ANALYTIC STREAMLINE CALCULATIONS ON LINEAR TETRAHEDRA

Darin P. Diachin* and James A. Herzog†

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439
{diachin,herzog}@mcs.anl.gov

Abstract

Analytic solutions for streamlines within tetrahedra are used to define operators that accurately and efficiently compute streamlines. The method presented here is based on linear interpolation, and therefore produces exact results for linear velocity fields. In addition, the method requires less computation than the forward Euler numerical method. Results are presented that compare accuracy measurements of the method with forward Euler and fourth-order Runge-Kutta applied to both a linear and a nonlinear velocity field.

1 Introduction

Streamlines are a common tool used to visualize steady flow fields. They are generated by calculating integral curves along a given static velocity field and can be interpreted as the path a massless particle would follow when placed within the field. The motion of these massless particles is defined by

$$\frac{d}{dt}\mathbf{p}(t) = \mathbf{u}(\mathbf{p}(t)), \tag{1}$$

where $\mathbf{p}(t)$ represents the particle's position within the field, $\mathbf{u}(\mathbf{p}(t))$ is the velocity of the field at the given position, and t is a parameter along the streamline. Throughout this paper we scale t so that it is equivalent to the time scale of the velocity field.

Many flow fields in engineering and scientific applications are computed on a discrete mesh by using finite difference, finite volume, or finite element techniques. The work presented here pertains to calculating streamlines on tetrahedral meshes with velocities defined at the vertices. Our method can be applied to hexahedral meshes by decomposing each cell into five or six tetrahedra. One such decomposition algorithm is presented by Kenwright and Lane.[2]

Traditionally, streamlines are calculated by numerically integrating equation (1). Care must be taken to choose an appropriate time step for these integration techniques to maintain numerical accuracy and stability. Darmofal and Haimes provide an analysis of many integration algorithms used for calculating streamlines.[1] To ensure numerical stability and maintain local error bounds, they suggest using the eigenvalues of the velocity tensor along with a method-dependent amplification function to compute a time step.

By using analytic solutions for streamlines on linearly varying fields, we have eliminated the necessity for bounding the time step. By discretely stepping along these analytic streamlines, we achieve accuracies that are consistently better than the commonly used fourth-order Runge-Kutta method. To maintain efficiency, we use a technique similar to a specialized fourth-order Runge-Kutta method by Sikorski et al.[5] Our method requires fewer computations than the forward Euler integration technique implemented with a constant time step, provided enough memory is available to store a 3×3 matrix and a 3-vector for every cell through which the streamline passes. We will refer to our method as ANTS, abbreviated for analytic time stepping.

The remainder of this paper is organized as follows. In the next section we review the general framework for computing streamlines, and in Section 3 we provide an overview of the specialized fourth-order Runge-Kutta (SRK4) method mentioned above. Sections 4 and 5 present the analytic solutions for streamlines and their subsequent integration into the ANTS algorithm. Section 6 presents

*Graduate Research Appointee at ANL. Doctoral candidate at Northwestern University in Theoretical and Applied Mechanics

†Graduate Research Appointee at ANL. Doctoral candidate at Stanford University in Scientific Computing and Computational Mathematics

a few important aspects of our implementation, and Section 7 provides accuracy and timing results for streamline calculations performed on both linear and nonlinear velocity fields. Section 8 summarizes our conclusions.

2 Calculating Streamlines

```
Pick a seed location, p(0), for a streamline
t ← 0
While the particle is in the domain
    Locate the mesh cell containing p(t)
    Transform the cell into computational        (1)
        space if necessary
    While the particle is in the cell
        Interpolate the particle's velocity      (2)
            at time t
        Integrate equation (1) to get the        (3,4)
            particle's position at time t + h
        t ← t + h
    Endwhile
    Map the streamline back to physical          (1)
        space if necessary
Endwhile
```

Figure 1: Common numerical algorithm for computing streamlines

The general algorithm for calculating a streamline on a discrete mesh is provided in Figure 1. The first important step is locating the cell in the mesh that contains the initial position of the streamline. While the efficiency of an algorithm for searching an entire mesh to locate this cell is largely dependent on the shape of the domain and the nature of the mesh data structure, much work has been done developing general-purpose algorithms for point location in individual cells. In particular, there exist efficient algorithms for point location in both tetrahedral and hexahedral cells.[2]

With some streamline calculation algorithms the numerical integration of equation (1) is performed more efficiently in a transformed space.[6] While we perform our integration in physical space, we do use a transformation from physical coordinates to canonical coordinates to formulate the governing equation for a streamline. This transformation is discussed in Section 4.

Interpolation techniques are required to estimate the velocity in equation (1) at any point in the domain using the discrete velocity values at nearby grid points. ANTS is based on linear interpolation which provides a continuous velocity field over the entire domain. However, the method results in discontinuities in the acceleration at cell boundaries for nonlinear velocity fields. Linear interpolation is also described in detail in Section 4.

The steps responsible for introducing numerical error are numbered on the right-hand side of Figure 1. The first source of error is associated with the transformation to computational space. This error is generally greater for hexahedral cells due to the nonlinear transformations commonly used.[2, 6] Since ANTS does not require a transformation to computational space, it does not introduce error of type 1.

Type 2 error is introduced when the interpolation function does not match the nature of the velocity field. For example, linear interpolation is exact only on linearly varying fields. In the case of a quadratically varying field, type 2 error can be managed by using quadratic interpolation techniques or, less effectively, by reducing the cell size.

Two types of error are associated with integrating equation (1) using linear interpolation on tet meshes. Type 3 error is the numerical error introduced by the specific integration technique and is the type of error that is greatly reduced with ANTS. Type 4 error is associated with the discontinuity in acceleration across cell boundaries. Care must be taken to limit the amount of velocity information used from a given cell in calculations performed in neighboring cells. For instance, ANTS, SRK4, and forward Euler schemes all use velocity information interpolated at a position near a face of a cell to track across the boundary into a neighboring cell. If the time step is too large, a significant amount of error will be introduced due to the discontinuity in acceleration. Fourth-order Runge-Kutta has the same problem if one-fourth the time step is sufficiently large.

Type 4 error can be eliminated using the results presented in Section 5 of this paper. The process requires computing successive intersections of the analytic solution with subsequent cell faces. However, this process is, in general, computationally more expensive than ANTS.

3 Specialized Fourth-Order Runge-Kutta Method

The ANTS algorithm is based on a specialized version of the fourth-order Runge-Kutta (SRK4) method.[5] In this method, Sikorski et al. use a linear interpolation function on tetrahedra to reduce each

time step of the integration to a matrix-vector multiplication and a vector-vector addition, provided the tet geometry and velocity data are static and a constant time step is used. They demonstrate how the fourth-order Runge-Kutta formulae in this case can be reduced to a linear operation,

$$\mathbf{p}(t+h) = H\mathbf{p}(t) + \mathbf{d}, \tag{2}$$

where H is a constant 3×3 matrix, $\mathbf{d}$ is a constant 3-vector, and $\mathbf{p}(t)$ is any position within the tetrahedron. We will refer to H as the time-stepping matrix and $\mathbf{d}$ as the time-stepping vector, and we will refer to equation (2) as the time-stepping formula. Note that the time step for SRK4 must be carefully chosen to maintain numerical stability. We show in this paper that error due to integration can be reduced by computing the time-stepping matrix and vector using the analytic solution to the streamline instead of the fourth-order approximation used in SRK4. The new algorithm for calculating streamlines is provided in Figure 2.

Calculate the time-stepping matrix and time-stepping vector for each cell using the analytic formulation for streamlines
Pick a seed location, $\mathbf{p}(0)$, for a streamline
$t \leftarrow 0$
While the particle is in the domain
  Locate the mesh cell containing $\mathbf{p}(t)$
  While the particle is in the cell
    Calculate $p(t+h)$ using the time-stepping formula
    $t \leftarrow t + h$
  Endwhile
Endwhile

Figure 2: Analytic time-stepping algorithm for computing streamlines

4 Governing Equation

In this section we formulate the governing equation for streamlines that is used in deriving the time-stepping matrix and vector. The derivation comprises two fundamental steps: first, the linear interpolation function is used to derive the equation for streamlines in canonical coordinates, and second, the canonical equation is mapped into physical space by using the transformation between the two spaces.

Regarding notation, we use the variable name $\mathbf{n}$ for positions in canonical space and $\mathbf{v}$ for canonical velocities, while $\mathbf{p}$ and $\mathbf{u}$ represent the respective quantities in physical coordinates (see Figure 3). Further, the following subscript convention is used to relate tets under the transformation from physical to canonical space: variables with subscript zero correspond to data defined on the vertex mapped to the canonical origin, subscript one corresponds to data on vertices mapped to $(1,0,0)$, subscript two corresponds to $(0,1,0)$ and subscript three corresponds to vertex $(0,0,1)$. We refer to $\mathbf{p}_0$ as the tetrahedron's origin vertex.

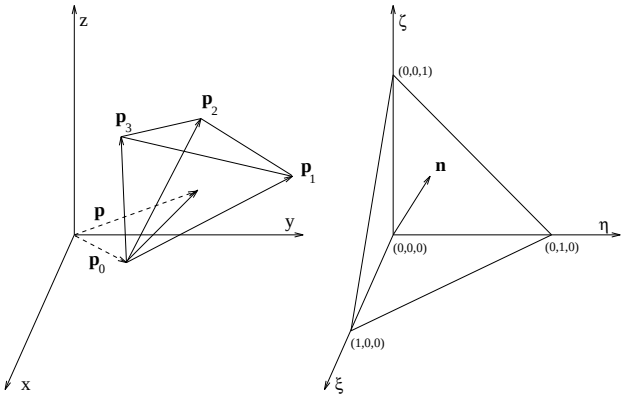

Figure 3: The tetrahedron on the left represents a cell in physical coordinates, and the tetrahedron on the right is the canonical tetrahedron.

4.1 Linear Interpolation

A linear interpolation scheme assumes that both scalar and vector data vary linearly in each coordinate direction within the cell. The scalar interpolation formula for the canonical tet is

$$s(\xi,\eta,\zeta) = (s_1 - s_0)\xi + (s_2 - s_0)\eta + (s_3 - s_0)\zeta + s_0, \tag{3}$$

where s could represent any of a number of scalar parameters, including temperature or chemical species concentration. Here, s_0, s_1, s_2, and s_3 are the values of s at the vertices.

The vector interpolation formula is analogous:

$$\begin{aligned} \mathbf{v}(\xi,\eta,\zeta) &= (\mathbf{v}_1 - \mathbf{v}_0)\xi + (\mathbf{v}_2 - \mathbf{v}_0)\eta \\ &\quad +(\mathbf{v}_3 - \mathbf{v}_0)\zeta + \mathbf{v}_0 \\ &= V\mathbf{n} + \mathbf{v}_0. \end{aligned} \tag{4}$$

The matrix V has columns containing the difference of the vertex velocity vectors in canonical coordinates. Noting that the velocity is the time derivative

of the position, $\mathbf{n}(t)$, and writing each coordinate in terms of the time t, we have the governing equation for a streamline in canonical coordinates:

$$\frac{d\mathbf{n}(\xi(t),\eta(t),\zeta(t))}{dt} = V\mathbf{n}(\xi(t),\eta(t),\zeta(t)) + \mathbf{v}_0. \quad (5)$$

4.2 Transformation from Physical to Canonical Coordinates

Transformations between physical space and computational space are common for streamline calculation algorithms on both tetrahedral and hexahedral meshes. For example, transformations are often used in point location and data interpolation algorithms. In the case of tetrahedral meshes, a point is determined to be in a given cell if the point's canonical coordinates with respect to the cell are all greater than zero and the sum of the canonical coordinates is less one. Further, linear interpolation in physical coordinates for scalar or vector quantities can be performed by mapping into canonical coordinates and applying equation (3) or (4), respectively.

A distinct advantage of using tetrahedral cells rather than hexahedral cells is that the transformation from physical space to canonical space is linear. This property allows for an analytic solution to the inverse transformation, which is essential to the derivation of the analytic solutions for streamlines in physical coordinates. In the case of hex cells, the transformation to computational space is nonlinear and requires expensive iterative methods to perform. [2]

The transformation from physical to canonical coordinates is provided below:

$$\begin{aligned} \mathbf{n}(\xi,\eta,\zeta) &= B(\mathbf{p}(x,y,z) - \mathbf{p}_0), \quad (6)\\ B &= [\mathbf{p}_1 \quad \mathbf{p}_2 \quad \mathbf{p}_3]^{-1}. \end{aligned}$$

Here, B is the inverse of the 3×3 matrix containing the vectors along the edges emanating from the tet's origin vertex (see Figure 3).

Several quantities related to the transformation, which are used in deriving the defining equation for streamlines in physical coordinates, are provided below:

$$\begin{aligned} \mathbf{n}(\xi(t),\eta(t),\zeta(t)) &= B(\mathbf{p}(x(t),y(t),z(t)) - \mathbf{p}_0),\\ \frac{d}{dt}\mathbf{n}(\xi(t),\eta(t),\zeta(t)) &= B\frac{d}{dt}\mathbf{p}(x(t),y(t),z(t)),\\ V &= BU,\\ \mathbf{v}_0 &= B\mathbf{u}_0. \end{aligned}$$

U is the matrix with columns containing the difference in vertex velocities, and $\mathbf{u}_0$ is the velocity at the tet's origin. We derive the governing equation for streamlines in physical coordinates by replacing the quantities above into the governing equation for streamlines in canonical coordinates (equation (5)):

$$\begin{aligned} \frac{d}{dt}\mathbf{p}(x(t),y(t),z(t)) &= A\mathbf{p}(x(t),y(t),z(t)) \quad (7)\\ &\quad -A\mathbf{p}_0 + \mathbf{u}_0,\\ A &= UB. \end{aligned}$$

5 Analytic Solutions

This section outlines the methods for deriving the time-stepping matrix and time-stepping vector using analytic solutions to equation (7). There are four general subsets of solutions to this equation depending on the rank of A. Each case is presented independently below. For ease of notation, the physical coordinates, $\mathbf{p}(\xi(t),\eta(t),\zeta(t))$, will be written as a function of t for the remainder of this paper. Also, we equate the *rank of the system* with the rank of A.

5.1 Rank Three Systems

The solution to equation (7) for a full rank system is provided below:

$$\begin{aligned} \mathbf{p}(t) &= e^{At}\mathbf{k}_1 + \mathbf{k}_2, \quad (8)\\ \mathbf{k}_1 &= \mathbf{p}(0) - \mathbf{k}_2,\\ \mathbf{k}_2 &= \mathbf{p}_0 - A^{-1}\mathbf{u}_0, \end{aligned}$$

where $\mathbf{p}(0)$ is the initial physical position. The matrix e^{At} is the exponential of the matrix At and is defined with the power series

$$e^{At} = I + At + \frac{A^2t^2}{2!} + \ldots . \quad (9)$$

The exponential matrix is discussed further by Moler and Van Loan.[3] Note that A has full rank in this case, and therefore the inverse of A exists.

The time-stepping formula provides the position of the particle at time $t+h$ for a constant time step h. The formula is derived by evaluating $\mathbf{p}(t+h)$ and writing the result in terms of $\mathbf{p}(t)$:

$$\begin{aligned} \mathbf{p}(t+h) &= e^{A(t+h)}\mathbf{k}_1 + \mathbf{k}_2, \quad (10)\\ &= e^{Ah}\mathbf{p}(t) + (I - e^{Ah})\mathbf{k}_2,\\ &= H\mathbf{p}(t) + \mathbf{d}. \end{aligned}$$

This formulation uses the property of exponential matrices that $\exp(A(t+h)) = \exp(At)\exp(Ah)$ for scalars t and h.

5.2 Rank Two Systems

The solutions to the governing equation for rank deficient systems can be derived by changing coordinate systems, $\tilde{\mathbf{p}} = W\mathbf{p}$, where W is orthonormal mapping. The critical step is to find this mapping W under which the transformed matrix, $\tilde{A}$, has the appropriate number of zero rows. A rank two $\tilde{A}$ matrix will have one zero row, and a rank one $\tilde{A}$ will have two zero rows. In the case of a rank two system, the governing equation is written in the new coordinate system as follows:

$$\begin{bmatrix} \dot{\tilde{x}} \\ \dot{\tilde{y}} \\ \dot{\tilde{z}} \end{bmatrix} = \begin{bmatrix} \tilde{a}_{11} & \tilde{a}_{12} & \tilde{a}_{13} \\ \tilde{a}_{21} & \tilde{a}_{22} & \tilde{a}_{23} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} + \begin{bmatrix} \tilde{w}_1 \\ \tilde{w}_2 \\ \tilde{w}_3, \end{bmatrix},$$

$$\tilde{\mathbf{w}} = \begin{bmatrix} \tilde{w}_1 & \tilde{w}_2 & \tilde{w}_3 \end{bmatrix}^T = \tilde{\mathbf{u}}_0 - \tilde{A}\tilde{\mathbf{p}}_0.$$

The solution in the $\tilde{z}$ coordinate is found using direct integration of the constant $\tilde{w}_3$,

$$\tilde{z}(t) = \tilde{w}_3 t + \tilde{z}(0), \tag{11}$$

where $\tilde{z}(0)$ is the initial $\tilde{z}$ coordinate. We will refer to the solution in the $\tilde{x}$ and $\tilde{y}$ coordinates $\mathcal{P}(t) = [\tilde{x}(t), \tilde{y}(t)]^T$:

$$\mathcal{P}(t) = e^{\mathcal{A}t}\left[\mathcal{P}(0) - \boldsymbol{\beta}_1\tilde{z}(0) - \boldsymbol{\beta}_2\right] + \boldsymbol{\alpha}\mathrm{t} + \boldsymbol{\beta}_1\tilde{z}(0) + \boldsymbol{\beta}_2,$$

$$\mathcal{A} = \begin{bmatrix} \tilde{a}_{11} & \tilde{a}_{12} \\ \tilde{a}_{21} & \tilde{a}_{22} \end{bmatrix}.$$

$\mathcal{P}(t)$ is a function of the exponential of the left principle submatrix of $\tilde{A}t$, the initial position $\tilde{\mathbf{p}}(0) = [\tilde{x}(0), \tilde{y}(0), \tilde{z}(0)]^T$, and the 2-vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}_1$, and $\boldsymbol{\beta}_2$ defined below:

$$\boldsymbol{\alpha} = \frac{\tilde{\mathrm{w}}_3}{\det(\mathcal{A})}\begin{bmatrix} \tilde{a}_{12}\tilde{a}_{23} - \tilde{a}_{13}\tilde{a}_{22} \\ \tilde{a}_{11}\tilde{a}_{23} - \tilde{a}_{13}\tilde{a}_{21} \end{bmatrix},$$

$$\boldsymbol{\beta}_1 = \frac{1}{\det(\mathcal{A})^2}\sum_{\mathrm{i}=1}^{4}\begin{bmatrix} s_{1i} \\ s_{2i} \end{bmatrix}:$$

$$\begin{aligned} s_{11} &= \tilde{a}_{21}\tilde{a}_{12}\tilde{a}_{22}\tilde{a}_{23}, & s_{21} &= \tilde{a}_{12}\tilde{a}_{21}\tilde{a}_{11}\tilde{a}_{23}, \\ s_{12} &= -\tilde{a}_{21}\tilde{a}_{12}^2\tilde{a}_{13}, & s_{22} &= -\tilde{a}_{12}\tilde{a}_{21}^2\tilde{a}_{13}, \\ s_{13} &= \tilde{a}_{12}\tilde{a}_{11}\tilde{a}_{22}\tilde{a}_{23}, & s_{23} &= \tilde{a}_{21}\tilde{a}_{11}\tilde{a}_{22}\tilde{a}_{13}, \\ s_{14} &= -\tilde{a}_{11}\tilde{a}_{22}^2\tilde{a}_{23}, & s_{24} &= -\tilde{a}_{22}\tilde{a}_{11}^2\tilde{a}_{23}, \end{aligned}$$

$$\boldsymbol{\beta}_2 = \frac{1}{\det(\mathcal{A})^2}\sum_{\mathrm{i}=1}^{8}\begin{bmatrix} t_{1i} \\ t_{2i} \end{bmatrix}:$$

$$\begin{aligned} t_{11} &= \tilde{a}_{21}\tilde{a}_{12}\tilde{a}_{22}\tilde{w}_1, & t_{21} &= \tilde{a}_{12}\tilde{a}_{23}\tilde{a}_{11}\tilde{w}_2, \\ t_{12} &= -\tilde{a}_{21}\tilde{a}_{12}\tilde{a}_{13}\tilde{w}_3, & t_{22} &= -\tilde{a}_{12}\tilde{a}_{21}\tilde{a}_{23}\tilde{w}_3, \\ t_{13} &= \tilde{a}_{12}\tilde{a}_{11}\tilde{a}_{23}\tilde{w}_3, & t_{23} &= \tilde{a}_{23}\tilde{a}_{22}\tilde{a}_{13}\tilde{w}_3, \\ t_{14} &= \tilde{a}_{12}\tilde{a}_{11}\tilde{a}_{22}\tilde{w}_2, & t_{24} &= \tilde{a}_{21}\tilde{a}_{11}\tilde{a}_{22}\tilde{w}_1, \\ t_{15} &= \tilde{a}_{12}\tilde{a}_{23}\tilde{a}_{22}\tilde{w}_3, & t_{25} &= \tilde{a}_{21}\tilde{a}_{11}\tilde{a}_{13}\tilde{w}_3, \\ t_{16} &= -\tilde{a}_{13}\tilde{a}_{22}^2\tilde{w}_3, & t_{26} &= -\tilde{a}_{12}\tilde{a}_{21}^2\tilde{w}_1, \\ t_{17} &= -\tilde{a}_{11}\tilde{a}_{22}^2\tilde{w}_2, & t_{27} &= -\tilde{a}_{22}\tilde{a}_{11}^2\tilde{w}_2, \\ t_{18} &= -\tilde{a}_{22}\tilde{a}_{11}^2\tilde{w}_1, & t_{28} &= -\tilde{a}_{23}\tilde{a}_{11}^2\tilde{w}_3. \end{aligned}$$

Note if $\det(\mathcal{A}) = 0$ another orthonormal mapping W would have to be used to achieve a solution of this form.

The time-stepping matrix and vector in the tilde coordinate system are found using the same procedure used for the rank three system. Position $\tilde{\mathbf{p}}(t+h) = [\mathcal{P}(t+h), \tilde{z}(t+h)]^T$ is calculated and written in terms of $\tilde{\mathbf{p}}(t)$. The results are below:

$$\tilde{H} = \begin{bmatrix} e^{\mathcal{A}h} & (I - e^{\mathcal{A}h})\beta_1 \\ \vec{\mathbf{0}}^T & 1 \end{bmatrix},$$

$$\tilde{\mathbf{d}} = \begin{bmatrix} (I - e^{\mathcal{A}h})\beta_2 + \alpha h \\ \tilde{w}_3 h \end{bmatrix}$$

Finally, $\tilde{H}$ and $\tilde{\mathbf{d}}$ are mapped into physical coordinates using the following transformations:

$$\begin{aligned} H &= W\tilde{H}W^T, \\ \mathbf{d} &= W^T\tilde{\mathbf{d}}. \end{aligned} \tag{12}$$

5.3 Rank One Systems

Rank one systems are solved similarly to rank two systems. An orthonormal mapping is used such that the two bottom rows of $\tilde{A}$ are zeroed:

$$\begin{bmatrix} \dot{\tilde{x}} \\ \dot{\tilde{y}} \\ \dot{\tilde{z}} \end{bmatrix} = \begin{bmatrix} \tilde{a}_{11} & \tilde{a}_{12} & \tilde{a}_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} + \begin{bmatrix} \tilde{w}_1 \\ \tilde{w}_2 \\ \tilde{w}_3 \end{bmatrix}.$$

The solutions in $\tilde{y}$ and $\tilde{z}$ are both linear in t, and $\tilde{x}$ is a function of $\tilde{\mathbf{p}}(0)$, the exponential of $\tilde{a}_{11}$, and the scalars α and β defined below:

$$\begin{aligned} \tilde{x}(t) &= \left[\tilde{x}(0) + \frac{\tilde{a}_{12}}{\tilde{a}_{11}}\tilde{y}(0) + \frac{\tilde{a}_{13}}{\tilde{a}_{11}}\tilde{z}(0) - \beta\right]e^{\tilde{a}_{11}t} \\ &\quad -\frac{\tilde{a}_{12}}{\tilde{a}_{11}}\tilde{y}(0) - \frac{\tilde{a}_{13}}{\tilde{a}_{11}}\tilde{z}(0) + \alpha t + \beta, \\ \alpha &= -\frac{\tilde{a}_{12}\tilde{w}_2 + \tilde{a}_{13}\tilde{w}_3}{\tilde{a}_{11}}, \\ \beta &= -\frac{\tilde{a}_{11}\tilde{w}_1 + \tilde{a}_{12}\tilde{w}_2 + \tilde{a}_{13}\tilde{w}_3}{\tilde{a}_{11}^2}, \\ \tilde{y}(t) &= \tilde{w}_2 t + \tilde{y}(0), \\ \tilde{z}(t) &= \tilde{w}_3 t + \tilde{z}(0). \end{aligned}$$

Note if $\tilde{a}_{11} = 0$, another orthonormal mapping W would have to be employed to achieve a solution having the above form.

The time-stepping matrix and vector quantities are derived analogously to the rank three and two methods:

$$\tilde{H} = \begin{bmatrix} e^{\tilde{a}_{11}h} & \frac{\tilde{a}_{12}}{\tilde{a}_{11}}(e^{\tilde{a}_{11}h}-1) & \frac{\tilde{a}_{13}}{\tilde{a}_{11}}(e^{\tilde{a}_{11}h}-1) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\tilde{\mathbf{d}} = \begin{bmatrix} (1-e^{\tilde{a}_{11}h})\beta + \alpha h \\ \tilde{w}_2 h \\ \tilde{w}_3 h \end{bmatrix}.$$

These quantities are mapped into physical space using transformations in equation (12).

5.4 Rank Zero

Rank zero systems occur when the four velocities at the vertices are the same, thus resulting in a zero difference matrix. In such a tet, a particle will track parallel to this velocity value, $\mathbf{u}$. Therefore, the time-stepping matrix is simply the identity matrix and the time-stepping vector is $\mathbf{d} = h\mathbf{u}$.

6 Implementation

In this section we discuss four important issues relevant to our implementation of ANTS. The first issue pertains to the determination of the rank of A. We use a two-step process. First we compute the determinant. If $\det(A)$ is sufficiently large with respect to the infinity norm of A (greater than 10^{-12}), we conclude the rank of A is three; otherwise, we compute the singular value decomposition, $A = U_D S V_D^T$. The matrices U_D and V_D are orthonormal, and S is a diagonal matrix of singular values, σ_1, σ_2, and σ_3. The numerical rank is defined to be the number of ratios $\frac{\sigma_1}{\sigma_1}$, $\frac{\sigma_2}{\sigma_1}$, and $\frac{\sigma_3}{\sigma_1}$ greater than machine precision.

Second, the singular value decomposition is also used to implement the rank deficient system coordinate mapping discussed in Section 5. We use U_D^T to map physical space into the appropriate coordinates:

$$\tilde{\mathbf{p}} = U_D^T \mathbf{p}. \tag{13}$$

Substituting into equation (7) gives $\tilde{A} = S V_D U_D^T$.

Third, we consider the treatment of rank three systems that are numerically rank two (and similarly rank two matrices that are numerically rank one). We have performed several tests that involved comparing the rank two time-stepping matrix with the resulting time-stepping matrix for a slightly perturbed, numerically rank three system. We have observed that the difference between the two matrices is consistent with the perturbation. This suggests it is acceptable to approximate solutions of numerically rank two systems with exact rank two solutions. We are currently developing the theory to study this question quantitatively. Note this problem can be side-stepped using SRK4 to compute the time-stepping quantities for numerically rank deficient systems.

Finally, Moler and Van Loan discuss numerous methods for calculating the exponential of a matrix.[3] We prefer the matrix decomposition methods because the decomposition of the matrix A can be performed once and used in the efficient calculation of $\exp(At)$ for all scalar values of t. In particular, we use the Schur decomposition, $A = ZTZ^T$, where Z is an orthonormal matrix and T is quasi-upper triangular. The exponential of At is calculated as follows:

$$e^{At} = Z e^{Tt} Z^T. \tag{14}$$

To compute $\exp(Tt)$, we use an algorithm in which the exponential of the block diagonals of T are calculated explicitly, and the off-diagonals are calculated by using a recursive relation involving the exponentiated block diagonals.[4]

7 Results

The analytic method (ANTS) was compared with two common numerical methods, first order forward Euler (FE) and fourth-order Runge-Kutta (RK4). Critical points of the analysis include the accuracy, computation time, and memory required by the various algorithms.

Given any velocity field $\mathbf{u}(\mathbf{p}(t))$ and initial position $\mathbf{p}_0$, the numerical methods provide an approximate solution in the form of a sequence of points $(\mathbf{p}_i)$ for $i = 0 \ldots N$, with $\mathbf{p}_i \approx \mathbf{p}(t_i)$ for some discrete time t_i. We denote the ith time-step by $h_i = t_i - t_{i-1}$ for $i = 1, \ldots, N$. The two numerical methods FE and RK4 can be defined by the method used to obtain $\mathbf{p}_i$ from $\mathbf{p}_{i-1}$. For Forward Euler,

$$\mathbf{p}_i = \mathbf{p}_{i-1} + h_i \mathbf{u}(\mathbf{p}_{i-1}). \tag{15}$$

Fourth-order Runge-Kutta is defined by

$$\mathbf{p}_i = \mathbf{p}_{i-1} + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4), \tag{16}$$

$$F_1 = h_i \mathbf{u}(\mathbf{p}_{i-1}), \tag{17}$$

$$F_2 = h_i \mathbf{u}(\mathbf{p}_{i-1} + F_1/2), \tag{18}$$

$$F_3 = h_i \mathbf{u}(\mathbf{p}_{i-1} + F_2/2), \quad \text{and} \tag{19}$$
$$F_4 = h_i \mathbf{u}(\mathbf{p}_{i-1} + F_3). \tag{20}$$

The accuracy and stability of the methods are dependent upon the selection of h_i.[1] Our analysis includes both constant time step solutions and time-adapted solutions. The time step was adapted based on a variation of the method proposed by Haimes and Darmofal for steady flow fields[1] in which the velocity tensor was evaluated and used to select a time step that would control the average global error of the result.

Two model flow fields were used to test the methods. The first is similar to that used by several researchers.[1, 5, 6] The velocity field is given by the linear function

$$u_1(x, y, z) = -x - 3y, \tag{21}$$
$$v_1(x, y, z) = -y + 3x, \tag{22}$$
$$w_1(x, y, z) = -z. \tag{23}$$

Provided with initial coordinate (x_0, y_0, z_0), this field has exact streamlines given by

$$x_1(t) = e^{-t}\left(x_0 \cos(3t) - y_0 \sin(3t)\right), \tag{24}$$
$$y_1(t) = e^{-t}\left(x_0 \sin(3t) + y_0 \cos(3t)\right), \tag{25}$$
$$z_1(t) = z_0 e^{-t}. \tag{26}$$

The field has the notable property that it is interpolated exactly on the cells used for our methods. Therefore, any error introduced into the solution is a result of the numerical method. A second field, which is not interpolated exactly, was used to investigate the behavior of ANTS for nonlinear fields and is given by

$$u_2(x, y, z) = -.2\frac{x}{\sqrt{x^2 + y^2}} - 2y, \tag{27}$$
$$v_2(x, y, z) = -.2\frac{y}{\sqrt{x^2 + y^2}} + 2x, \tag{28}$$
$$w_2(x, y, z) = -z. \tag{29}$$

This field has the exact solution

$$x_2(t) = (r_0 - .2t)\cos(\theta_0 + 2t), \tag{30}$$
$$y_2(t) = (r_0 - .2t)\sin(\theta_0 + 2t), \tag{31}$$
$$z_2(t) = z_0 e^{-t}, \quad \text{where} \tag{32}$$
$$r_0 = \sqrt{x_0^2 + y_0^2}, \quad \text{and} \tag{33}$$
$$\theta_0 = \arctan(y_0/x_0). \tag{34}$$

Two measures of accuracy are used to evaluate the methods. The final error is defined to be $E_f = \|\mathbf{p}_N - \mathbf{p}(t_N)\|$. Our definition of average global error,

$$\epsilon = \frac{1}{T_f}\sum_{i=1}^{N} h_i \|\mathbf{p}_i - \mathbf{p}(t^i)\|, \quad \text{where} \tag{35}$$
$$T_f = \sum_{i=1}^{N} h_i, \tag{36}$$

is adapted from that given by Darmofal and Haimes.[1]

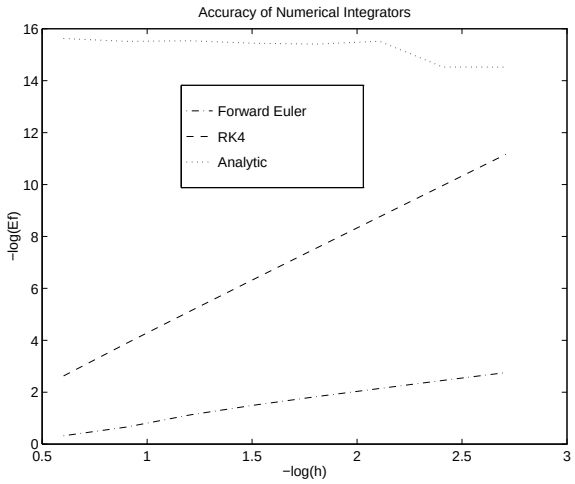

Figure 4: Order of the methods, using field 1

All software was written in C, and the numerical results were obtained on a Silicon graphics workstation with a single, 100 MHz MIPS R4000 processor. A cubic mesh with 1000 cubic cells was decomposed into a tetrahedral mesh. Streamline calculations for field 1 were initiated with $p_0 = (-.5, -.6, -.4)$ and terminated once the time had accumulated beyond 4.0 units. Streamlines for field 2 were initiated with $p_0 = (1.2, .2, .2)$ and terminated at 5.0 time units.

The orders of the methods in terms of the final errors are depicted in figure 4. The numerical methods are as expected, with FE having order 1 and RK4 having order 4. The order of the analytical solution is approximately machine precision at 15. The small reduction in accuracy with smaller time steps can be attributed to cumulative roundoff errors. Figure 5 depicts the calculated streamlines for a large fixed time step on field 1. The time step has been chosen such that the RK4 stability limit is exceeded.[1] The analytic solution remains accurate and stable under these conditions. Figure 6 shows the streamlines as calculated for field 2. Here we see that FE is unstable, RK4 is stable but not very accurate, and ANTS is stable and accurate. In order to provide the full picture, figures 7 and 8 show x and y as functions of

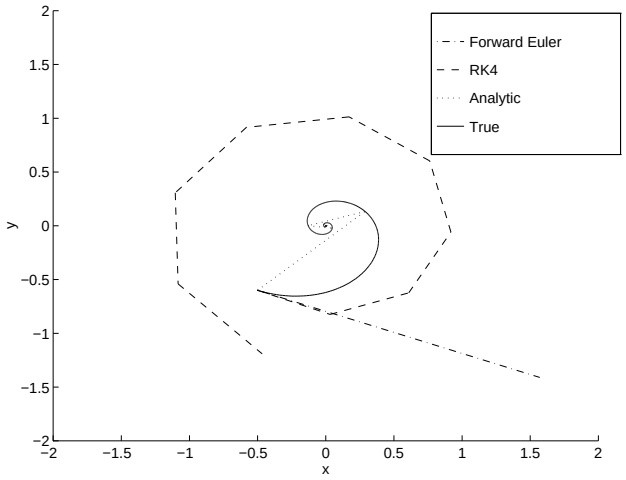

Figure 5: Plots of the solutions with constant $h = .90$ on linear velocity field

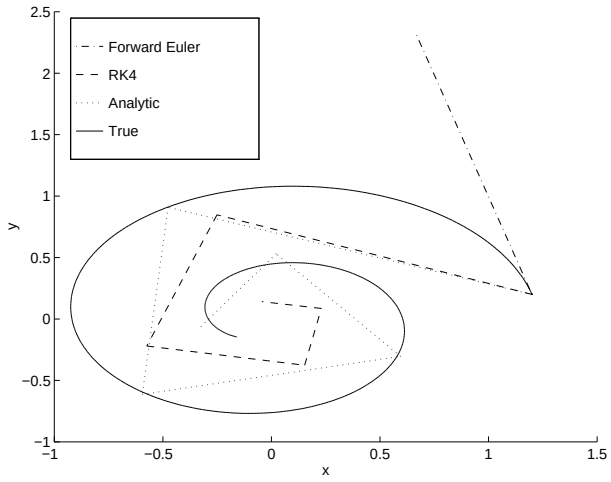

Figure 6: Plots of the solutions with constant $h = .90$ on nonlinear velocity field

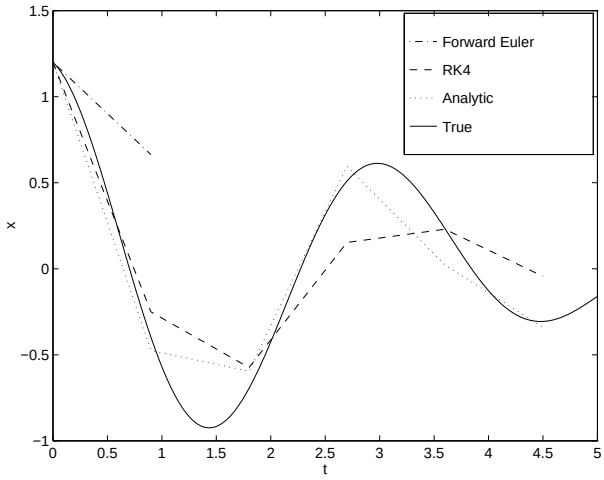

Figure 7: Solution on nonlinear field $h = .90$

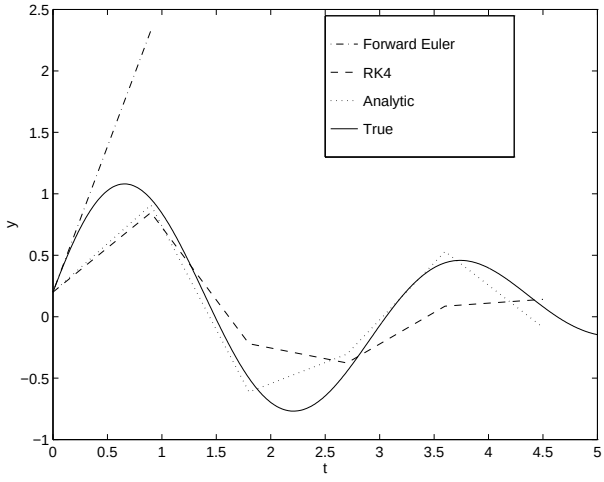

Figure 8: Solution on nonlinear field $h = .90$

t. Note that we restrict our graphs to the x-y plane because these are the more interesting views.

Timing results were obtained by averaging 100 streamline calculations on field 1. The best performance of ANTS is obtained by precomputing H and d for each cell and then using (2) to compute the streamlines. The first set of timing results uses precomputed matrices and involves streamlines computed with a predetermined constant time step. The time required for the ANTS solution was typically .63 and .28 times that required for FE and RK4, respectively. For a mesh with a large number of cells, computer memory may not be available to precompute and store H and $\mathbf{d}$. In this case, the ANTS solution took 2.3 times as long as FE and was approximately equal in speed to RK4.

In the previous timings, all methods completed the same number of time steps and obtained different levels of accuracy. A more meaningful comparison of computational costs was obtained by adapting the time step and requiring that all of the methods satisfy a given bound of the average global error. This takes into account the fact that less accurate methods require smaller, and hence more, time steps to obtain a solution of comparable accuracy. The timing data were obtained by specifying a bound on the average global error and solving a modified form of the time step equation given by Darmofal and Haimes.[1]

The computation time is plotted against the errors obtained by the methods in figure 9. The curve for FE contains only two data points, in or-

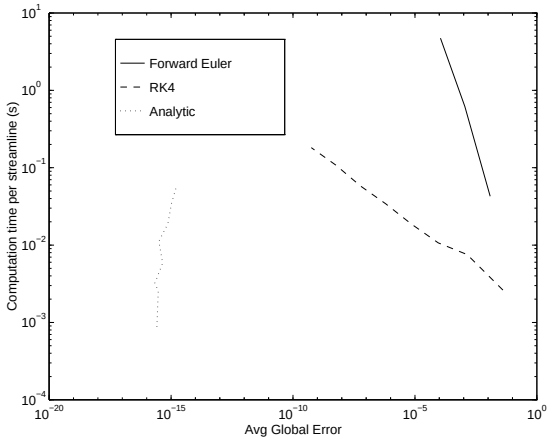

Figure 9: Computation time as a function of average global error requirements.

Table 1: Timings for matrix computations, 1 unit equals the time required for a single $Ax + b$ computation where A is a 3×3 matrix

Matrix	Time (units)
Transform	3.2
A	3.1
A^{-1}	2.0
Schur decomposition	29.5
H and $\mathbf{d}$	10.4

der to preserve the scale on the timing axis. Clearly, however, FE is computationally expensive compared with RK4 and ANTS. In the graph, the ANTS solutions appear to become faster as the tolerance decreases. However, the analytic solutions were computed on the same time steps as the RK4, and the behavior observed corresponds to increases in cumulative roundoff error as the time step is adapted to the error requirement. The ANTS solutions are faster and more accurate than RK4 in this case.

Because precomputed matrices require a large amount of computer memory for storage and improve the speed of the streamline calculations, times required to compute the respective matrices are given in table 1. The times have been normalized by the time required to compute a matrix vector multiplication followed by a vector addition. These timings were obtained by averaging the times required for approximately 30,000 matrix evaluations on field 2. The balance of memory use and computational speed will depend on available resources and priorities for a given application. If the time steps are known at initialization time, it is recommended that H and $\mathbf{d}$ be computed and stored for each cell. If the time step is not known until computation time, precomputation and storage of A, A^{-1}, and the Schur decomposition will speed the calculation of H and $\mathbf{d}$ once a time step is selected.

8 Conclusion

We have taken advantage of the fact that analytic solutions exist for streamlines in a linear velocity field to find analytic solutions to linearly interpolated fields resulting from many flow-solving techniques. The accuracy of the analytical streamlines is approximately machine precision and superior to the results from the common forward Euler and fourth order Runge-Kutta methods. In addition, by precomputing matrices associated with the analytical technique, the streamline calculations require less time than do both numerical methods. Therefore, if storage memory is available, the analytic solver provides faster more accurate results than the two numerical methods tested.

The analytical method also appears to provide stable solutions even when numerical methods have failed. The test case of an inward-spiraling flow illustrated that both of the numerical methods can fail to properly capture the proper flow pattern, whereas the analytical method succeeded.

Acknowledgements

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. The authors would like to thank Paul Plassmann, Carl Ollivier-Gooch, and Lori Freitag for helpful and insightful conversations regarding the work presented in this paper.

References

[1] D. L. Darmofal and R. Haimes. An analysis of 3d particle path integration algorithms. *Journal of Computational Physics*, 123:182–195, 1995.

[2] David N. Kenwright and David A. Lane. Optimization of time-dependent particle tracing using tetrahedral decomposition. In *Proceedings of Visualization '95*, pages 321–327. IEEE Computer Society Press, 1995.

[3] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20:801–832, 1978.

[4] B. N. Parlett. A recurrence among the elements of functions of triangular matrices. *Linear Algebra and Its Applications*, 14:117–121, 1976.

[5] K. Sikorski S. K. Ueng and Kwan-Liu Ma. Fast algorithms for visualizing fluid motion in steady flow on unstructured grids. In *Proceedings of Visualization '95*, pages 313–320. IEEE Computer Society Press, 1995.

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