec{a}$. The resulting four-stage Runge-Kutta scheme with the fourth order interpolant is denoted as RK44. This scheme can be thought of as a hybrid between a multi-stage and a multi-step scheme since the base scheme is multi-stage while the interpolant is essentially a multi-step approximation.

To illustrate the loss of accuracy which occurs when a lower order accurate interpolant is used in conjunction with the RK4 scheme, we consider a linear interpolant between t^n and t^{n+1},

$$\vec{u}^{n+\frac{1}{2}}(\vec{x}) = \frac{1}{2}\vec{u}^{n+1}(\vec{x}) + \frac{1}{2}\vec{u}^{n}(\vec{x}).$$

The resulting scheme would have a local accuracy of $p = \min(2, 4) = 2$. We denote this four-stage scheme with linear interpolation as RK42. A simple example which shows the loss of accuracy of the RK42 scheme is a velocity field which only depends on time $\vec{u} = \vec{u}(t)$. Thus, using a linear interpolant with Equation (4) and assuming $\vec{u} = \vec{u}(t)$, the four stage scheme is identical to the two-stage method given in Equation (3). Therefore, the RK42 scheme is not fourth order but rather is second order.

The internal memory usage of the higher order, hybrid schemes can be significant because q planes of velocity field data are needed to construct the interpolant. The entire field of velocity data must be stored because the particle position at the intermediate times is unknown. Thus, it must be possible to calculate the velocity at any spatial location. The total internal memory usage for a three-dimensional hybrid multi-stage scheme is

$$\text{multi-stage memory } \le q\,N_f + 3N_p \text{ words.}$$

Thus, the ratio of multi-stage to multi-step internal memory usage is

$$\frac{\text{multi-stage memory}}{\text{multi-step memory}} = \frac{q\,N_f + 3N_p}{N_f + 6s\,N_p} \approx q.$$

Since the storage for the field information is generally much greater than the number of particles, this ratio is approximately q. Therefore, on memory considerations alone, higher-order multi-step schemes are much more efficient than the higher-order multi-stage schemes.

Unlike the classic RK4 scheme, the hybrid RK44 scheme must now have special start-up procedures for the initial interpolant. At the first step of the algorithm, only two velocity fields are generally available, $\vec{u}^0$ and $\vec{u}^1$. If previous velocity fields are available, (say for example the particle trace initiation is delayed for q iterations of the flow solver), then it is possible to construct a higher order interpolant even at the first timestep. However, in the situation where the particle is released without any delay, the interpolant in the first step can be only a linear interpolant which has a $q = 2$ error. Therefore, we are faced with the same accuracy barrier as in the multi-step schemes. Namely, the start-up scheme has a local error which is at best $O(k^3)$, so the global error is at best $O(k^3)$.

A final possible higher order multi-stage scheme is the RK4 scheme applied to timesteps of value $2k$. This scheme is written:

$$\begin{aligned}
\vec{a} &:= 2k\,\vec{u}(\vec{x}^n, t^n), \\
\vec{b} &:= 2k\,\vec{u}(\vec{x}^n + \frac{1}{2}\vec{a}, t^n + k), \\
\vec{c} &:= 2k\,\vec{u}(\vec{x}^n + \frac{1}{2}\vec{b}, t^n + k), \\
\vec{d} &:= 2k\,\vec{u}(\vec{x}^n + c, t^n + 2k), \\
\vec{x}^{n+2} &:= \vec{x}^n + \frac{1}{6}(\vec{a} + 2\vec{b} + 2\vec{c} + \vec{d}),
\end{aligned}$$

which we denote the RK4 2x scheme. The particle position is updated only every other timestep in this scheme. The RK4 2x scheme does not require any interpolant for the intermediate values because they are located at t^{n+1}. Also, the scheme does not require any start-up information and has a global accuracy which is fourth order, $O((2k)^4)$ specifically. In comparison to the RK4 scheme, the memory usage is also reduced to $3N_f + 3N_p$ words which is still about three times a multi-step scheme memory usage. However, the RK4 2x scheme has two significant disadvantages. First, the stability limit is halved because of the timestep doubling. We discuss this further in Section IV. Second, unlike the previously discussed multi-step and multi-stage schemes, the RK4 2x scheme is not easily extendible to non-constant timesteps.

Though spatial interpolation and accuracy is not addressed in this paper, an additional complication arises when using multi-stage schemes when the underlying grid changes in time. In this case, the previous grid information (i.e. spatial locations of nodes) is also necessary to interpolate the velocity field at the desired intermediate time. Thus, the memory requirements double assuming the grid information storage is the same order as the velocity field storage. Also, if the particular spatial location does not exist at a previous time needed to form the time interpolant (e.g. the location was previously in the interior of a moving object), the time interpolant cannot be constructed and the particle integration aborts. With a multi-step algorithm, the particle integration aborts only when the particle actually encounters a domain boundary at the current time.

Although the RK4 schemes we have considered are explicit, they still require four velocity evaluations per timestep. In our visualization applications for unstructured grids[5], the process of finding a spatial location within the grid and then constructing the velocity interpolant is typically the most significant portion of the computational effort per timestep. By comparison, the implicit multi-step schemes, such as BD4, require a velocity evaluation for every sub-iteration of the Newton-Raphson solution process. Our experience with the BD4 scheme has shown that typically 2-4 sub-iterations are required, thus, the number of velocity evaluations for BD4 is usually less than or equal to the number of velocity evaluations for the RK4 schemes. As a result, no significant difference in computational effort exists between the RK4 and BD4 schemes.

IV. Analysis of linearized problem

In this section, we consider a linearized velocity field, such that

$$\vec{u}(\vec{x}, t) = \vec{u}(0,0) + \frac{\partial \vec{u}}{\partial t}t + \frac{\partial \vec{u}}{\partial \vec{x}}\vec{x}.$$

If we travel with velocity, $\vec{u}(0,0) + (\partial\vec{u}/\partial t)t$, the nature of the flow around the current location is determined by the velocity tensor, $\partial\vec{u}/\partial\vec{x}$. In particular, the eigenvalues of the velocity tensor, $\lambda_{1,2,3}$, are the fundamental quantities which determine the qualitative features of the flow pattern. The study of the eigenvalues of the velocity tensor is a well-researched topic in the classification of critical point flow [10, 11, 12]. The various possible flow patterns and the corresponding eigenvalue-based flow classifications are summarized in Figure 1.

The velocity tensor eigenvalues also play a significant role in determining the stability and accuracy of the numerical integration schemes. For the linear problem, a second order scheme can integrate exactly the particular solution due to the $\vec{u}(0,0) + (\partial\vec{u}/\partial t)t$ terms. Since all of the schemes we analyze are at least second order, we do not consider the non-homogeneous term. The integration of the homogeneous portion due to $\partial\vec{u}/\partial\vec{x}$ produces errors and it is these errors which we quantify further. Neglecting the non-homogeneous terms, the linearized problem becomes,

$$\frac{d\vec{x}}{dt} = \frac{\partial \vec{u}}{\partial \vec{x}}\vec{x}. \tag{5}$$

This coupled system of three equations can be decoupled into the respective eigenmodes,

$$\frac{d\tilde{x}_i}{dt} = \lambda_i \tilde{x}_i, \tag{6}$$

where $\tilde{x}_i$ is the amplitude of an eigenmode. The numerical scheme is applied to Equation (5), however, it is straightforward to show that each eigenmode behaves as if the scheme were applied to Equation (6) directly. We write the update scheme as

$$\tilde{x}_i^{n+1} = g(\lambda_i k)\, \tilde{x}_i^n,$$

where $g(\lambda_i k)$ is the scheme dependent amplification or growth factor. Higher order multi-step schemes generate parasitic roots, where more than one growth factor exists for a given $\lambda_i k$. These parasitic solutions are a result of the numerical approximation and do not track the equation being solved. When multiple growth factors exist, we limit ourselves to the growth factor with the largest magnitude, since as time goes to infinity, this root dominates all others. For the exact answer, the amplification factor, $g_e(\lambda_i k)$, is

$$g_e(\lambda_i k) = e^{\lambda_i k}.$$

In the following, we investigate the relative stability and accuracy of: 2nd order, single-stage Adams-Moulton or trapezoidal integration (TRAP); 4th order backwards differentiation (BD4); 4th order Adams-Bashforth (AB4); and, RK4.

A. Relative stability

The absolute stability region of a scheme is the area in the $\lambda_i k$ plane for which $|g(\lambda_i k)| < 1$. For the exact answer, the left-half plane is stable while the right-half plane is unstable. Although many numerical algorithms are concerned only with absolute stability, relative stability is of primary interest for particle path integrators. For example, if the physical mode of a system were (un)stable, the resulting mode of the numerical scheme should also be (un)stable.

In Figure 2, the magnitude of the growth factors for each of these schemes is plotted. The stability boundary (from which the relative stability boundary can be inferred) is denoted by the circles. TRAP is the only scheme considered for which the numerical stability boundary and the exact stability boundary are the same. For example, the stability boundary of RK4 does not include all of the left-half plane. Thus, for the RK4 scheme, some modes which should be damped actually grow in time. As an example, we use the RK4 scheme to integrate the two-dimensional model flow from Murman & Powell[1]:

$$\begin{aligned} u &= ax - by, && (7) \\ v &= ay + bx. && (8) \end{aligned}$$

The exact particle paths are then given by

$$\begin{aligned} x_e(t) &= \exp(at)\left[x_0 \cos bt - y_0 \sin bt\right], \\ y_e(t) &= \exp(at)\left[x_0 \sin bt + y_0 \cos bt\right]. \end{aligned}$$

For this example, we set $a = -1$, $b = 3$ which gives an inward spiraling flow. For these coefficients, the stability boundary of the RK4 scheme is exceeded when $k \geq 0.89$. In Figure 3, the exact particle path is compared with the RK4 paths for $k = 0.88$ and $k = 0.90$ for a starting location of $(x_0, y_0) = (1, 0)$. Using the unstable timestep, $k = 0.90$, clearly gives an outward spiraling path; however, the stable timestep, $k = 0.88$ gives an inward spiraling path as one would expect from the exact path. This type of behavior also can affect the integration of steady velocity fields (see Section A).

The BD4 scheme is stable along the entire negative real axis (only a portion of which is visible in Figure 2); this can be extremely advantageous for flows exhibiting rapid deceleration. Although this cannot be seen from the plot, the region of instability for BD4 is actually bounded and, for large values of $|\lambda_i k|$ in the right-half plane, the unstable physical modes are actually damped by the numerical algorithm. AB4 has significant stability constraints as is expected because of its explicit nature. By contrast, the explicit multi-stage scheme, RK4, has a much larger relative stability region.

B. Accuracy

Although a scheme may be operating within its relative stability region, the accuracy of the algorithm is still unknown. The truncation error can give some accuracy information but is only valid in the limit $k \rightarrow 0$. Instead, we assess accuracy using the growth factor error, E, which we define as

$$E(\lambda_i k) = \left| \frac{g(\lambda_i k) - g_e(\lambda_i k)}{g_e(\lambda_i k)} \right|.$$

The growth factor error includes the effects of both amplification and phase error. Figure 4 contains plots of $\log E$ in the $\lambda_i k$ complex plane for all four schemes. For $|\lambda_i k| < 0.5$, the second order nature of the TRAP scheme is evident in comparison to the other schemes all of which are fourth order. However, for large values of $|\lambda_i k|$, TRAP is actually competitive with BD4 and AB4.

The stability limitations of the AB4 scheme can be clearly seen by the significant decrease in accuracy in the left-half plane. Another interesting aspect of these plots is the poor accuracy of the BD4 scheme for $\mathrm{Real}(\lambda_i k) < -0.45$. Although BD4 is stable along the entire negative real axis, its accuracy is quite poor. However, since these modes decay quite rapidly, this should not significantly degrade the overall accuracy of the method. Of all schemes, RK4 appears to offer the best accuracy over a range of finite $\lambda_i k$.

V. Numerical study of algorithm accuracy

In this section, we consider the performance of several particle path integration schemes for some simple model problems. The schemes, a short description, and their global accuracy are listed in Table 1. The start-up schemes for the AB4 algorithm are the lower order AB algorithms while the initialization method for BD4 first uses TRAP, followed by BD2 and BD3. We use TRAP instead of BD1 in the first step of the BD4 algorithm because of the better accuracy at no extra cost. We compare the numerical and exact particle positions by defining the average global error to be

$$\epsilon = (k/T_f) \sum_{i=1}^{T_f/k} ||\vec{x}^i - \vec{x}_e(t^i)||,$$

recalling T_f is the final time. ϵ is simply an average of the global error, ϵ_G (see Equation 2), at every timestep.

The first model problem is a steady swirling flow with an axial velocity gradient. Specifically, the velocity field and eigenvalues are:

$$\begin{array}{rcl} u &=& \alpha_r y \\ v &=& -\alpha_r x \\ w &=& \alpha_z z \end{array} \quad \Rightarrow \quad \begin{array}{rcl} \lambda_{1,2} &=& \pm i\alpha_r \\ \lambda_3 &=& \alpha_z \end{array}$$

For these results, $\alpha_r = 1.0$ and $\alpha_z = -0.1$; thus, the swirling component of the flow dominates the stability and accuracy of the results. Starting with an initial condition of $x_0 = y_0 = z_0 = 1$, we ran each simulation until $T_f = 100$ for a variety of timestep sizes; the results appear in Figure 5 and the asymptotic error slopes are tabulated in Table 2. The slopes of TRAP, AB4, and RK2 are clearly second order while RK4, RK44, and RK4 2x are fourth order. The BD4 algorithm is nearly third order accurate with a slope of 2.93. Thus, the impact of the start-up error for AB4 and BD4 can be clearly seen. Also, for larger timesteps, the instability of the AB4 scheme is evident in the large error.

We next consider a simple time-dependent flow with no spatial velocity variations.

$$\begin{array}{rcl} u &=& 0 \\ v &=& 0 \\ w &=& \alpha t^{\alpha-1} \end{array} \quad \Rightarrow \quad \lambda_{1,2,3} = 0$$

This model mimics a flow with large unsteadiness but relatively small spatial gradients. The exact answer for this flow is simply, $z = z_0 + t^\alpha$. Thus, schemes with $\partial^{\alpha+1} z/\partial t^{\alpha+1}$ as the lowest order derivative in the global error can exactly integrate this velocity field. Table 3 shows the results for $T_f = 1$, $k = 0.02$ and $\alpha = 2$ and 3. Schemes which use the exact particle positions for start-up are included in the table for comparison. For the $\alpha = 2$ case, only AB4 is inaccurate; this is a result of the start-up error from the initial AB1 iteration. When the exact start-up positions are used to generate the necessary initial data, (see the AB4 *es* results in Table 3), this start-up error is eliminated and the fourth order Adams-Bashforth scheme performs well for $\alpha = 2$. For $\alpha = 3$, start-up errors are present in the AB4, BD4, and RK44 results. The large error in the RK42 results is from the use of a second order interpolant. As discussed in Section III B, this lowers the RK42 scheme to only second order accuracy. In fact, RK42 is equivalent to the RK2 and TRAP schemes when no spatial variations exist. The loss of accuracy and equivalence of RK2, RK42, and TRAP are clearly observed from the data of Table 3. Note, these three schemes do not require any special start-up procedures. From this model's results, we conclude that lower order start-up and interpolation can significantly degrade the accuracy of the particle path integration especially for flows with large unsteadiness.

The final model flow has a large, unsteady axial gradient of the axial velocity and steady velocity fields in the other directions. Specifically,

$$\begin{array}{rcl} u &=& -x \\ v &=& -0.1y \\ w &=& \alpha_0 z e^{\alpha_t t} \end{array} \quad \Rightarrow \quad \begin{array}{rcl} \lambda_1 &=& -1 \\ \lambda_2 &=& -0.1 \\ \lambda_3 &=& \alpha_0 e^{\alpha_t t} \end{array}$$

As $\alpha_0 \rightarrow \infty$, the flow has very large axial gradients initially which die away exponentially with $\alpha_t t$ for $\alpha_t < 0$. The TRAP and BD4 algorithms are well-suited for this type of flow field because of their excellent stability properties along the negative real axis. Although neither BD4 or TRAP accurately represent the large axial velocity gradient, they are stable and this allows the timestep to be set for accurately capturing the moderate flow variations in other directions. The results for $\alpha_0 = -20$ and $\alpha_t = -0.1$ appear in Figure 6 and Table 2. Since we are more concerned with the accuracy of the x and y directions, the abscissa is labeled by $\alpha_x k$. The TRAP and BD4 schemes perform well over a wide range of timesteps while the other schemes suffer at moderate timesteps because of instability from the large axial gradients. Thus, TRAP and BD4 may be more robust than the explicit multi-stage schemes and may be desirable for some flows, especially flows with large flow variations in one direction while relatively moderate variations in the other directions. Note, the asymptotic error slopes from Table 2 show that the TRAP, AB4, and RK2 schemes are second order, BD4 is third order, and RK44 and RK4 2x are fourth order. Thus, except for the RK44 results, the predicted order of accuracy is attained in this model; the RK44 scheme still attains fourth order accuracy contrary to the third order start-up error present in the scheme.

VI. Timestep limits

From the previous analysis, we have found that the eigenvalues of the velocity tensor have a major impact on the accuracy of particle path integrations. In this section, this analysis is used to properly select the timestep for both steady and unsteady velocity fields. In either the steady or unsteady case, the timestep could be constrained to satisfy $E(\lambda_i k) < \epsilon$ where ϵ is some predetermined allowable error. In the worst case, the cumulative error would be $N\,E$ where N is the number of integration steps. This assumes the local growth factor error does not cancel previous local errors. In the following, we use $\epsilon = 10^{-3}$. In our applications of the previous integration schemes, we have found $\epsilon = 10^{-3}$ generally gives accurate results. Figure 4 can be used to guide the selection of an appropriate timestep given an estimate for the velocity tensor eigenvalues and a particular integration scheme. Then, the timestep restriction can be satisfied by

$$k_{\max} = \frac{|\lambda_i k|_\epsilon}{|\lambda_{\max}|}, \tag{9}$$

where $\lambda_{\max}$ is the largest magnitude eigenvalue and $|\lambda_i k|_\epsilon$ is the smallest magnitude of $\lambda_i k$ which guarantees that $E(\lambda_i k) < \epsilon$ when $|\lambda_i k| < |\lambda_i k|_\epsilon$. The procedure to enforce these timestep limits for steady and unsteady flows is described below.

A. Steady data

For steady flows, timesteps may be adaptively sized to account for local flow variations. The goal is to use a timestep which is as large as possible while maintaining acceptable accuracy. The timestep from Equation (9) can lead to timestep limits which are distinctly different than those previously suggested by other authors[1, 3, 7]. These authors suggested setting the timestep such that the particle does not travel more than a given fraction of the length of a cell in an integration step,

$$k_{\max} = \frac{\eta\, l}{|\vec{u}|}, \tag{10}$$

where l is a measure of the local cell size, η is the allowable fraction of the cell size which a particle may travel in a single step, and $|\vec{u}|$ is the magnitude of the local velocity. However, this timestep limit could result in significant difficulties in a region where the flow speed is near zero and the cell size is finite. In this case, the timestep can grow unbounded resulting in significant errors unless the eigenvalues of the velocity tensor are small.

From a practical point of view, the timestep limit of Equation (9) requires the computation of the eigenvalues of the local velocity tensor. Since this is expensive even for a 3×3 matrix, a more efficient technique would be to use an estimate for the magnitude of the largest possible eigenvalue. One possible estimate for the largest eigenvalue is $\tilde{\lambda}_{\max}$, defined as,

$$\begin{aligned} |\tilde{\lambda}_{\max}| = \;& \min\left[\max\left(|\vec{u}_x|, |\vec{u}_y|, |\vec{u}_z|\right),\right. \\ & \left.\max\left(|\nabla u|, |\nabla v|, |\nabla w|\right)\right] \quad \geq |\lambda_{\max}|, \end{aligned}$$

where $\vec{u}_x$ is the x derivative of the velocity vector, ∇ is the gradient operator, and the vector norm $|\vec{u}|$ is the sum of the magnitudes of the elements of $\vec{u}$. This estimate can be derived using the matrix norm to bound the largest eigenvalue and then specifically employing the 1-norm and infinity norm to arrive at the particular expression. In practice, we employ a timestep limit which is a blend of both limits,

$$k_{\max} = \min\left(\frac{\eta\, l}{|\vec{u}|}, \frac{|\lambda_i k|_\epsilon}{|\tilde{\lambda}_{\max}|}\right). \tag{11}$$

The cell fraction timestep limit is used to avoid errors when the flow may have significant variations in the velocity tensor from cell-to-cell.

Problems associated with timestep limits based on Equation (10) have been observed by Murman & Powell[1] for trajectory integrations of vortical flows using the model velocity field in Equations (7) and (8) as well as computed conical flow vortex solutions. In their case, trajectory limit cycles occurred as the particle approached the center of the vortex; this phenomenon is a result of the timestep increasing as the particle neared the core axis and $|\vec{u}| \to 0$. The limit cycle occurs at a radial location where the timestep is no longer in the relative stability region of the integrator. We illustrate this using a forward Euler integration scheme (i.e. AB1) which is also the integrator used by Murman & Powell. For a forward Euler scheme, the constraint that $E < 0.001$ requires that $|\lambda_i k|_\epsilon = 0.04$ approximately. To duplicate the previous results of Murman & Powell, a grid of square cells with length $l = 0.08$ is used, the velocity constants are $a = -1/2$ and $b = 3$, and the cell fraction is $\eta = 1.0$. Then, a particle path is integrated starting from $(x, y) = (1, 0)$ using a forward Euler integrator and the timesteps based on Equation (10) and (11). The results are shown in Figure 7. The limit cycle previously noted by Murman & Powell is evident when the timestep is set by Equation (10), however, the problem is eliminated by the use of Equation (11).

Murman & Powell eliminated this limit cycle by increasing the accuracy of their integrator and decreasing η. However, this strategy does not guarantee the problem is eliminated for all flowfields and grids. A better strategy which corrects the central difficulty is to use a timestep based on the local eigenvalues of the velocity tensor. As can be seen from Figure 7, acceptable answers can be obtained when the correct timestep limit is used even with a low accuracy integrator.

B. Unsteady data

For unsteady flows, the timestep cannot be varied by the particle integration algorithm and is set by the timestep between available planes of velocity data. Before visualization, the timestep is generally sized by the user. If the user is co-processing, the timestep can be set equal to the timestep of the flow algorithm. If the user is post-processing, the timestep must then be set not only by the desired accuracy level but also by the amount of available external memory. In general, the particle timestep, k, is an integer multiple, m, of the flow solver timestep, Δt,

$$k = m\Delta t.$$

Minimizing the total external memory requires the user to maximize m while preserving accuracy.

The timestep limit of Equation (11) is based solely on spatial variations of the flow, however, for unsteady flows, the temporal variations should also be considered when determining the appropriate timestep size. This suggests the following addition to the timestep limit,

$$k_{\max} = \min\left(\frac{\eta l}{|\bar{u}|}, \frac{|\lambda_i k|_\epsilon}{|\tilde{\lambda}_{\max}|}, \eta_u \tau_u\right), \qquad (12)$$

where τ_u is the approximate timescale of the flow unsteadiness and η_u is some fraction used to insure the flow unsteadiness is accurately integrated. To satisfy the timestep limit of Equation (12), the user must have some knowledge of the largest eigenvalue and the timescales expected in the flow. To guarantee that all possible particle paths are accurately calculated, this constraint must be satisfied at all spatial locations, or, at a minimum, the constraint should be satisfied at all spatial locations for which the particle paths are desired.

To better understand the practical implications of Equation (12), we calculated the eigenvalue spectrum for the tapered cylinder calculation of Jespersen and Levit[13] at NASA Ames. This computation features unsteady vortex shedding off of a tapered cylinder at a Reynolds number of 150 based on midspan cylinder radius, R. The convective timescale of the flow is defined as $\tau = R/U_\infty$, and at this Reynolds number, we expect the timescale of the unsteadiness to be the same order as τ. Thus, based solely on the expected unsteadiness, the maximum timestep should be on the order of $\eta_u \tau$. The eigenvalues are from the velocity data at $t = 1200\tau$. To calculate the eigenvalues, the hexahedral computational cells were divided into six tetrahedra. Tetrahedral cells result in a unique linear interpolation and therefore constant gradients and eigenvalues within each tetrahedron. The outer envelope for the velocity tensor eigenvalue spectrum is shown in Figure 8. The imaginary portion of the envelope extends to approximately ± 0.8 and the real portion extends from -0.53 to 0.45. Note, the eigenvalue contour spikes along the real axis correspond to the one purely real eigenvalue which always exists. In order for the eigenvalue spectrum to be contained within the $E = 0.001$ contour in Figure 4, RK4 and BD4 require $k_{\max} = 0.7\tau$ and 0.3τ, respectively. A weaker constraint would require that the eigenvalue timestep limit be satisfied only over a significant portion of the flow. The applicability of this weaker accuracy constraint can be judged using Figure 9 which is a plot of the distribution of the eigenvalue magnitude. Over 90% of the eigenvalues for this calculation have magnitude below 0.2. If we enforce the weak constraint by containing all eigenvalues with magnitude below 0.2 in the $E = 0.001$ contour, the approximate timestep requirements for RK4 and BD4 are $k_{\max} = 2.8\tau$ and 1.2τ. Although only a small fraction of the eigenvalues are near magnitude 1.0, if particle paths will be integrated through these regions, then the previous strong timestep constraint must be used. However, regardless of whether the weak or strong timestep constraint is used, we conclude that accurate particle path integrations for this data set require timesteps at most on the order of the physical timescale of the flow.

Finally, the calculation of accurate particle paths in a post-processing mode could produce significant demands on the external memory requirements if the simulation length lasts several physical timescales. In the Jespersen and Levit data set, the computational timestep was $\Delta t = 0.1\tau$. Thus, strict enforcement of the accuracy constraint requires $m = 7$ and 3 for RK4 and BD4 while weak enforcement requires $m = 28$ and 12 for the same schemes. Thus, the RK4 scheme requires less external memory then the BD4. For the strict constraint, the number of data planes is on the order of the number of integration steps of the flow solver. The weak constraint allows an order of magnitude decrease in the external memory requirements. If the unsteadiness timescale, τ_u, is large such that

$k_{\max}$ is being restricted by the eigenvalue accuracy constraint, some of the external memory demands could be lessened by constructing an interpolant between data planes. Then, using the interpolant to reconstruct the velocity data at the smaller timestep required by the eigenvalue constraint, it should be possible to maintain accuracy as well as decrease the external memory demands. A different approach to alleviate the external memory demands of post-processing unsteady data sets would be to calculate and/or visualize the particle paths in a co-processing mode[14].

VII. Conclusions

We have analyzed a variety of particle path integration schemes suitable for use with unsteady velocity fields. In particular, we addressed implementation, stability, and accuracy issues for a variety of multi-step and multi-stage schemes. Many multi-stage and multi-step schemes have been shown to be limited to third order accuracy due to start-up errors. Furthermore, from a linear analysis, the importance of the eigenvalues of the velocity tensor matrix in determining the stability and accuracy of a generic integration scheme has been shown. For large timesteps outside of the region of relative stability, integration schemes can result in an incorrect classification of critical points. Similar errors occur for streamline integration in steady (or instantaneous) velocity fields if the timestep is chosen improperly.

From the schemes tested, the higher order Runge-Kutta scheme, RK44 and RK4 2x, and the higher order backwards differentiation scheme, BD4, perform well. The RK44 scheme is generally the most accurate scheme tested for finite timesteps but incurs a much larger internal memory penalty than the BD4 scheme. While the RK44 scheme requires the internal storage of four velocity fields, the BD4 scheme only requires the current velocity field. Also, the backwards differentiation scheme is generally more robust than the multi-stage schemes for flows with large spatial gradients in one direction. Depending on the demands of the particular application, the BD4 and RK44 or RK4 2x schemes offer the best performance of the algorithms tested.

Finally, using the velocity data from an unsteady flow simulation, we found the maximum timestep between data frames to be at most on the order of the physical timescale of the flow for accurate particle path integrations. This result seems likely to hold true for most flows and suggests that simultaneous calculation of flow field and particle paths may be necessary to eliminate the disk storage of large amounts of velocity data.

Acknowledgments

The authors would like to thank Tom Woodrow of NASA Ames Research Center and Dave Edwards of United Technologies Research Center for their financial support of this work.

References

[1] E.M. Murman and K.G. Powell. Trajectory integration in vortical flows. *AIAA Journal*, 27(7):982–984, 1989.

[2] D. Modiano, M. Giles, and E. Murman. Visualization of three-dimensional CFD solutions, January 1989.

[3] T. Strid and A. Rizzi. Development and Use of Some Flow Visualization Algorithms. VKI Lecture Series on Computer Graphics and Flow Visualization in CFD, 1989.

[4] G. Volpe. Streamlines and Streamribbons in Aerodynamics. AIAA Paper 89-0140, 1989.

[5] M. Giles and R. Haimes. Advanced Interactive Visualization for CFD. *Computing Systems in Engineering*, 1(1):51–62, 1990.

[6] D. Darmofal and R. Haimes. Visualization of 3-D Vector Fields : Variations on a Stream. AIAA Paper 92-0074, 1992.

[7] S. Shirayama. Processing of computed vector fields for visualization. *Journal of Computational Physics*, 106:30–41, 1993.

[8] L.N. Trefethen. *Finite Difference and Spectral Methods*. M.I.T. Course Notes, 1989.

[9] J.C. Butcher. *The Numerical Analysis of Ordinary Differential Equations*. Wiley, 1987.

[10] A.E. Perry and M.S. Chong. A description of eddying motions and flow patterns using critical-point concepts. *Annual Review of Fluid Mechanics*, 19:125–155, 1987.

[11] M.S. Chong, A.E. Perry, and B.J Cantwell. A general classification of three-dimensional flow fields. *Physics of Fluids A*, 2(5):765–777, 1990.

[12] H. Vollmers, H.-P. Kreplin, and H.U. Meier. Separation and vortical-type flow around a prolate spheroid – evaluation of relevant parameters. AGARD CP 342, 1983.

[13] Dennis Jespersen and Creon Levit. Numerical simulation of flow past a tapered cylinder. AIAA Paper 91-0751, 1991.

[14] R. Haimes. pV3: A Distributed System for Large-Scale Unsteady CFD Visualization. AIAA Paper 94-0321, 1994.

Appendix A: Multi-step scheme coefficients

In this study, three families of schemes are used for the multi-step particle path integration algorithms: Adams-Bashforth, Adams-Moulton, and backwards differentiation. The general forms for the coefficients are shown in Table 4 and the specific coefficients for the schemes are presented in Tables 5-7.

Appendix B: Non-constant timestep algorithms

Most of the algorithms which we have discussed for constant timesteps can easily be extended to non-constant timesteps. As an example, we derive the non-constant timestep version of the BD4 scheme. For the constant timestep algorithm, the position coefficients, α_i, can be found in two equivalent manners: (1) using a truncation error analysis and eliminating consecutively higher error terms, (2) differentiating a polynomial interpolant. Although both techniques yield the same results, the polynomial interpolant method allows easier generalization to non-constant timesteps. Specifically, consider a polynomial interpolant, $x_p(t)$, of the following form:

$$\vec{x}_p(t) = \sum_{i=0}^{4} \gamma_i(t)\, \vec{x}^{n+1-i}$$

where

$$\gamma_i(t) = \prod_{j \neq i} \frac{t - t^{n+1-j}}{t^{n+1-i} - t^{n+1-j}}.$$

Differentiating the interpolant gives

$$\frac{d}{dt}\vec{x}_p(t) = \sum_{i=0}^{4} \vec{x}^{n+1-i}\, \frac{d}{dt}\gamma_i(t).$$

Finally, since we are interested in calculating the time derivative of the particle position at t^{n+1}, the position coefficients, α_i, can be found by simply equating the coefficients for $\frac{d}{dt}\vec{x}_p$ at t^{n+1}. This gives

$$\alpha_i = \left. \frac{d\gamma_i}{dt} \right|_{t^{n+1}}.$$

At every iteration, the values of α_i must be re-evaluated using this analytic expression.

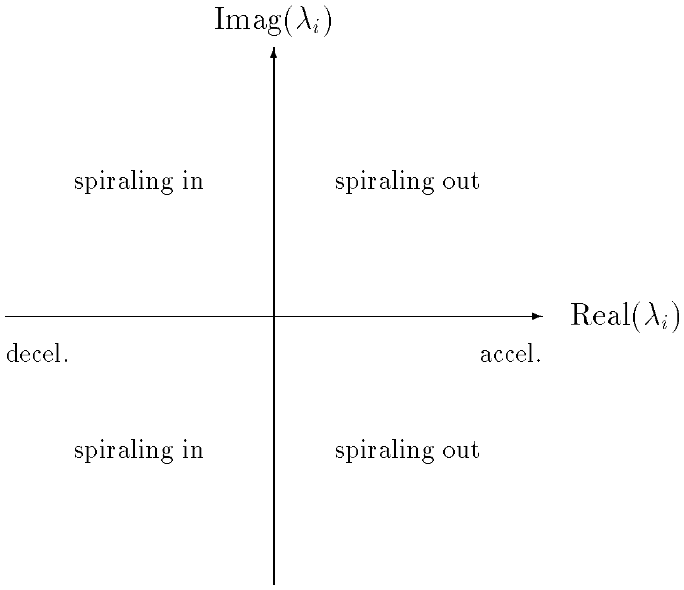


Figure 1: Flow classifications in eigenvalue plane, $\lambda_{1,2,3} = \mathrm{eig}\,(\partial\vec{u}/\partial\vec{x})$.

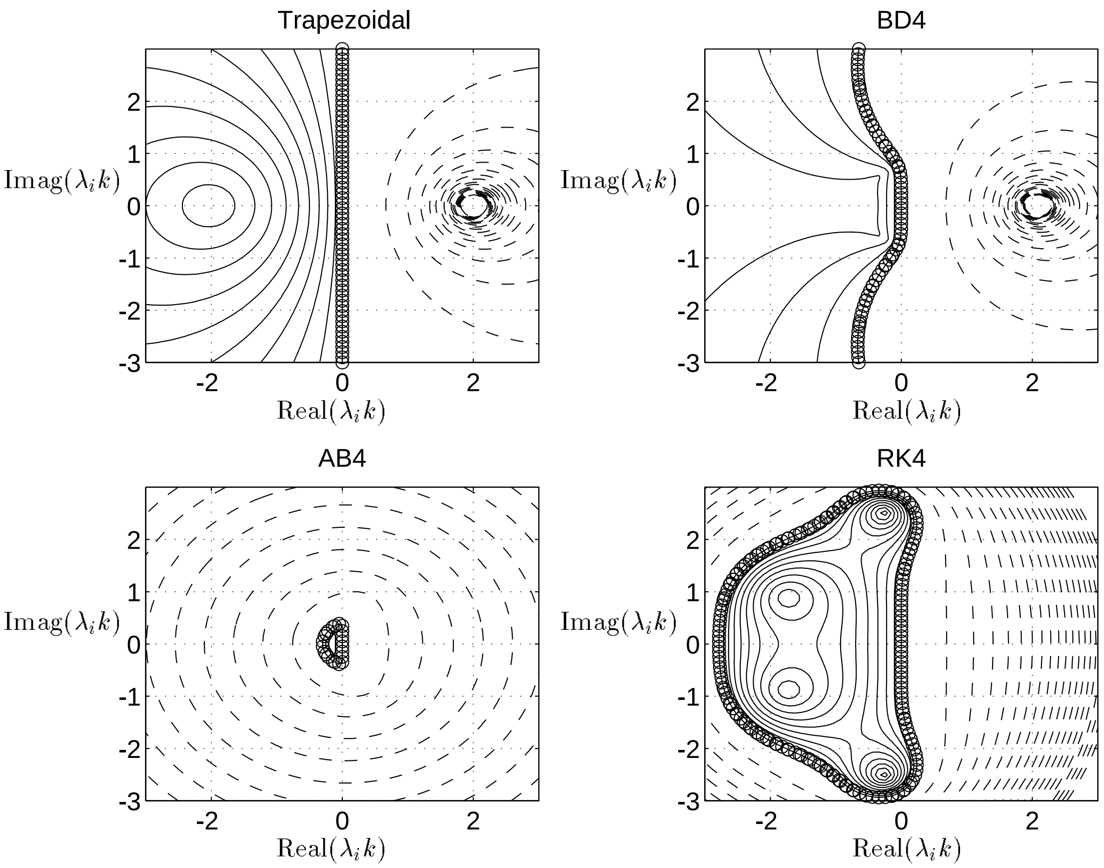


Figure 2: Magnitude of growth factor in $\lambda_i k$ plane. $|g| < 1$, solid line; $|g| = 1$, circles; $|g| > 1$, dashed line.

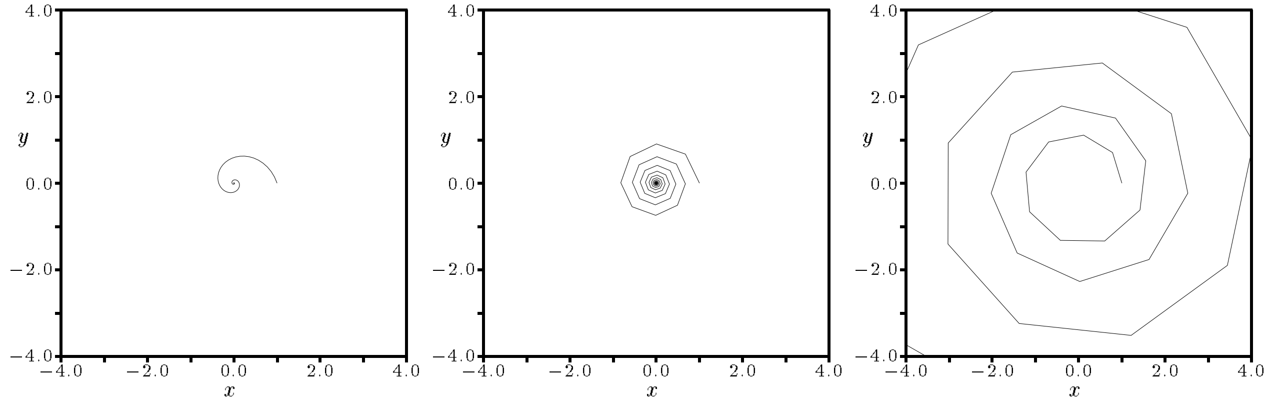


Figure 3: Comparison of exact particle path with numerical paths for stable and unstable timesteps, RK4 algorithm. Initial particle location $(x_0, y_0) = (1, 0)$.

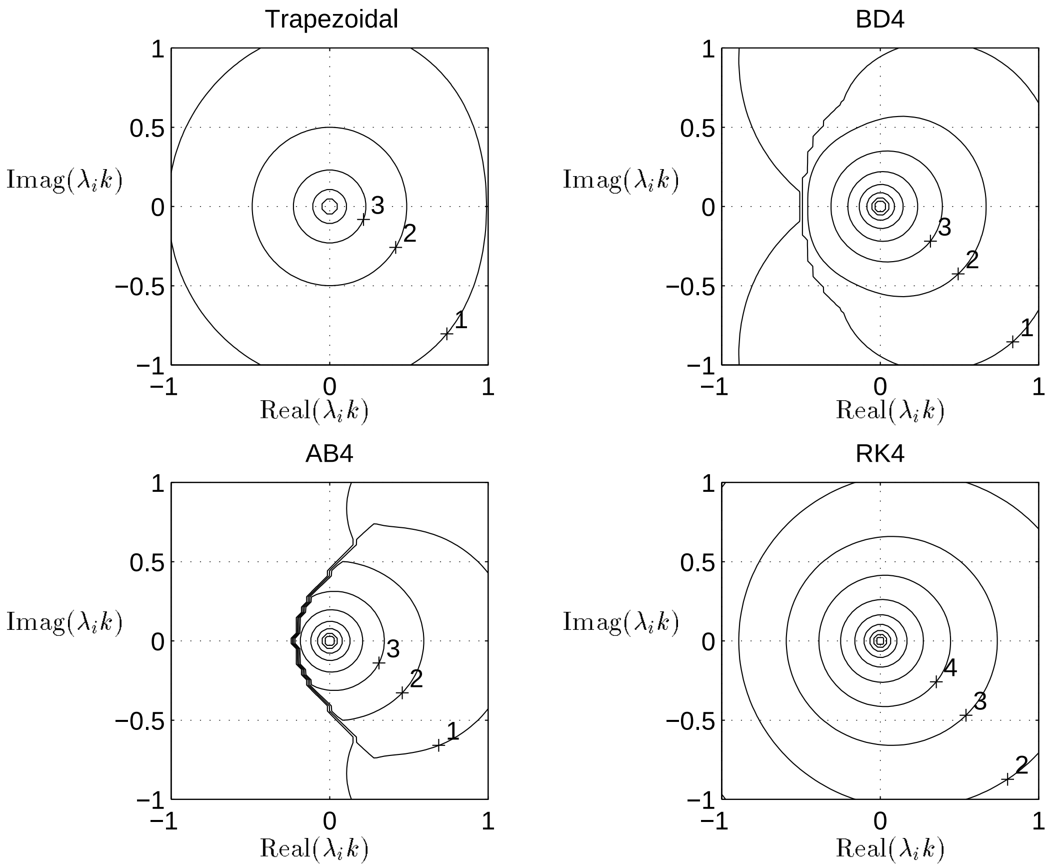


Figure 4: Contour plots of $-\log E$ in $\lambda_i k$ plane.

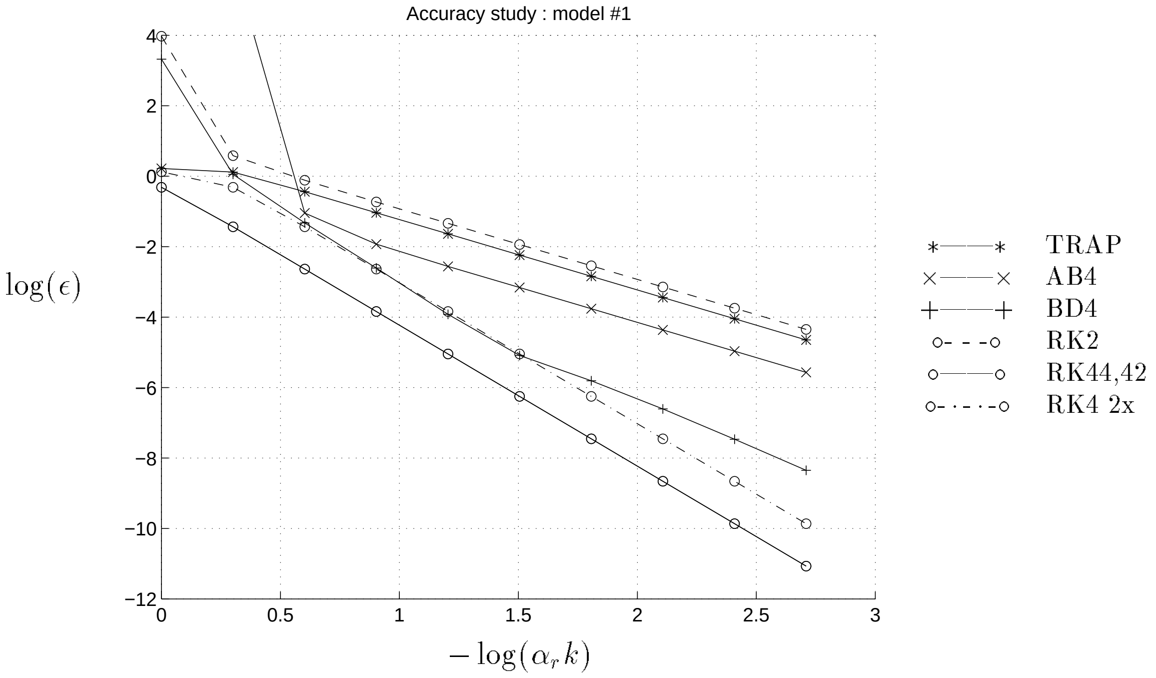


Figure 5: Results from Model #1. $\alpha_r = 1$, $\alpha_z = -0.1$. $T_f = 100$. $x_0 = y_0 = z_0 = 1$. Note: results for RK42 and RK44 scheme identical for steady vector fields.

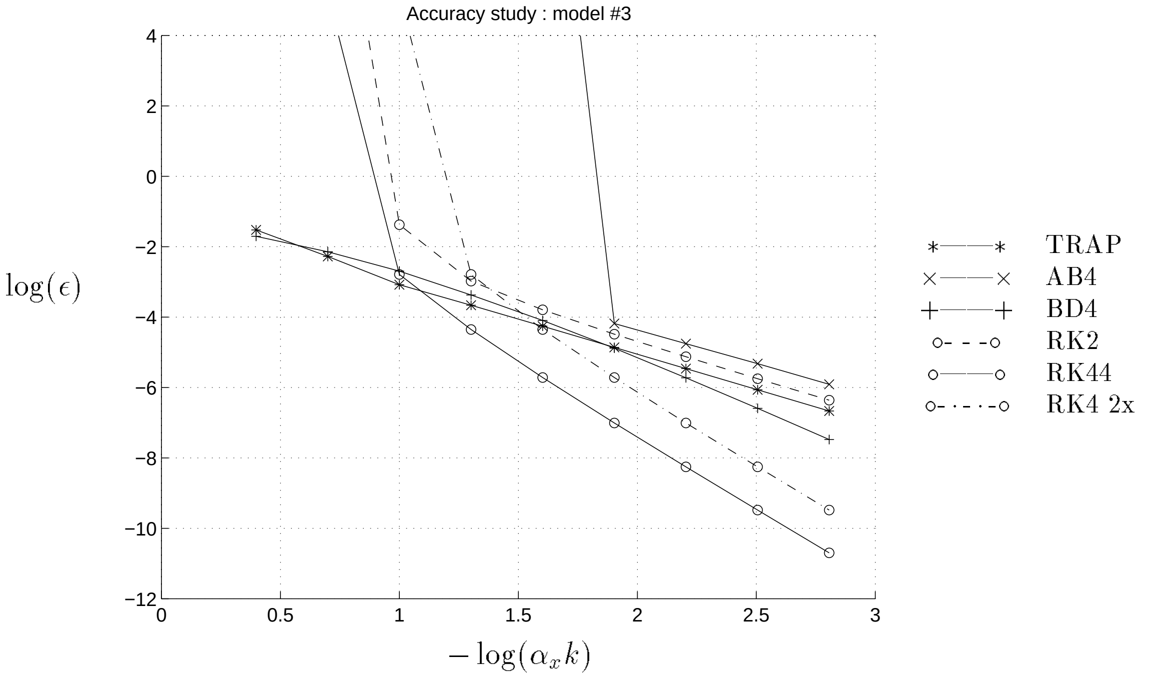
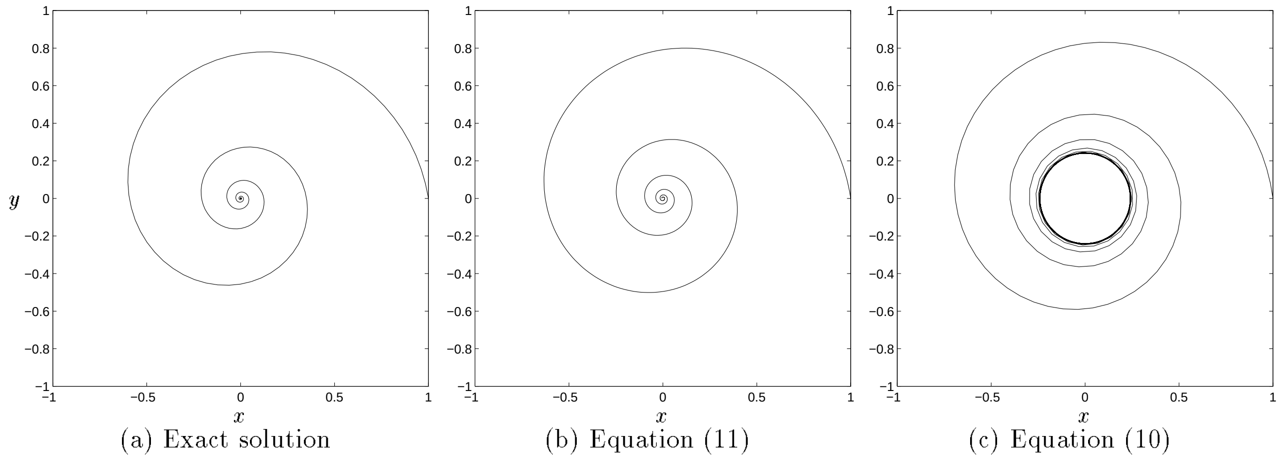


Figure 6: Results from Model #3. $\alpha_0 = -20$, $\alpha_t = -0.1$. $T_f = 20$. $x_0 = y_0 = z_0 = 1$.



(a) Exact solution (b) Equation (11) (c) Equation (10)

Figure 7: Particle trajectories for steady flows using forward Euler integrator and different timestep limits.

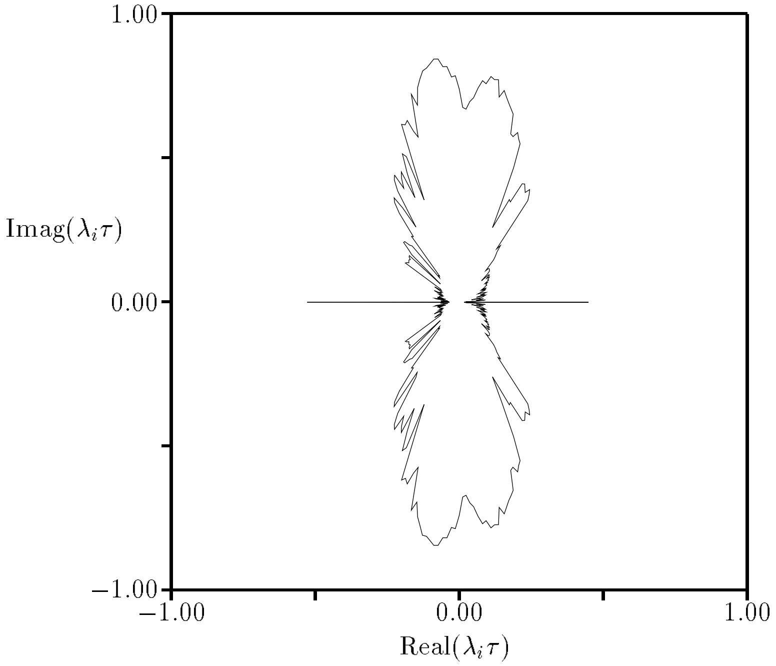


Figure 8: Outer envelope of velocity tensor eigenvalues for Jespersen & Levit[13] tapered cylinder calculation.

r	Scheme	Description
2	**TRAP**	Trapezoidal integration
3	**BD4**	Fourth order backwards differentiation, $O(k^3)$ start-up
2	**AB4**	Fourth order Adams-Bashforth, $O(k^2)$ start-up
2	**RK2**	Second order Runge-Kutta
2	**RK42**	Fourth order Runge-Kutta, $O(k^2)$ interpolant
3	**RK44**	Fourth order Runge-Kutta, $O(k^4)$ interpolant, $O(k^3)$ start-up
4	**RK4 2x**	Fourth order Runge-Kutta, double timestep

Table 1: Global error (r) and summaries of schemes used in numerical studies

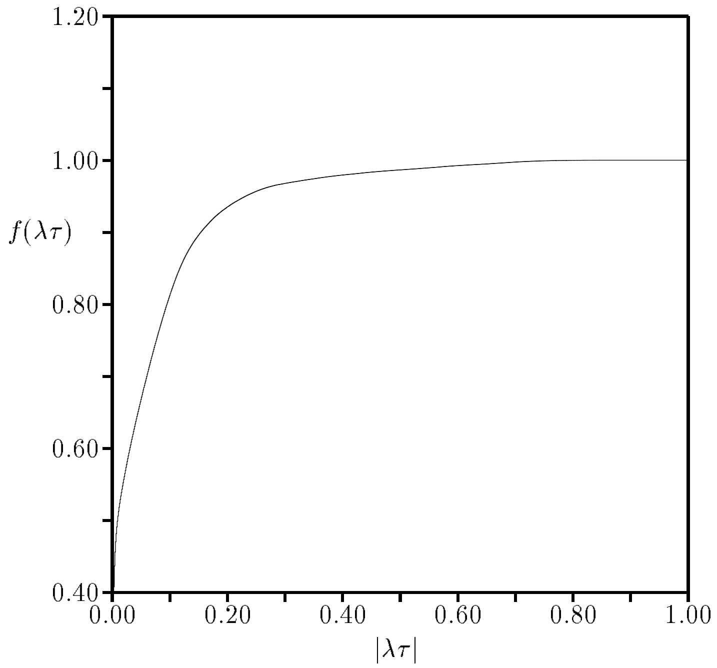


Figure 9: Fraction of eigenvalues, $f(\lambda\tau)$, with magnitude less than $|\lambda\tau|$ for Jespersen & Levit[13] tapered cylinder calculation.

	$\log\epsilon$ vs. $\log\alpha k$ slope	
Scheme	Model #1	Model #3
TRAP	2.00	2.00
BD4	2.93	2.94
AB4	2.00	1.95
RK2	2.00	2.03
RK44	4.00	4.04
RK4 2x	4.00	4.07

Table 2: Asymptotic slopes of error from Model #1 and #3 results in Figures 5 and 6. Slopes calculated from final two data points of each line.

$\log\epsilon$	α	
Scheme	2	3
TRAP	-15.80846	-3.99140
BD4	-14.91478	-4.67769
BD4 es	-14.66056	-14.85560
AB4	-3.39794	-4.55909
AB4 es	-16.20640	-15.77275
RK2	-15.80846	-3.99140
RK42	-15.80846	-3.99140
RK44	-15.80846	-5.39794
RK44 es	-16.20640	-15.77275
RK4 2x	-Infinity	-15.57438

Table 3: Results from Model #2. $T_f = 1$, $k = 0.02$. $x_0 = y_0 = z_0 = 1$. es - exact start-up.

Scheme		0	1	2	3	4
Adams-Bashforth	α_i	x	x			
	β_i		x	x	x	x
Adams-Moulton	α_i	x	x			
	β_i	x	x	x	x	x
Backwards Diff.	α_i	x	x	x	x	x
	β_i	x				

Table 4: General form of multi-step algorithms. x : non-zero coefficient

number of step, s	order, p	β_1	β_2	β_3	β_4
1	1	1			
2	2	$\frac{3}{2}$	$-\frac{1}{2}$		
3	3	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$	
4	4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$

Table 5: Adams-Bashforth coefficients

number of step, s	order, p	β_0	β_1	β_2	β_3
1	2	$\frac{1}{2}$	$\frac{1}{2}$		
2	3	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$	
3	4	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$

Table 6: Adams-Moulton coefficients

number of step, s	order, p	β_0	α_1	α_2	α_3	α_4
1	1	1	-1			
2	2	$\frac{2}{3}$	$-\frac{4}{3}$	$\frac{1}{3}$		
3	3	$\frac{6}{11}$	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$	
4	4	$\frac{12}{25}$	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$

Table 7: Backwards differentiation coefficients