Journal of Computational Physics **164,** 377–406 (2000)
doi:10.1006/jcph.2000.6602, available online at http://www.idealibrary.com on IDEAL®

Accurate and Efficient Simulation of Rigid-Body Rotations

Samuel R. Buss[1]

Department of Mathematics, University of California, San Diego, La Jolla, California 92093

Received June 14, 1999; revised March 27, 2000

This paper introduces efficient and accurate algorithms for simulating the rotation of a three-dimensional rigid object and compares them to several prior methods. First, we introduce a second-order-accurate method that incorporates a third-order correction; then we introduce a third-order-accurate method; and finally we give a fourth-order-accurate method. These methods are single-step and the update operation is only a single rotation. The algorithms are derived in a general Lie group setting. Second, we introduce a near-optimal energy-correction method which allows exact conservation of energy. This algorithm is faster and easier to implement than implicit methods for exact energy conservation. Our third-order method with energy conservation is experimentally seen to act better than a fourth-order-accurate method. These new methods are superior to naive Runge–Kutta or predictor–corrector methods, which are only second-order accurate for sphere-valued functions. The second-order symplectic McLachlan–Reich methods are observed to be excellent at approximate energy conservation but are not as good at long-term accuracy as our best methods. Finally we present comparisons with fourth-order-accurate symplectic methods, which have good accuracy but higher computational cost. © 2000 Academic Press

Key Words: rotation; rigid body; simulation; energy conservation; stability; Lie algebra; symplectic simulations.

1. INTRODUCTION

In this paper we consider the problem of simulating accurately and efficiently the motion of a three-dimensional, rigid object. We are firstly interested in methods which preserve, or nearly preserve, the physical quantities of angular energy and angular momentum and secondly interested in methods which are highly accurate over a period of time.

For simplicity, this paper considers primarily the situation where the body is moving freely, with no applied forces or torques; however, our methods are stated generally so

[1] Supported in part by NSF Grant DMS-9803515.

as to still apply when forces and torques are present. Our algorithms are designed so that angular momentum is exactly preserved, and therefore the quality of our algorithms will be judged by the two criteria of how well they conserve energy and how accurately they predict orientation. Conservation of energy is a physical law and of course energy should be conserved by any accurate simulation, but conservation of energy is desirable even in simulations where accuracy is not important, since if a simulation conserves energy, then the simulation is guaranteed to be stable.

Section 2 of this paper introduces our notation and reviews the mathematical and physical theory of rigid rotation. In Section 3, we list a variety of methods for simulation of rigid rotation. These methods are loosely characterized as “first-order,” or “second-order,” or “third-order,” or “fourth-order.”[2] Our new simulation methods are essentially Taylor series methods adapted to the nonlinear situation of rigid rotations. One important new aspect of Section 3 is the introduction of additional third-order terms which dramatically improve the performances of a traditional second-order or third-order simulation. The additional third-order term is obtained by an analysis of the effect of time-varying rotation vectors: since the space of rotations is not a Euclidean space, an additional third-order term is introduced. Later (in Section 5) we show how to derive the additional third-order term in the general setting of Lie algebras, and further how to derive higher order methods. Because of the additional third- and higher- order terms, the traditional methods of solving differential equations, such as Runge–Kutta and predictor–corrector methods which are fourth-order correct in a Euclidean space, are only second-order correct for orientation of a rigid body.[3] Section 3 also describes the second-order-accurate McLachlan–Reich explicit symplectic methods and their extensions to fourth-order-accurate methods.

Section 3 concludes with a rough calculation of the relative computational costs of the most effective simulation methods.

Section 4 introduces a method of correcting a simulation to preserve energy. The energy correction is applied after a simulation step (or series of simulation steps) and readjusts the rigid-body orientation to preserve its angular energy. This is done by solving a 3×3 matrix equation for the orientation that restores the previous energy value. The adjusted orientation is obtained by moving at almost right angles to the path of the correct orientation so that very little simulation error is introduced by the energy-correction step.

Section 5 derives our third-order correction term in the setting of Lie algebras. This derives a fourth-order-accurate algorithm as well and is extended to even higher orders. In addition, it means that our higher order (third- and fourth-order) methods can be extended from the rotation group to general Lie groups.

In Section 6 we show the results of experiments. We focus first on energy preservation in the case where the energy correction is never applied. This measures how well the simulation preserve energy and is an important indication of the stability of the simulations. Our results here show the importance of the inclusion of the correct third-order terms, since our “augmented” second- and third-order methods greatly outperform second-order

[2] The order of accuracy refers to the accuracy in orientation: thus “second-order” means that angular velocity is computed with first-order accuracy and orientation is computed with second-order accuracy, etc.

[3] Since preparing the first version of this paper, we have become aware of work by Munthe-Kaas [26, 27], Crouch–Grossman [5], and Marthinsen–Owren [21, 29], who have given third- and fourth-order-accurate implementations of Runge–Kutta algorithms on arbitrary Lie manifolds. We have not carried out experiments with any of these more sophisticated Runge–Kutta algorithms: see the end of Section 3 for our speculative comparisons with our own algorithms.

methods such as Runge–Kutta and predictor–corrector. However, the outstanding performer in terms of energy preservation is the symplectic algorithm of McLachlan–Reich. It is seen to have a very small energy drift even over as many as 200,000,000 simulation steps when doing large rotations in each time step.

In the second part of Section 6, we measure the long-term accuracy of the simulations. First we consider the behavior of the sub-fourth-order-accurate methods. Here our third-order method vastly outperforms the augmented second-order method, and the augmented second-order method significantly outperforms the traditional Runge–Kutta and Adams–Bashforth–Moulton methods. To quantify this, if one considers the trade-off between accuracy and computational effort, then our augmented second-order method is approximately six times more efficient than either Runge–Kutta or Adams–Bashforth–Moulton; whereas the third-order method is yet significantly more accurate. The McLachlan–Reich second-order method showed somewhat mixed results since its long-term accuracy depended heavily on the order in which the axes were considered. For the worst axis ordering, it performed comparably to the other second-order methods, whereas in the optimal axis ordering, the long-term accuracy was like that of a third-order algorithm. Interestingly, the McLachlan–Reich symplectic method usually gained long-term accuracy when combined with our energy-conservation correction (especially with the worst axis ordering). Overall, the best long-term accuracy was obtained by our third-order method with the energy correction applied, which showed better than fourth-order accuracy. The computational cost per step of this method is only slightly worse than the second-order McLachlan–Reich symplectic methods, and they are both a little cheaper than Runge–Kutta methods. Finally, Section 6 considers fourth-order-accurate methods. Here the McLachlan–Reich fourth-order method takes the fewest steps to achieve a given level of accuracy, but it has the downside that 13 separate rotations must be taken per step. The large number of rotations per step takes time and also means additional opportunity for roundoff errors to accrue. The fourth-order method performed quite well either with or without energy preservation, but for some unexplained reason, the third-order method with energy preservation still outperformed the fourth-order method.

The experiments reported in Section 6 confirm that our new third-order and fourth-order methods are correct and achieve third-order and fourth-order (respectively) accuracy. When combined with our energy-preservation method, the third-order method behaves like a better-than-fourth-order-accurate method.

In prior work, a number of authors have considered the problem of accurately simulating rotation and of ensuring stability through energy conservation. Massoud and Youssef [22] considered the general problem of numerically solving first-order sphere-valued differential equations and found that the Runge–Kutta algorithm was not even as good as merely using the average of the current and the next rotation matrix. This method is only second-order accurate and does not even preserve orthogonality of the rotation matrix. A couple of papers have designed energy-conserving algorithms based on Newmark's algorithm: Simo and Wong [33] have given a second-order-accurate algorithm which exactly preserves momentum and energy and Géradin and Rixen [12] have developed a related second-order-accurate algorithm which also preserves energy. These algorithms preserve energy and momentum exactly but have the drawback of being implicit algorithms, which means that they are computationally inefficient since the calculation of a single time step requires an iterative procedure based on Newton's method to find the next configuration. By comparison, our algorithms are quite simple and do not involve any iteration. Simo and Wong [33] also gave an explicit algorithm based on Newmark's algorithm which preserves momentum but not energy. Their best explicit algorithm, ALGO_C1 with $\gamma = 1$ and $\beta = 0$, is compared with our algorithms in Section 6: it is seen that this algorithm is essentially equivalent to our simplest "first-order algorithm," i.e., that it is quite inaccurate and does not preserve energy well. Another approach to simulating rigid-body motion is the symplectic algorithm of Ge and Marsden [11]: their algorithm has long-term stability but is only first-order accurate, does not exactly conserve energy, and has the disadvantage of being implicit. Further implicit symplectic algorithms for rigid-body-rotation have been given by Channell and Scovel [4], Ge [10], Lewis and Simo [19], McLachlan and Scovel [25], and Reich [30, 31]. McLachlan [23] and Reich [31] introduced an explicit symplectic algorithm for rigid-body rotation, which exactly preserves momentum. As discussed above, we compare below the performance of the McLachlan–Reich explicit algorithm to our other algorithms: it generally lives up to the high reputation of symplectic algorithms and, in particular, is excellent at approximately conserving energy over the long term. Dullweber–Leimkuhler–McLachlan [6] report experiments on the accuracy of rigid-body movements for molecular dynamics; they also include comparisons with Runge–Kutta style algorithms. Holder–Leimkuhler–Reich [15] give another symplectic algorithm for rigid-body rotation which is time-symmetric with variable step size.

For a survey of symplectic algorithms, the reader can refer to Channell and Neri [3]. According to them, symplectic algorithms frequently do not have good accuracy or exact energy preservation, but they do often succeed in preserving the global, long-term behavior of a system. Theoretical reasons for the long-term stability of symplectic algorithms have been given by [1, 14, 32]. There are various methods of extending explicit symplectic methods to higher order explicit symplectic methods [3, 35, 24]. Using one of these methods, we implemented and tested a fourth-order version of the McLachlan–Reich symplectic method. It is compared with our third- and fourth-order methods in Section 6.

2. THE THEORY OF 3D ROTATIONS

This section reviews the basic theory of 3D rotations and establishes notation. We include a description of Poinsot's inertial ellipsoid since this will greatly help our intuition about the accuracy of our algorithms and will form the basis for our energy-preserving algorithm. The contents of this section are standard and well known (see, e.g., [13] or [34]).

Rotation matrices and vectors. Since translational motion may be decoupled from orientation and rotation, we will be interested in only the orientation and rotational motion of a rigid body. We presume that we have a rigid body which is changing orientation over time according to some fixed trajectory and that the center of mass is fixed while the body's orientation changes as a function of time. There are two frames of reference: the *world (or spatial) coordinate system*, which is a fixed frame of reference, and the *body coordinate system*, which is a frame of reference attached to the body and which moves with the body. The two frames of reference have a common origin which is presumed to be at the center of mass of the body. The orientation of the body at time t can be specified with an orientation matrix $\Omega = \Omega(t)$ such that if $\boldsymbol{v}$ is a vector specified in body coordinates, then $\Omega\boldsymbol{v}$ is the vector expressed in world coordinates. We have $\Omega^{-1} = \Omega^T$, where the superscript ^{T} indicates the transpose.

A rotation vector $\boldsymbol{w}$ is a vector which specifies an action of rotation of $\|\boldsymbol{w}\|$ radians around the axis $\boldsymbol{w}$ with the direction of rotation specified by the right-hand rule. It is known that every orientation matrix corresponds to a rotation vector (see [12] for a comprehensive survey of various representations of orientation and rotations); and we write $R_{\boldsymbol{w}}$ for the orientation matrix which corresponds to the rotation vector $\boldsymbol{w}$.

Rotation vectors can be used to express instantaneous angular velocity: in fact, there is a vector $\boldsymbol{\omega} = \boldsymbol{\omega}(t)$ such that if $\mathbf{x}$ is a point on the body specified in spatial coordinates, then the velocity of point $\mathbf{x}$ is given by

$$\dot{\boldsymbol{x}} = \boldsymbol{\omega} \times \boldsymbol{x}.$$

This is proved as follows: let $\boldsymbol{X} = \Omega^{-1}\boldsymbol{x} = \Omega^T\boldsymbol{x}$ be the (constant) vector specifying the point $\boldsymbol{x}$ in body coordinates. Then $\dot{\boldsymbol{x}} = \dot{\Omega}\boldsymbol{X} = \dot{\Omega}\Omega^T\boldsymbol{x}$, since $\boldsymbol{X}$ is constant. The matrix $\dot{\Omega}\Omega^T$ is skew-symmetric, since

$$\frac{d}{dt}(\Omega\Omega^T) = \dot{\Omega}\Omega^T + \Omega\dot{\Omega}^T = \dot{\Omega}\Omega^T + (\dot{\Omega}\Omega^T)^T = 0.$$

Now, if $\boldsymbol{v} = \langle v_1, v_2, v_3\rangle$ and $\boldsymbol{u}$ is an arbitrary vector, then $\boldsymbol{v} \times \boldsymbol{u} = \tilde{\boldsymbol{v}}u$, where $\tilde{\boldsymbol{v}}$ is the matrix

$$\tilde{\boldsymbol{v}} = \begin{pmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{pmatrix}.$$

It follows that every skew-symmetric matrix, in particular the matrix $\dot{\Omega}\Omega^T$, has an associated rotation vector. We let $\boldsymbol{\omega} = \boldsymbol{\omega}(t)$ be the vector associated with $\dot{\Omega}\Omega^T$; i.e., $\boldsymbol{\omega}(t)$ is the instantaneous rotation vector at time t, and $\dot{\boldsymbol{x}} = \boldsymbol{\omega} \times \boldsymbol{x}$ for any point $\boldsymbol{x}$ fixed on the body.

In the previous paragraph, $\boldsymbol{X}$ was a constant vector (in body coordinates). When $\boldsymbol{X}$ varies with time and $\boldsymbol{x} = \Omega\boldsymbol{X}$, then the time derivative of x in spatial coordinates is

$$\dot{\boldsymbol{x}} = \dot{\Omega}\boldsymbol{X} + \Omega\dot{\boldsymbol{X}} = \boldsymbol{\omega} \times \Omega\boldsymbol{X} + \Omega\dot{\boldsymbol{X}} = \boldsymbol{\omega} \times \boldsymbol{x} + \Omega\dot{\boldsymbol{X}}. \tag{1}$$

Arguing dually, we have

$$\frac{d}{dt}(\Omega^{-1}\boldsymbol{x}) = \dot{\boldsymbol{X}} = -(\Omega^{-1}\boldsymbol{\omega}) \times \boldsymbol{X} + \Omega^{-1}\dot{\boldsymbol{x}} = \Omega^{-1}(-\boldsymbol{\omega} \times \boldsymbol{x} + \dot{\boldsymbol{x}}). \tag{2}$$

The inertia matrix, momentum, and energy. For translational motion, the fundamental equations of motion are that $\boldsymbol{p} = m\boldsymbol{v}$ and $E = \frac{1}{2}mv^2$, where $\boldsymbol{p}$, m, $\boldsymbol{v}$, and E are momentum, mass, velocity, and energy, respectively, and $v^2 = \boldsymbol{v} \cdot \boldsymbol{v} = \|v\|^2$. The physics of rotational motion is analogous but more complicated. Most notably, the rotational analogue of the scalar mass is a 3×3 matrix I called the *inertia matrix* (or "inertia tensor"). The matrix I changes with the body's orientation: if we let J denote the inertia matrix expressed in body coordinates, then J is constant and $I = \Omega J\Omega^{-1}$. It is known that J is positive definite and self-adjoint (Hermitian) and, in particular, it is possible to choose the body coordinate system so that J is a diagonal matrix with all entries on the diagonal positive. Likewise, I is real, positive definite, and symmetric. The self-adjointness (or the fact that J is diagonal) implies that $\boldsymbol{u} \cdot (I\boldsymbol{v}) = (I\boldsymbol{u}) \cdot \boldsymbol{v}$, for all vectors $\boldsymbol{u}$, $\boldsymbol{v}$, where "$\cdot$" denotes the dot product.

The angular momentum is denoted $\boldsymbol{L}$ and is constant unless external torques are applied. If $\boldsymbol{T}$ is the applied external torque then $\dot{\boldsymbol{L}} = \boldsymbol{T}$. The fundamental equation of angular motion for rigid bodies is

$$\boldsymbol{L} = I\boldsymbol{\omega}. \tag{3}$$

This implies $\boldsymbol{\omega} = I^{-1}\boldsymbol{L}$, and so the instantaneous rate of rotation can be determined from knowledge of $\boldsymbol{L}$. In the situation where there are no externally applied torques, $\boldsymbol{L}$ is constant; however, $\boldsymbol{\omega}$ is not constant in general since the inertia matrix changes with the body's orientation. This corresponds to the fact that spinning objects will be observed to wobble when the rotation vector is not an eigenvector of the inertia matrix.

Taking the time derivative of Eq. (3) using the chain rule and Eqs. (1) and (2), we obtain Euler's equation for the derivative of the momentum:

$$\begin{aligned}\dot{\boldsymbol{L}} &= \frac{d}{dt}(I\boldsymbol{\omega}) = \frac{d}{dt}(\Omega J\Omega^{-1}\boldsymbol{\omega})\\ &= \boldsymbol{\omega} \times (\Omega J\Omega^{-1}\boldsymbol{\omega}) + \Omega J\Omega^{-1}(-\boldsymbol{\omega}\times\boldsymbol{\omega} + \dot{\boldsymbol{\omega}})\\ &= \boldsymbol{\omega} \times (I\boldsymbol{\omega}) + I\dot{\boldsymbol{\omega}}\\ &= \boldsymbol{\omega} \times \boldsymbol{L} + I\dot{\boldsymbol{\omega}}.\end{aligned} \tag{4}$$

Solving for $\dot{\boldsymbol{\omega}}$ yields

$$\dot{\boldsymbol{\omega}} = I^{-1}(\dot{\boldsymbol{L}} - \boldsymbol{\omega} \times I\boldsymbol{\omega}) = I^{-1}(\dot{\boldsymbol{L}} - \boldsymbol{\omega} \times \boldsymbol{L}). \tag{5}$$

Taking the time derivative of Eq. (4), and again using the chain rule and Eqs. (1) and (2), we obtain a formula for the second derivative of momentum:

$$\ddot{\boldsymbol{L}} = \dot{\boldsymbol{\omega}} \times \boldsymbol{L} + \boldsymbol{\omega} \times \dot{\boldsymbol{L}} + \boldsymbol{\omega} \times I\dot{\boldsymbol{\omega}} - I(\boldsymbol{\omega} \times \dot{\boldsymbol{\omega}}) + I\ddot{\boldsymbol{\omega}}. \tag{6}$$

Using Eq. (4) to rewrite the third term gives

$$\ddot{\boldsymbol{L}} = \dot{\boldsymbol{\omega}} \times \boldsymbol{L} + 2\boldsymbol{\omega} \times \dot{\boldsymbol{L}} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{L}) - I(\boldsymbol{\omega} \times \dot{\boldsymbol{\omega}}) + I\ddot{\boldsymbol{\omega}}. \tag{7}$$

Solving for $\ddot{\boldsymbol{\omega}}$ gives

$$\ddot{\boldsymbol{\omega}} = \boldsymbol{\omega} \times \dot{\boldsymbol{\omega}} + I^{-1}(\ddot{\boldsymbol{L}} - \dot{\boldsymbol{\omega}} \times \boldsymbol{L} - 2\boldsymbol{\omega} \times \dot{\boldsymbol{L}} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{L})). \tag{8}$$

Differentiating (8) and simplifying gives the third derivative of rotation as

$$\begin{aligned}\dddot{\boldsymbol{\omega}} = 2\boldsymbol{\omega} \times \ddot{\boldsymbol{\omega}} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \dot{\boldsymbol{\omega}}) + I^{-1}[\dddot{\boldsymbol{L}} - 3\boldsymbol{\omega} \times \ddot{\boldsymbol{L}} - 3\dot{\boldsymbol{\omega}} \times \dot{\boldsymbol{L}} - \ddot{\boldsymbol{\omega}} \times \boldsymbol{L} + \dot{\boldsymbol{\omega}} \times (\boldsymbol{\omega} \times \boldsymbol{L})\\ + 2\boldsymbol{\omega} \times (\dot{\boldsymbol{\omega}} \times \boldsymbol{L}) + 3\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \dot{\boldsymbol{L}}) - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{L}))].\end{aligned} \tag{9}$$

Equations (5), (8), and (9) will appear as higher order terms in our simulation algorithms in Section 3.

The angular kinetic energy of a rigid body can be defined as

$$E = \frac{1}{2}\boldsymbol{L} \cdot \boldsymbol{\omega}. \tag{10}$$

An alternative definition of energy can be given as follows. For nonzero $\boldsymbol{\omega}$, let $\boldsymbol{n}$ be the unit vector $\boldsymbol{n} = \boldsymbol{\omega}/\|\boldsymbol{\omega}\|$ in the same direction as $\boldsymbol{\omega}$. The angular inertia around the instantaneous axis of rotation is defined as $\mathcal{I} = \boldsymbol{n} \cdot I\boldsymbol{n}$. Note that $\mathcal{I}$ is a scalar and a function of time. Then clearly the energy E is equal to $\frac{1}{2}\mathcal{I}\|\boldsymbol{\omega}\|^2$, which is a familiar formula from the case of a constant rotation axis.

For our purposes, the important aspect of energy is that it is constant in the absence of external torques. To prove this, take the derivative of Eq. (10) to get

$$\begin{aligned}\dot{E} &= \frac{1}{2}(\boldsymbol{L} \cdot \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \cdot \dot{\boldsymbol{L}}) = \frac{1}{2}((I\boldsymbol{\omega}) \cdot (I^{-1}(\dot{\boldsymbol{L}} - \boldsymbol{\omega} \times \boldsymbol{L})) + \boldsymbol{\omega} \cdot \dot{\boldsymbol{L}}) \\ &= \frac{1}{2}(\boldsymbol{\omega} \cdot (\dot{\boldsymbol{L}} - \boldsymbol{\omega} \times \boldsymbol{L}) + \boldsymbol{\omega} \cdot \dot{\boldsymbol{L}}) = \boldsymbol{\omega} \cdot \dot{\boldsymbol{L}} - \frac{1}{2}\boldsymbol{\omega} \cdot (\boldsymbol{\omega} \times \boldsymbol{L}) = \boldsymbol{\omega} \cdot \dot{\boldsymbol{L}},\end{aligned} \tag{11}$$

where we have used Eq. (5) and the self-adjointness of I. When there are no external torques, $\dot{\boldsymbol{L}} = \boldsymbol{0}$, and the energy is constant.

The inertial ellipsoid. Poinsot's inertial ellipsoid is an ellipsoid rigidly attached to the rigid body that provides a good qualitative description of the orientation and angular velocity of the body. It is convenient to work mostly in body coordinates, especially because the inertial matrix J in body coordinates is a constant, diagonal matrix. Accordingly, let $\boldsymbol{\nu} = \Omega^{-1}\boldsymbol{\omega}$ be the instantaneous rotation vector expressed in body coordinates. Define the vector $\boldsymbol{\rho}$ as $\boldsymbol{\rho} = \boldsymbol{\nu}/(\|\boldsymbol{\nu}\|\sqrt{\mathcal{I}})$. The direction of $\boldsymbol{\rho}$ is the same as $\boldsymbol{\nu}$, and its magnitude depends on only its direction. Of course, $\boldsymbol{n} = \sqrt{\mathcal{I}}\,\Omega\boldsymbol{\rho}$ and therefore

$$\boldsymbol{\rho} \cdot J\boldsymbol{\rho} = 1. \tag{12}$$

If we let J's diagonal elements be J_{11}, J_{22}, J_{33} and $\boldsymbol{\rho} = \langle \rho_1, \rho_2, \rho_3 \rangle$, Eq. (12) states $J_{11}\rho_1^2 + J_{22}\rho_2^2 + J_{33}\rho_3^2 = 1$. Therefore the set of $\boldsymbol{\rho}$'s satisfying $\boldsymbol{\rho} \cdot J\boldsymbol{\rho} = 1$ is an ellipsoid, called the *inertial ellipsoid*.

Let $F(\boldsymbol{\rho}) = \boldsymbol{\rho} \cdot J\boldsymbol{\rho}$. The outward normal to the inertial ellipsoid at the point $\boldsymbol{\rho}$ is in the direction of the gradient, $\nabla F(\boldsymbol{\rho})$, of F at $\boldsymbol{\rho}$. Expressing the gradient in body coordinates, we have

$$\nabla F = \langle 2J_{11}\rho_1, 2J_{22}\rho_2, 2J_{33}\rho_3 \rangle = 2J\boldsymbol{\rho} \tag{13}$$

or, letting $\boldsymbol{L}_{\mathrm{bd}} = \Omega^{-1}\boldsymbol{L}$ be the angular momentum in body coordinates,

$$\nabla F = 2\Omega^{-1}I\Omega\boldsymbol{\rho} = \frac{2}{\|\boldsymbol{\omega}\|\sqrt{\mathcal{I}}}\Omega^{-1}\boldsymbol{L} = \frac{2}{\|\boldsymbol{\omega}\|\sqrt{\mathcal{I}}}\boldsymbol{L}_{\mathrm{bd}} = \sqrt{\frac{2}{E}}\boldsymbol{L}_{\mathrm{bd}}. \tag{14}$$

Thus the outward normal of the surface $F = 1$ at $\boldsymbol{\rho}$ is in the direction of the momentum vector.

Finally, since

$$\boldsymbol{\rho} \cdot \boldsymbol{L}_{\mathrm{bd}} = \frac{\boldsymbol{\omega} \cdot \boldsymbol{L}}{\|\boldsymbol{\omega}\|\sqrt{\mathcal{I}}} = \frac{2E}{\sqrt{2E}} = \sqrt{2E}, \tag{15}$$

the vector $\boldsymbol{\rho}$ must lie in the plane perpendicular to the momentum vector consisting of points whose dot product with $\boldsymbol{L}_{\mathrm{bd}}$ equals $\sqrt{2E}$: this plane is called the *invariable plane*. Under the assumption of no externally applied torques, E and $\boldsymbol{L}$ are constant (in spatial coordinates), and so the invariable plane is a spatially fixed, unvarying plane.

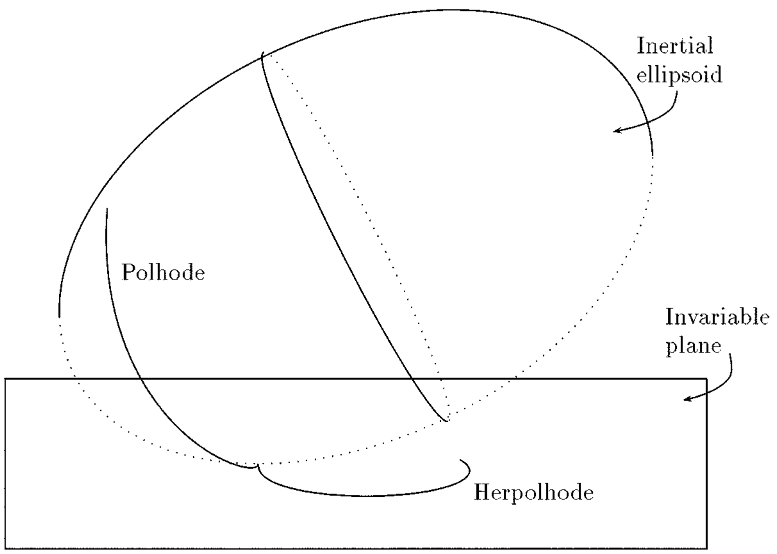

FIG. 1. The inertial ellipsiod, the invariable plane, the polhode, and the herpolhode. The angular momentum is downward, so the invariable plane is horizontal. The herpolhode is a curve traced out on the invariable plane, and the polhode is a curve on the surface of the inertial ellipsoid.

Therefore the Poinsot construction has established so far that at any given instant in time, the inertial ellipsoid is tangent to the invariable plane at the point $\boldsymbol{\rho}$ and the rotation axis passes through the point $\boldsymbol{\rho}$. Assuming there are no external torques, this means that the inertial ellipsoid rolls along the invariable plane without slipping. The vector $\boldsymbol{\rho}$ traces out a closed curve (loop) on the inertial ellipsoid, called the *polhode*. It also traces out a curve on the invariable plane; this curve is called the *herpolhode*.[4] These features are illustrated in Fig. 1.

There is one further fact about the polhode that we need for our constructions in Section 4; namely, the polhode is equal to the intersection of the inertial ellipsoid with a second ellipsoid which is also fixed in the body frame of reference. Equation (14) implies that the points $\boldsymbol{\rho}$ on the polhode satisfy $\|\nabla F(\boldsymbol{\rho})\|^2 = 2\|\boldsymbol{L}\|^2/E$. From Eq. (13), we compute

$$\|\nabla F(\boldsymbol{\rho})\|^2 = 4J_{11}^2\rho_1^2 + 4J_{22}^2\rho_2^2 + 4J_{33}^2\rho_3^2 = \frac{2\|\boldsymbol{L}\|^2}{E}. \tag{16}$$

The last equation defines an ellipsoid of course and since $\boldsymbol{\rho}$ is in body coordinates, this ellipsoid is fixed in the body frame of reference. Therefore the polhode is the intersection of two ellipsoids fixed in the body frame; or more precisely, the polhode is one of the two connected components of the intersection of the ellipsoids.

The intuition of the inertial ellipse rolling on the invariable plane will give us good insight into why certain simulation methods work well or poorly. Furthermore, in Section 4 we will use the polhode and invariable plane and the characterization of the polhode as the intersection of two ellipsoids to develop a method for preserving energy during simulation of rotation.

[4] As Goldstein [13] remarks, this gives rise to the Jabberwockian statement: the polhode rolls without slipping on the herpolhode lying in the invariable plane.

3. SIMULATION METHODS

Our algorithms for simulation of the rotation of a rigid body use discrete time steps to calculate the orientation and momentum of the body at times $t_1 < t_2 < t_3 < \cdots$. Our goal is to compute, at each instant of time, the orientation matrix $\Omega_i = \Omega(t_i)$. We assume that the angular momentum $\boldsymbol{L}$ is given exogenously; that is to say, the angular momentum (and sometimes its derivatives) is given as an input and we do not compute it. For the experiments reported in Section 6, there are no external torques and the angular momentum $\boldsymbol{L}$ is constant and $\dot{\boldsymbol{L}} = \ddot{\boldsymbol{L}} = \mathbf{0}$. In more general applications, the angular momentum will change with applied torques and impulses—in some applications, the applied torques will depend on the angular velocity or the angular acceleration; however, in any event, the computation of applied torques is application dependent and beyond the scope of the present paper. We write $\boldsymbol{L}_i, \dot{\boldsymbol{L}}_i, \ddot{\boldsymbol{L}}_i$, etc. for the externally given angular momentum and its derivatives at time t_i.

Under the assumption that the angular momentum is known, then computing the orientation Ω_i at time i is sufficient to fully specify the rest of the parameters of the motion of the rigid body, including angular velocity and angular acceleration, by use of Eqs. (5) and (8). Since the body is rigid, the inertia matrix in body coordinates is a constant matrix J, and in world coordinates, the inertia matrix at time t_i is $I_i = \Omega_i J \Omega_i^{-1}$.

All the algorithms given below compute Ω_{i+1} from Ω_i by first computing a “average rotation” vector $\bar{\boldsymbol{\omega}}$ and then setting

$$\Omega_{i+1} = R_{h\bar{\omega}}\Omega_i,$$

where $h = t_{i+1} - t_i$ is the time increment and $R_{h\bar{\omega}}$ is the rotation matrix which corresponds to the rotation vector $h\bar{\boldsymbol{\omega}}$, a rotation around the axis $\bar{\boldsymbol{\omega}}$ of $h\|\bar{\boldsymbol{\omega}}\|$ radians. The algorithms presented below do not require that the time intervals $t_{i+1} - t_i$ are all equal, with the two exceptions of the Adams–Bashforth–Moulton predictor–corrector method and the Simo–Wong method. The latter method is easily modified to handle unequal time intervals.

The first-order method. We first discuss the (poorly performing) first-order method. For this method, we just calculate the instantaneous velocity at time t_i according to Eq. (3) and apply this velocity during the entire time step:

THE FIRST-ORDER METHOD

```
$\bar{\boldsymbol{\omega}} = \boldsymbol{\omega}_i = I_i^{-1}\boldsymbol{L}_i$.
Update orientation as $\Omega_{i+1} := R_{h\bar{\omega}}\,\Omega_i$.
```

It is easy to qualitatively analyze the behavior of the first-order method with the aid of the inertial ellipsoid. Let us view the momentum as pointing straight down (in spatial terms): then the invariable plane is horizontal and at time t_i the inertial ellipsoid is above and tangent to the invariable plane, P_i, at a point with body coordinates $\boldsymbol{\rho}_i$. We call this point the “lowest point” on the inertial ellipsoid. The rotation vector $\boldsymbol{\omega}_i$ passes through this lowest point. When the inertial ellipsoid is rotated around this rotation vector, then clearly the new lowest point on the inertial ellipsoid will be below the original invariable plane P_i.

Therefore, after one step of the first-order method, the next time step will find that the lowest point on the inertial ellipsoid is lower than the previous lowest point. That is to say, $\boldsymbol{\rho}_{i+1} \cdot \boldsymbol{L}_{\text{bd}} > \boldsymbol{\rho}_i \cdot \boldsymbol{L}_{\text{bd}}$, under the assumption of constant momentum. By Eq. (15), $E = (\boldsymbol{\rho} \cdot \boldsymbol{L}_{\text{bd}}/2)^2$, and this means that the energy has increased from one time step to the next.

Therefore, the first-order method will cause the energy to increase cumulatively and monotonically; and the experiments below show that this increase in energy can be quite quick and dramatic, even for relatively small rotation increments. Therefore the first-order method is almost never an appropriate method, except in applications where the rotation increment is very small or where accuracy and energy conservation are unimportant.

The second-order method. To improve on the poorly performing first-order method, we try a second-order method. The average rotation is now estimated as $\bar{\boldsymbol{\omega}} := \boldsymbol{\omega} + \frac{h}{2}\dot{\boldsymbol{\omega}}$, where the instantaneous rate of change of the rotation vector is calculated using Eq. (5).

THE SECOND-ORDER METHOD

$\boldsymbol{\omega}_i = I_i^{-1}\boldsymbol{L}_i$.
$\dot{\boldsymbol{\omega}}_i = I_i^{-1}(\dot{\boldsymbol{L}}_i - \boldsymbol{\omega}_i \times \boldsymbol{L}_i)$.
Let $\bar{\boldsymbol{\omega}} := \boldsymbol{\omega}_i + \frac{h}{2}\dot{\boldsymbol{\omega}}_i$.
Update orientation as $\Omega_{i+1} := R_{h\bar{\omega}}\,\Omega_i$.

In terms of computational effort, the second-order method is only slightly slower than the first-order method.

The experiments below show that the second-order method is substantially better than the first-order method. However, the experiments still reveal a steady and monotonic increase in the energy, and we therefore seek yet better methods.

The false third-order method. The obvious next method to try is a third-order method based on the second time derivative of the angular velocity. We include this method for completeness sake; however, the analysis of the "augmented second-order method" shows that this method is not truly third order at all. This is confirmed by the experiments below, which show that the false third-order method is not much better than the second-order method.

FALSE THIRD-ORDER METHOD

$\boldsymbol{\omega}_i = I_i^{-1}\boldsymbol{L}_i$.
$\dot{\boldsymbol{\omega}}_i = I_i^{-1}(\dot{\boldsymbol{L}}_i - \boldsymbol{\omega}_i \times \boldsymbol{L}_i)$.
$\ddot{\boldsymbol{\omega}} = \boldsymbol{\omega}_i \times \dot{\boldsymbol{\omega}}_i + I_i^{-1}(\ddot{\boldsymbol{L}}_i - \dot{\boldsymbol{\omega}}_i \times \boldsymbol{L}_i - 2\boldsymbol{\omega}_i \times \dot{\boldsymbol{L}}_i + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{L}_i))$.
Let $\bar{\boldsymbol{\omega}} := \boldsymbol{\omega} + \frac{1}{2}\dot{\boldsymbol{\omega}}h + \frac{1}{6}\ddot{\boldsymbol{\omega}}h^2$.
Update orientation as $\Omega_{i+1} := R_{h\bar{\omega}}\,\Omega_i$.

Augmented second-order method. The justification for the second-order method and the false third-order method above is based on using the derivatives of the rotation vector $\boldsymbol{\omega}$ to estimate the average rotation vector during the current time interval. However, since rotation operators are not commutative, the average of the rotation vectors is not the equal to the "effective" rotation vector: that is to say, merely taking the average of the applied rotation vectors does not yield a rotation vector which correctly gives the overall rotation of the body in the next time interval. Instead, one must also account for the order in which the rotations are applied.

We give a simple calculation which illustrates this point and then will use this to obtain an adjustment to the rotation vector which is second-order accurate. Suppose that a disk is rolling along a line. The motion of the disk can be described as a series of infinitesimal rotations: if the disk starts at position p on the line and rolls at velocity v, then at time t, the disk is rotating at a rate of ω radians per second around the point at position $p + vt$. Thus, the

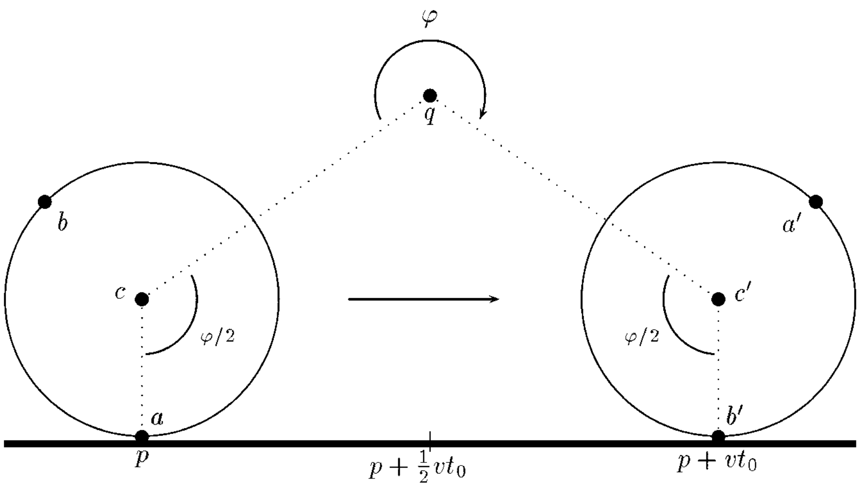

FIG. 2. The starting and stopping position of a disk which has rolled along the line. The rolling motion is equivalent to rotation by angle φ around the point q. The points a, b are fixed points on the circumference of the disk; the point c is the center of the disk. The positions of the points after the disk has finished rolling are labeled a', b', c'.

line along which the disk is rolling is also the line through which the instantaneous rotations occur. After time t_0 has elapsed, the disk has moved distance vt_0 and rotated $\varphi = \omega t_0$ radians, where $\omega = v/r$. The overall motion of the disk during this time is equivalent to a rotation of ωt_0 radians around a fixed point q as illustrated in Fig. 2. This q is an "effective average" of the instantaneous rotations even though it does not lie on the line of instantaneous rotations.

In Fig. 2, the disk has moved a considerable distance, so the point q is high above the line of instantaneous rotations. We will be considering the net effect of rotations over a short period of time, and in this case, q will be seen to be close to the line of instantaneous rotations. We wish to give a second-order-accurate approximation for the position of the point q. First, by symmetry, the point q must lie above the midpoint $p + \frac{1}{2}vt_0$ of the line. In fact, a first-order-accurate approximation for q is simply $q \approx p + \frac{1}{2}vt_0$.

Let h be the height of q above the line. From geometric considerations, if we let c and c' be the starting and ending positions of the center of the disk, the lines $\overline{cq}$ and $\overline{c'q}$ both make an angle of $\varphi/2$ with the vertical (refer to Fig. 2). Therefore,

$$\tan(\varphi/2) = \frac{(1/2)vt_0}{r-h} = \frac{r\theta}{2(r-h)}.$$

Using the approximation $\tan(\varphi/2) \approx \varphi/2 + \varphi^3/24$, and solving for h, we obtain

$$h \approx \frac{\varphi^2 r}{12 - \varphi^2} \approx \frac{1}{12}\varphi^2 r.$$

So, the position of q is approximately a height of $(1/12)\varphi^2 r$ above the midpoint of the line. This estimate for h and q is clearly (better than) second-order accurate.

To transfer our calculation of q's position to the setting of 3D rotations, consider the situation where the rigid body is undergoing an instantaneous rotation given by rotation vector $\boldsymbol{\omega}$ and its rate of change has been computed as $\dot{\boldsymbol{\omega}}$. We wish to compute the motion of the body over time interval h as a pure rotation based on a rotation vector $\bar{\boldsymbol{\omega}}$. In the second-order and the false third-order methods we approximated this with the rotation vector $\boldsymbol{\omega} + h\dot{\boldsymbol{\omega}}/2$. When $\dot{\boldsymbol{\omega}}$ is parallel to $\boldsymbol{\omega}$, this is fine; but in the situation where $\dot{\boldsymbol{\omega}}$ is perpendicular to $\boldsymbol{\omega}$, we can give a better approximation by modeling the rotation with a planar disk which is rolling with a total rotation of $h\|\boldsymbol{\omega}\|$ traveling a total distance $h\|\dot{\boldsymbol{\omega}}\|$. According to the analysis of a rolling plane, the net rotation should be approximated by using $\boldsymbol{\omega} + h\dot{\boldsymbol{\omega}}/2$ plus a lateral displacement of $\frac{1}{12}h\|\boldsymbol{\omega}\| \cdot h\|\dot{\boldsymbol{\omega}}\|$ (under the assumption that $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\omega}}$ are perpendicular). To simultaneously handle the components of $\dot{\boldsymbol{\omega}}$ perpendicular to and parallel to $\boldsymbol{\omega}$, one can use

$$\bar{\boldsymbol{\omega}} = \boldsymbol{\omega} + \frac{h}{2}\dot{\boldsymbol{\omega}} + \frac{h^2}{12}\dot{\boldsymbol{\omega}} \times \boldsymbol{\omega}.$$

Note how the cross-product neatly handles only the perpendicular components and correctly chooses the displacement direction.

We still need to verify that modeling the 3D rotations in terms of rolling in the plane does not introduce additional significant error and that the above approximation for $\bar{\boldsymbol{\omega}}$ is second-order accurate. This could be proved using a more careful proof than we gave above; however, we shall not do this. Instead we shall obtain, in Section 5, a second, more general proof of the above approximation based on general Lie algebras.

We thus have derived the following algorithm incorporating the third-order energy-correction term:

AUGMENTED SECOND-ORDER METHOD

```
$\boldsymbol{\omega}_i = I_i^{-1}\boldsymbol{L}_i.$
$\dot{\boldsymbol{\omega}}_i = I_i^{-1}(\dot{\boldsymbol{L}}_i - \boldsymbol{\omega}_i \times \boldsymbol{L}_i).$
Let $\bar{\boldsymbol{\omega}} := \boldsymbol{\omega}_i + \frac{h}{2}\dot{\boldsymbol{\omega}}_i + \frac{h^2}{12}(\dot{\boldsymbol{\omega}}_i \times \boldsymbol{\omega}_i).$
Update orientation as $\Omega_{i+1} := R_{h\bar{\boldsymbol{\omega}}}\,\Omega_i.$
```

The experiments reported below confirm that the inclusion of the third-order term significantly improves conservation of energy. The computational complexity of the augmented second-order method is quite good: it requires only one more cross-product calculation than the earlier second-order method.

True third-order method. By incorporating the above third-order correction, the correct third-order simulation algorithm is obtained:

TRUE THIRD-ORDER METHOD

```
$\boldsymbol{\omega}_i = I_i^{-1}\boldsymbol{L}_i.$
$\dot{\boldsymbol{\omega}}_i = I_i^{-1}(\dot{\boldsymbol{L}}_i - \boldsymbol{\omega}_i \times \boldsymbol{L}_i).$
$\ddot{\boldsymbol{\omega}} = \boldsymbol{\omega}_i \times \dot{\boldsymbol{\omega}}_i + I_i^{-1}(\ddot{\boldsymbol{L}}_i - \dot{\boldsymbol{\omega}}_i \times \boldsymbol{L}_i - 2\boldsymbol{\omega}_i \times \dot{\boldsymbol{L}}_i + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{L}_i)).$
Let $\bar{\boldsymbol{\omega}} := \boldsymbol{\omega}_i + \frac{h}{2}\dot{\boldsymbol{\omega}}_i + \frac{h^2}{6}\ddot{\boldsymbol{\omega}}_i + \frac{h^2}{12}((\dot{\boldsymbol{\omega}}_i + \frac{h}{3}\ddot{\boldsymbol{\omega}}_i) \times \boldsymbol{\omega}_i).$
Update orientation as $\Omega_{i+1} := R_{h\bar{\boldsymbol{\omega}}}\,\Omega_i.$
```

Our experiments confirm that this method is third-order correct.

The difference between the computational complexity of the augmented second-order method and that of the third-order method is one instance of multiplying a matrix and a vector and one instance of vector cross-product.

The algorithm above includes a fourth-order correction term, $\frac{h^3}{36}\ddot{\boldsymbol{\omega}}_i \times \boldsymbol{\omega}_i$. This is included only because $(\dot{\boldsymbol{\omega}}_i + \frac{h}{3}\ddot{\boldsymbol{\omega}}_i)$ is an estimate for the average rate of change of $\boldsymbol{\omega}$ over the time step. There is no other theoretical justification for this term, since there are other fourth-order terms that could equally well be included. However, in our experiments we have found that energy conservation is somewhat improved with inclusion of this term—in our experiments this term reduces the energy drift by a factor of approximately one-third.

True fourth-order method. Section 5 derives a fourth-order-accurate method which includes the additional fourth-order correction term $\frac{h^3}{24}\ddot{\boldsymbol{\omega}}_i \times \boldsymbol{\omega}_i$ into the calculation of $\bar{\boldsymbol{\omega}}$.

TRUE FOURTH-ORDER METHOD

```
$\boldsymbol{\omega}_i = I_i^{-1}\boldsymbol{L}_i$.
$\dot{\boldsymbol{\omega}}_i = I_i^{-1}(\dot{\boldsymbol{L}}_i - \boldsymbol{\omega}_i \times \boldsymbol{L}_i)$.
$\ddot{\boldsymbol{\omega}} = \boldsymbol{\omega}_i \times \dot{\boldsymbol{\omega}}_i + I_i^{-1}(\ddot{\boldsymbol{L}}_i - \dot{\boldsymbol{\omega}}_i \times \boldsymbol{L}_i - 2\boldsymbol{\omega}_i \times \dot{\boldsymbol{L}}_i + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{L}_i))$.
$\dddot{\boldsymbol{\omega}}_i = 2\boldsymbol{\omega}_i \times \ddot{\boldsymbol{\omega}}_i - \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \dot{\boldsymbol{\omega}}_i) + I^{-1}[\dddot{\boldsymbol{L}} - 3\boldsymbol{\omega}_i \times \ddot{\boldsymbol{L}}_i - 3\dot{\boldsymbol{\omega}}_i \times \dot{\boldsymbol{L}}_i - \ddot{\boldsymbol{\omega}}_i \times \boldsymbol{L}_i$
        $+ \dot{\boldsymbol{\omega}}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{L}_i) + 2\boldsymbol{\omega}_i \times (\dot{\boldsymbol{\omega}}_i \times \boldsymbol{L}_i) + 3\boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \dot{\boldsymbol{L}}_i)$
        $- \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{L}_i))]$.
Let $\bar{\boldsymbol{\omega}} := \boldsymbol{\omega}_i + \frac{h}{2}\dot{\boldsymbol{\omega}}_i + \frac{h^2}{6}\ddot{\boldsymbol{\omega}}_i + \frac{h^2}{12}\dot{\boldsymbol{\omega}}_i \times \boldsymbol{\omega}_i + \frac{h^3}{24}\dddot{\boldsymbol{\omega}}_i + \frac{h^3}{24}\ddot{\boldsymbol{\omega}}_i \times \boldsymbol{\omega}_i$.
Update orientation as $\Omega_{i+1} := R_{h\bar{\boldsymbol{\omega}}}\,\Omega_i$.
```

Simo and Wong's explicit algorithm. As discussed above, Simo and Wong [33] gave several algorithms for simulation of rigid rotations. Their energy-preserving algorithm is unfortunately implicit and requires an iterative use of Newton's method in each time step; in addition, it is much more difficult to implement than the algorithms above. Simo and Wong also gave a couple of explicit algorithms which are much faster and easier to implement. Their best explicit algorithm, ALGO_C1 with $\gamma = 1$ and $\beta = 1$, preserves angular momentum exactly but does not exactly preserve energy, very similar to our own algorithms.

SIMO–WONG EXPLICIT METHOD ($\gamma = 1$ AND $\beta = 0$)

```
$\boldsymbol{\omega}_i = I_i^{-1}\boldsymbol{L}_i$.
Let $\boldsymbol{\nu}_i = \Omega_i^{-1}\boldsymbol{\omega}$.
Set $\boldsymbol{a}_i = (\boldsymbol{\nu}_i - \boldsymbol{\nu}_{i-1})/h$.
Let $\bar{\boldsymbol{\nu}} := \boldsymbol{\nu}_i + \frac{h}{2}\boldsymbol{a}_i$.
Update orientation as $\Omega_{i+1} := \Omega_i R_{h\bar{\boldsymbol{\nu}}}$.
```

The Simo–Wang algorithm is not self-starting since it requires knowledge of $\boldsymbol{\nu}_{i-1}$: thus some other algorithm must be used in the first time step to start the simulation.

The essential idea of the Simo–Wong explicit algorithm is to use the current and previous angular velocities to estimate the angular acceleration *in body coordinates* and then to use this to forward estimate the angular velocity as $\bar{\boldsymbol{\omega}}$. From the intuition of the Poinsot inertial ellipsoid, it is clear that this will place the rotation vector $\bar{\boldsymbol{\omega}}$ below the invariable plane. This is in contrast to our third-order adjustment $\frac{1}{12}(\dot{\boldsymbol{\omega}} \times \boldsymbol{\omega})$ introduced in the augmented second-order method above, which pushes the rotation vector up above the invariable plane. We therefore expect the Simo–Wong method to be worse than our second-order method above: this is borne out by our experiments, which show that the Simo–Wong method has accuracy almost identical to the accuracy of the simple first-order method.

McLachlan's and Reich's explicit symplectic algorithm. McLachlan [23] and Reich [31] describe a simple explicit symplectic algorithm for simulation of rigid-body rotation, which exactly preserves angular momentum. We present here an equivalent, but somewhat complicated, version of their algorithm based on the use of spatial coordinates instead of body coordinates. The essential idea of this algorithm is that the rotation vector is computed in body coordinates and then is separated into its components along the body's x, y, and z axes. One of these components of the rotation is performed and then the process is repeated (a total of five times).

McLachlan–Reich explicit symplectic algorithm

```
    Update[1, $\frac{1}{2}h$]
    Update[2, $\frac{1}{2}h$]
    Update[3, $h$]
    Update[2, $\frac{1}{2}h$]
    Update[1, $\frac{1}{2}h$]
```

where Update[x, τ] is the algorithm:

```
Update[$x$, $\tau$] ($x$ specifies one of the body's principal axes)
    Compute $\boldsymbol{L}_{\mathrm{bd}} = \Omega^{-1}\boldsymbol{L}$, angular momentum in body coordinates.
    Let $\boldsymbol{\nu} := J^{-1}\boldsymbol{L}_{\mathrm{bd}}$.
    Let $\bar{\boldsymbol{\nu}}$ be the vector which is the component of $\boldsymbol{\nu}$ in the direction of the principal axis $x$ of the body.
    Update orientation as $\Omega := \Omega R_{\tau\bar{\boldsymbol{\nu}}}$.
```

Some explanation is in order here: the above algorithm updates the orientation in five steps, each step being a rotation around one of the body's principal axes. Accordingly, the best implementation of the algorithm uses body coordinates. Thus each of the five rotations involves calculating and applying a rotation in a two-dimensional subspace. Calculating this rotation requires computing $\sin\theta$ and $\cos\theta$ for a rotation through angle θ, and doing a 3×2 by 2×2 matrix multiplication to update the orientation. The body's momentum vector in body coordinates can be updated with a 2×2 matrix multiplication (using the transposed matrix). Since J is diagonal it is easy to compute the body rotation vector from the body momentum vector.

Therefore, the computational cost of the McLachlan–Reich update step is dominated by the time required to compute 10 trigonometric functions: i.e., five pairs of sines and cosines. A commonly used way to reduce the computational cost is to use Cayley transforms, which use $\sin x \approx t/(1 + t^2/4)$ and $\cos x \approx (1 - t^2/4)/(1 + t^2/4)$—this has the advantage of preserving the matrix orthogonality as well as the symplectic property of the algorithm. Note that the Cayley transforms give third-order-accurate approximations to $\sin x$ and $\cos x$.

The algorithm chooses an arbitrary ordering of the axes to determine the order in which the rotations are applied. (We used the 1–2–3 ordering above.) Our experiments (not all reported below) found that the ordering of the axes made a large difference in the performance of the algorithm. Experimental results are reported below for both the overall best-performing ordering (2–3–1) and the overall-worst performing ordering (1–2–3) of the axes.

One of the common benefits claimed for symplectic algorithms is long-term stability, e.g., as measured by fluctuating energy which never leaves a bounded region (see, e.g., [3]). This claim was amply borne out by our experiments.

Yoshida [35] gives a general method of transforming a n-order explicit symplectic method into an $(n + 2)$-order explicit symplectic method. For $n = 2$, this coincides with earlier constructions of Neri and Forest–Ruth [9] and allows the above second-order method to be transformed into a fourth-order method, albeit at a substantial increase in computational cost.

The basic algorithm can be described as follows: let $x_0 = 1/(2 - 2^{1/3})$ and let $x_1 = -2^{1/3}/(2 - 2^{1/3})$ (note $x_1 < 0$ and $2x_0 + x_1 = 1$). For a fourth-order-accurate update for time step h, first run the second-order symplectic method for a time step of $x_0 h$, then again for a time step of $x_1 h$, and then once again for a time step of $x_0 h$. This can be streamlined a little into an update procedure that contains 13 updates around the principal body axes:

McLachlan–Reich fourth-order explicit symplectic algorithm

```
    Update[1, $\frac{1}{2}x_0h$]
    Update[2, $\frac{1}{2}x_0h$]
    Update[3, $x_0h$]
    Update[2, $\frac{1}{2}x_0h$]
    Update[1, $\frac{1}{2}(x_0+x_1)h$]
    Update[2, $\frac{1}{2}x_1h$]
    Update[3, $x_1h$]
    Update[2, $\frac{1}{2}x_1h$]
    Update[1, $\frac{1}{2}(x_0+x_1)h$]
    Update[2, $\frac{1}{2}x_0h$]
    Update[3, $x_0h$]
    Update[2, $\frac{1}{2}x_0h$]
    Update[1, $\frac{1}{2}x_0h$]
```

Runge–Kutta and Adams–Bashforth–Moulton methods. Our experiments compare the above methods also with the standard fourth-order Runge–Kutta method and the Adams–Bashforth predictor/Adams–Moulton corrector (AB–AM) method. These two methods are very widely used and for Euclidean-valued functions are well known to be fourth-order accurate.

To implement Runge–Kutta and predictor–corrector methods for the sphere, one must be able to take weighted averages of rotations. To do this, we expressed the rotations as quaternions and then averaged them in Euclidean 4-space.[5] Our experiments show that these two methods are only second-order accurate in simulating rigid rotations: the reason for this is that the sphere is not a linear space.

Recently, a number of authors, including Munthe-Kaas [26, 27], Crouch and Grossman [5], and Marthinsen and Owren [21, 29], have given more sophisticated Runge–Kutta and predictor–corrector algorithms which are applicable to differential equations on Lie manifolds. We have not tried implementing these algorithms for rigid-body rotations. These Lie manifold Runge–Kutta and predictor–corrector algorithms are multistep, similar to the standard (Euclidean) Runge–Kutta algorithms. The standard fourth-order Runge–Kutta algorithms require four rotations per update step, and therefore they are somewhat slower than our new algorithms, which need to perform only one rotation per step (see the discussion about runtimes below). We would expect similar computational costs for the fourth-order Lie manifold Runge–Kutta algorithms. However, an advantage of the Lie manifold Runge–Kutta methods is that they could presumably be implemented so as to accommodate the situation where momentum is varying, say in response to external forces,

[5] We experimented also with using the spherical weighted averages introduced by Buss and Fillmore [2], but this gave only a very small improvement relative to the additional computation cost and we abandoned this approach.

without needing to know $\dot{\boldsymbol{L}}$, $\ddot{\boldsymbol{L}}$, and $\dddot{\boldsymbol{L}}$ explicitly. In comparison, our own algorithms, based on Taylor-series expansions, can accommodate changes in momentum but only if $\dot{\boldsymbol{L}}$, $\ddot{\boldsymbol{L}}$, and $\dddot{\boldsymbol{L}}$ are known or can be calculated.

Comparative runtimes. We now compare the runtimes of some of the above algorithms. The first-, second-, third-, and fourth-order algorithms are most efficiently implemented by converting the momentum vector into body coordinates, since this means that the inertia matrix can be kept as a diagonal matrix. Once they compute the average rotation vector $\bar{\boldsymbol{\omega}}$, the rotational update of the orientation matrix may be performed in body coordinates. The straightforward implementation of the rotation requires 2 trigonometric functions, 1 square root, 1 division, and 21 multiplications. In addition to this rotation, the first-order method uses 15 multiplications, the second-order uses 27 multiplications, the augmented second-order uses 36 multiplications, the third-order uses 61 multiplications, and the fourth-order uses 99 multiplications. All these counts assume that the derivatives of the momentum are zero; otherwise, additional cross-product calculations would be needed, increasing the numbers of multiplies. (Since the symplectic algorithms we are comparing with make no provision for the rate of change of the momentum, it seems only fair to assume the momentum is fixed.)

The symplectic algorithms do rotations only around principal axes, which reduces the computational cost of the rotations, but they do many more rotations per time step. A single step of the McLachlan–Reich second-order method uses 85 multiplications and 10 trigonometric function evaluations. A single step of the symplectic fourth-order method uses 221 multiplications and 26 trigonometric function evaluations.

The energy-preservation procedure discussed in the next section can be optimized to use 32 multiplications, 1 division, 3 square roots, and 1 rotation. We include its relative computational cost in Table I too.

We experimentally determined (on a Pentium II) that a sine or cosine evaluation costs about the same as 8 multiplications, a square root costs the same as 5 multiplications, and a division costs the same as 3 multiplications. This allows us to calculate the relative computational costs of a single step of each algorithm. The values in Table I are scaled so that the third-order method with energy preservation has unit computational cost per step. The other computational costs are scaled accordingly (smaller numbers for faster algorithms). Of course, one should treat the relative computational costs as only approximations since, in practice, minor differences in software and hardware configurations can cause significant changes in runtimes.

TABLE I
Relative Computational Costs of Various Algorithms

Base algorithm	Without energy preservation	With energy preservation
1st order	0.30	0.77
2nd order	0.37	0.83
Augmented 2nd order	0.41	0.87
3rd order	0.54	1.00
4th order	0.73	1.19
Symplectic 2nd order	0.84	1.20
Symplectic 4th order	2.19	2.65

If the symplectic algorithms are implemented with Cayley transforms to avoid the use of trigonometric functions, their runtimes improve to be about two-thirds of the runtimes reported above. This makes sense especially for the second-order algorithm since it is still second-order accurate with the use of Cayley transforms. For instance, the symplectic second-order method with Cayley transforms would have relative computational cost values of 0.53 and 0.98 (with and without the energy preservation, respectively).

Because they are multistep, the standard Euclidean Runge–Kutta algorithms have runtimes that are somewhat worse than even our fourth-order method. The Runge–Kutta's runtime is dominated by the time needed to perform four rotations, so its relative cost would be about 1.20 without energy preservation and about 1.67 with energy preservation. These relative cost levels do not include the cost of averaging four rotation values for the final rotation substep. The fourth-order Lie manifold Runge–Kutta algorithm of [26, 28] needs to use five rotations. Both the Euclidean Runge–Kutta and the Lie manifold Runge–Kutta algorithms can be improved as all but the last rotation update can be applied to a single vector—further speed improvements for rotation updates can be obtained using the Rodrigues formula or quaternions. Thus, the computation cost of the Runge–Kutta algorithms is only slightly worse than that of our fourth-order algorithm and is significantly better than that of the fourth-order symplectic algorithm.

4. PRESERVATION OF ENERGY

In this section we introduce a simple and computationally quick method of preserving energy. The scenario is as follows: we presume that we know the energy $E = E_i$ of the rotating object, at time t_i, which can be computed from the orientation Ω_i and the momentum by Eq. (10). To preserve energy, we wish the body to have the same energy at the next time step t_{i+1} (perhaps updated according to Eq. (11) if external torques are applied). Then one of the above methods is used to compute an orientation Ω_{i+1} at time t_{i+1}. Of course, none of the above methods preserve energy, so in general, the energy at time t_{i+1}, as given by Eq. (10), will be slightly different from the desired energy E. Our goal is to slightly perturb the orientation Ω_{i+1} so as to reorient the body to have energy exactly equal to E. To avoid confusion, we will temporarily call this perturbed orientation Ω'_{i+1}, but in the end we set $\Omega_{i+1} := \Omega'_{i+1}$.

Recall from the earlier section that the point $\boldsymbol{\rho}$ lies on the Poinsot inertial ellipsoid and the outward normal of the ellipsoid at $\boldsymbol{\rho}$ is parallel to the momentum vector (see the discussion around Eq. (14)). Energy is correctly preserved if and only if the point $\boldsymbol{\rho}$ remains on the correct polhode curve. The idea behind the energy-preservation perturbation is as follows: first compute the point $\boldsymbol{\rho}_{i+1}$ on the inertial ellipsoid from the momentum $\boldsymbol{L} = \boldsymbol{L}_{i+1}$ and the orientation Ω_{i+1}. Then find a point $\boldsymbol{\rho}'_{i+1}$ which lies on the correct polhode and is close to $\boldsymbol{\rho}_{i+1}$. Finally, reorient the rigid body so that the surface normal vector of the Poinsot inertial ellipsoid at the point $\boldsymbol{\rho}'_{i+1}$ is parallel to the momentum vector.

The only nonstraightforward part is how to choose the point $\boldsymbol{\rho}'_{i+1}$. For this, recall that the polhode is the intersection of the two ellipsoids

$$J_{11}\rho_1^2 + J_{22}\rho_2^2 + J_{33}\rho_3^2 = 1$$

and

$$J_{11}^2\rho_1^2 + J_{22}^2\rho_2^2 + J_{33}^2\rho_3^2 = \frac{\|\boldsymbol{L}\|^2}{2E}.$$

The level surfaces of the left-hand sides of these equations have normals in the directions of $\langle J_{11}\rho_1, J_{22}\rho_2, J_{33}\rho_3\rangle$ and $\langle J_{11}^2\rho_1, J_{22}^2\rho_2, J_{33}^2\rho_3\rangle$. Taking the cross-product and multiplying by $\rho_1\rho_2\rho_3$, we obtain the vector

$$\left\langle J_{22}J_{33}(J_{33}-J_{22})\rho_2^2\rho_3^2\rho_1,\ J_{11}J_{33}(J_{11}-J_{33})\rho_1^2\rho_3^2\rho_2,\ J_{22}J_{11}(J_{22}-J_{11})\rho_1^2\rho_2^2\rho_3\right\rangle,$$

which is orthogonal to the two ellipsoid normal vectors. Holding $\boldsymbol{\rho}_{i+1} = \langle \rho_{0,1}, \rho_{0,2}, \rho_{0,3}\rangle$ constant, we let $\alpha_1, \alpha_2, \alpha_3$ be the three values

$$\alpha_1 = J_{22}J_{33}(J_{33}-J_{22})\rho_{0,2}^2\rho_{0,3}^2, \quad \alpha_2 = J_{11}J_{33}(J_{11}-J_{33})\rho_{0,1}^2\rho_{0,3}^2,$$
$$\alpha_3 = J_{22}J_{11}(J_{22}-J_{11})\rho_{0,1}^2\rho_{0,2}^2.$$

and define

$$h(\rho_1, \rho_2, \rho_3) = \alpha_1\rho_1^2 + \alpha_2\rho_2^2 + \alpha_3\rho_3^2.$$

The level surfaces of h are hyperboloids and the point $\boldsymbol{\rho}_{i+1}$ lies on the surface defined by

$$\{\langle \rho_1, \rho_2, \rho_3\rangle : h(\rho_1, \rho_2, \rho_3) = h(\rho_{0,1}, \rho_{0,2}, \rho_{0,3})\}.$$

From the definition of h, the level surface hyperboloids intersect the polhode perpendicularly.

Now we can define the point $\boldsymbol{\rho}'_{i+1}$ by solving three simultaneous linear equations

$$\begin{aligned} J_{11}\rho_1^2 + J_{22}\rho_2^2 + J_{33}\rho_3^2 &= 1,\\ J_{11}^2\rho_1^2 + J_{22}^2\rho_2^2 + J_{33}^2\rho_3^2 &= \|L\|^2/(2E),\\ \alpha_1\rho_1^2 + \alpha_2\rho_2^2 + \alpha_3\rho_3^2 &= h(\rho_{0,1}, \rho_{0,2}, \rho_{0,3}), \end{aligned}$$

for the values ρ_i^2. Then set $\boldsymbol{\rho}'_{i+1}$ equal to $\langle \pm\sqrt{\rho_1^2}, \pm\sqrt{\rho_2^2}, \pm\sqrt{\rho_3^2}\rangle$, where the signs of the square roots are chosen to agree with the signs of $\rho_{0,1}, \rho_{0,2}, \rho_{0,3}$.

The calculation of $\boldsymbol{\rho}'_{i+1}$ has the effect of moving the point $\boldsymbol{\rho}_{i+1}$ along the surface of the hyperboloid at more-or-less right angles to the polhode—therefore little error is introduced by this process. The most computationally difficult part of computing $\boldsymbol{\rho}'_{i+1}$ is the calculation of the three square roots. The three simultaneous linear equations are easily solved, especially since the first two equations are fixed and thus pivoting and Gaussian elimination can be performed with them ahead of time.

The overall algorithm for conservation of energy is as follows:

```
Input: $\Omega_{i+1}$, Momentum $\boldsymbol{L} = \boldsymbol{L}_{i+1}$, Energy $E$.
    Compute $\boldsymbol{\omega}_{i+1} = I_{i+1}^{-1}\boldsymbol{L}_{i+1}$ and $\boldsymbol{\nu} = \Omega_{i+1}^{-1}\boldsymbol{\omega}_{i+1}$.
    Compute $\boldsymbol{\rho}_{i+1} = \frac{\boldsymbol{\nu}}{\|\boldsymbol{\nu}\|\sqrt{\mathcal{I}}}$ where $\mathcal{I} = \boldsymbol{\nu}\cdot J\boldsymbol{\nu}$.
    Solve for $\boldsymbol{\rho}' = \boldsymbol{\rho}'_{i+1}$ as above.
    Let $\boldsymbol{\tau} = \Omega_{i+1}\langle J_{11}\rho'_1, J_{22}\rho'_2, J_{33}\rho'_3\rangle$, a vector normal to the inertial ellipsoid at the
point $\boldsymbol{\rho}'$ (in spatial coordinates).
    Let $\boldsymbol{\omega}$ be the rotation vector in the direction of $\boldsymbol{\tau}\times\boldsymbol{L}$ with magnitude equal to the
angle $\theta$ between the two vectors: $\boldsymbol{\omega} = \theta\boldsymbol{\tau}\times\boldsymbol{L}/\|\boldsymbol{\tau}\times\boldsymbol{L}\|$.
    Let $\Omega'_{i+1} := R_{\boldsymbol{\omega}}\Omega_{i+1}$.
    Set $\Omega_{i+1} := \Omega'_{i+1}$.
```

In the next to last step, the rotation vector $\boldsymbol{\omega}$ has been chosen to make $R_{\boldsymbol{\omega}}$ be the rotation which aligns $\boldsymbol{\tau}$ to be parallel to $\boldsymbol{L}$.

It should be noted that there is no guarantee that the linear equations for $\boldsymbol{\rho}'$ will be solvable. For the free rigid body this can happen when the rotation vector is fixed, so there is no "wobble" or energy loss anyway. In addition, it can happen when very large rotations occur in a single step: in practice, we have never seen this condition arise, but it could be dealt with by letting the perturbation consist of a tilt toward or away from the principal axis which has either the highest or lowest moment of inertia.

5. HIGHER ORDER METHODS OVER LIE ALGEBRAS

We shall now give a second derivation of the nonlinear adjustment term $\frac{h^2}{12}(\dot{\boldsymbol{\omega}}_i \times \boldsymbol{\omega}_i)$ which was included in the calculation of the $\bar{\boldsymbol{\omega}}$ in the augmented second-order and in the true third-order methods. This second derivation will be based on the (inverse) exponential function on Lie algebras, and therefore it applies in the general setting of Lie algebras where $\boldsymbol{\omega}(t)$ can be replaced by any time-dependent member of the Lie algebra. Simultaneously, we derive the nonlinear adjustment term $\frac{h^3}{24}\ddot{\boldsymbol{\omega}}_i \times \boldsymbol{\omega}_i$ for the fourth-order method and show that there is no nonlinear adjustment needed with a term involving $\boldsymbol{\omega}_i \times (\dot{\boldsymbol{\omega}}_i \times \boldsymbol{\omega}_i)$.[6]

To move to the setting of a Lie algebra, we write $[\boldsymbol{u}, \boldsymbol{v}]$ for $\boldsymbol{u} \times \boldsymbol{v}$. We also write $[\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}]$ for $[\boldsymbol{u}, [\boldsymbol{v}, \boldsymbol{w}]]$, $[\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{x}]$ for $[\boldsymbol{u}, [\boldsymbol{v}, [\boldsymbol{w}, \boldsymbol{x}]]]$, etc. Adjoints $ad(\boldsymbol{u})$ are defined as usual, with $ad(\boldsymbol{u})(\boldsymbol{v}) = [\boldsymbol{u}, \boldsymbol{v}]$. We write $(ad(\boldsymbol{u}))^n$ for the n-fold iteration of $ad(\boldsymbol{u})$. Thus if $f(x)$ is a polynomial or a power series, $f(ad(\boldsymbol{u}))$ denotes an operator on the Lie algebra. We shall work with *right-invariant* Lie algebras, as they are more elegant for our setting. We follow roughly the methods and notation of [28].

The reader who is unfamiliar with Lie algebras can translate the above notation into the setting of rigid-body rotations: the elements X, Y, Z of the Lie algebra are merely rotation vectors. The associated Lie group is the group $SO(3)$ of orientations of a rigid body. The notation $\exp(Z)$ denotes the operation R_Z of rotating the rigid body according to the rotation vector Z.

Suppose $W(t)$ is a time-varying Lie algebra element. (In the setting of rigid-body rotations W is $\boldsymbol{\omega}(t)$.) Let W_0 denote $W(0)$, $\dot{W}_0$ denote $\dot{W}(0)$, $\ddot{W}_0$ denote $\ddot{W}(0)$, etc. Then we can approximate $W(t)$ with its Taylor series

$$W(t) = W_0 + t\dot{W}_0 + \frac{t^2}{2}\ddot{W}_0 + \frac{t^3}{3!}\dddot{W}_0 + O(t^4).$$

Let $h > 0$. We wish to find a fixed Lie algebra element Z such that $\exp(h \cdot Z)$ is equivalent to the result of applying the Lie algebra element $W(t)$ over the time interval $t = 0$ to $t = h$. This Z corresponds to the $\bar{\boldsymbol{\omega}}$ of our first- through fourth-order algorithms. We can express $\exp hZ$ as a "product integral," namely the limit at $N \to \infty$ of

$$\prod_{i=N-1}^{0} \exp((h/N)W(ih/N)).$$

[6] Our original proof was based on the Baker–Campbell–Hausdorf formula. The referees suggested the more direct proof, based on the (inverse) exponential function, which is presented below.

We write $y(h)$ for the limit of this product. The value of Z is of course a function of the time, $Z = Z(h)$. We write $Y(h) = h \cdot Z(h)$ and then we have that $\exp(Y(h)) = y(h)$. Taking first derivatives gives

$$(d\exp)_{Y(t)}(Y'(t)) = y'(t), \tag{17}$$

or, using the inverse exponential function, we write

$$Y'(t) = (d\exp)^{-1}_{Y(t)}(y'(t)). \tag{18}$$

From the definition of $y(t)$, we have the following power series for $y'(t) = W(t)$:

$$y'(t) = W_0 + t\dot{W}_0 + \frac{1}{2}t^2\ddot{W}_0 + \frac{1}{3!}t^3\dddot{W}_0 + \cdots. \tag{19}$$

Since $Y(0) = 0$, we write $Y(t)$ as a power series

$$Y(t) = tY_0 + \frac{1}{2}t^2Y_1 + \frac{1}{3!}t^3Y_2 + \frac{1}{4!}t^4Y_3 + \cdots, \tag{20}$$

and its derivative is $Y'(t) = Y_0 + Y_1t + \frac{1}{2}Y_2t^2 + \frac{1}{3!}Y_3t^3 + \cdots$. In addition, we have the power series for $(d\exp)_{Y(t)}$ and $(d\exp)^{-1}_{Y(t)}$ (c.f. [28]),

$$(d\exp)_{Y(t)} = 1 + \frac{1}{2}ad(Y(t)) + \frac{1}{3!}(ad(Y(t)))^2 + \frac{1}{4!}(ad(Y(t)))^3 + \cdots \tag{21}$$

and

$$(d\exp)^{-1}_{Y(t)} = 1 - \frac{1}{2}ad(Y(t)) + \frac{1}{12}(ad(Y(t)))^2 - \frac{1}{720}(ad(Y(t)))^4 + \cdots, \tag{22}$$

where the general term in the final power series is $(B_q/q!)(ad(Y(t)))^n$ for $n = 0, 1, 2, 4, 6, 8, \ldots$ where B_q are the Bernoulli numbers.

We can now calculate $Y_0, Y_1, Y_2, \ldots$ by substituting the above power series into either Eq. (17) or Eq. (18), then expanding both sides of the equation as a power series, and equating coefficients of common powers of t. We illustrate this procedure for the lower order terms using Eq. (17).

First, consider the constant terms on both sides of (17). On the right-hand side it is just W_0 of course. On the left-hand side, it is immediately seen to be just Y_0. Therefore $Y_0 = W_0$.

Second, consider the coefficient of t on both sides of Eq. (17). Again on the right-hand side it is just $\dot{W}_0$. On the left-hand side, a quick calculation shows it to be $Y_1 - \frac{1}{2}[Y_0, X_0]$. Since $Y_0 = X_0$, this left-hand side equals just Y_1. Equating the two sides gives $Y_1 = X_1$.

Third, consider the coefficient of t^2. On the left-hand side, it equals

$$\frac{1}{2}Y_2 + \frac{1}{2}\left([Y_0, Y_1] + \frac{1}{4}[Y_1, Y_0]\right) + \frac{1}{6}[Y_0, Y_0, Y_0] = \frac{1}{2}Y_2 - \frac{1}{4}[Y_1, Y_0].$$

Setting this equal to the coefficient $\frac{1}{2}\ddot{W}_0$ on the right-hand side gives $Y_2 = \ddot{W}_0 + \frac{1}{2}[\dot{W}_0, W_0]$.

From the above we get that a second-order-accurate formula for Z is $Z(h) = Y(h)/h = Y_0 + \frac{1}{2}Y_1h + \frac{1}{3!}Y_2h^2 + O(h^3)$, which equals

$$Z = W_0 + \frac{h}{2}\dot{W}_0 + \frac{h^2}{6}\ddot{W}_0 + \frac{h^2}{12}[\dot{W}_0, W_0] + O(h^3).$$

This has the same extra term as was introduced in Section 3 in the augmented second-order and the true third-order algorithms, proving the correctness of the third-order algorithm.

Continuing the above style of computations by hand yields $Y_3 = \dddot{W}_0 + [\ddot{W}_0, W_0]$. For higher order terms the number of terms grows exponentially and it is more convenient to use computer algebra systems. We coded some custom rewrite rules in Mathematica to compute these terms; alternatively the symbolic computation Matlab package *Diffman* [7] can be used. The next few results are

$$\begin{aligned} Y_4 = {} & \ddddot{W}_0 + \frac{3}{2}[\dddot{W}_0, W_0] + [\ddot{W}_0, \dot{W}_0] + \frac{1}{2}[\dot{W}_0, \dot{W}_0, W_0] - \frac{1}{6}[W_0, \ddot{W}_0, W_0] \\ & - \frac{1}{6}[W_0, W_0, \dot{W}_0, W_0], \end{aligned}$$

$$\begin{aligned} Y_5 = {} & \overset{\cdot\cdot\cdot\cdot\cdot}{W}_0 + \frac{5}{2}[\dddot{W}_0, \dot{W}_0] + 2[\ddddot{W}_0, W_0] + 2[\ddot{W}_0, \dot{W}_0, W_0] + \frac{1}{2}[\dot{W}_0, \ddot{W}_0, W_0] \\ & - \frac{1}{2}[W_0, \dddot{W}_0, W_0] - \frac{1}{2}[W_0, W_0, \ddot{W}_0, W_0] - [W_0, \dot{W}_0, \dot{W}_0, W_0]. \end{aligned}$$

This immediately gives higher order formulas for Z; namely, for sixth-order accuracy, we can use as the approximation for Z:

$$\begin{aligned} Z = {} & W_0 + \frac{h}{2}\dot{W}_0 + \frac{h^2}{6}\ddot{W}_0 + \frac{1}{12}h^2[\dot{W}_0, W_0] + \frac{h^3}{24}\dddot{W}_0 \\ & + \frac{h^3}{24}[\ddot{W}_0, W_0] + \frac{h^4}{120}\ddddot{W}_0 + \frac{h^4}{80}[\dddot{W}_0, W_0] + \frac{h^4}{120}[\ddot{W}_0, \dot{W}_0] \\ & + \frac{h^4}{240}[\dot{W}_0, \dot{W}_0, W_0] - \frac{h^4}{720}[W_0, \ddot{W}_0, W_0] - \frac{h^4}{720}[W_0, W_0, \dot{W}_0, W_0] \\ & + \frac{h^5}{720}\overset{\cdot\cdot\cdot\cdot\cdot}{W}_0 + \frac{h^5}{288}[\dddot{W}_0, \dot{W}_0] + \frac{h^5}{360}[\ddddot{W}_0, W_0] + \frac{h^5}{360}[\ddot{W}_0, \dot{W}_0, W_0] \\ & + \frac{h^5}{1440}[\dot{W}_0, \ddot{W}_0, W_0] - \frac{h^5}{1440}[W_0, \dddot{W}_0, W_0] - \frac{h^5}{1440}[W_0, W_0, \ddot{W}_0, W_0] \\ & - \frac{h^5}{720}[W_0, \dot{W}_0, \dot{W}_0, W_0] + O(h^6). \end{aligned}$$

Unfortunately, the number of terms apparently grows exponentially with the degree of the algorithm: so far the only cases where a term was unexpectedly dropped involved the $[W_0, \dot{W}_0, W_0]$ term from the fourth-order-accurate algorithm, and the term $[W_0, W_0, W_0, \dot{W}_0, W_0]$ from the sixth-order algorithm.

To the best of our knowledge, the above-derived higher order methods are novel; however, they are closely related to the Fer [8] and Magnus [20] expansions: in fact, both expansions are based on the same "product integral" as we used above. Recently, the use of the Fer and Magnus expansions for numerically solving differential equations has been investigated by a number of people [16–18, 28, 36]. Reference [28] discusses the use of graded Lie algebras and implementation issues for deriving Lie bracket identities in computational software.

6. EXPERIMENTAL RESULTS

We ran a number of experiments to compare the performance of the various simulation methods described above. All the experiments reported here were performed on a freely moving rigid body with no externally applied forces. The rigid body was a rectangular prism of uniform density with ratio of dimensions 1 : 4 : 18. In the reported experiments the body was started with momentum placed at an angle of 45 degrees to the two principal axes of least angular momenta. The 1 : 4 : 18 length ratio gives the dimensions in the third, the first, and the second axes (in that order).

Our first set of experiments measured the conservation of energy for the algorithms described in Section 3. The results are reported in Tables II and III. In all the experiments, the rigid body was initialized with energy equal to 82.1053 and the closer energy values are to this value, the better the simulation preserved energy. The mean rotation values are the average amount of rotation performed in a single simulation step by a perfect algorithm.

Examination of the figures reveals that as far as energy preservation is concerned, the algorithms fall into four groups: Group-0 has the McLachlan–Reich symplectic algorithms and these are by far the best at energy conservation. (The fourth-order McLachlan–Reich algorithm was also tested and did even better at energy conservation, but the results are not reported in the tables.) The augmented second-order, the third-order, and the fourth-order algorithms we call Group-1 and these performed relatively well. The Group-2 algorithms are the second-order, the false third-order, the Runge–Kutta, and the AB–AM predictor–corrector algorithms: these did passably well, but definitely performed less well than the Group-1 methods. Finally, the Group-3 algorithms, including the first-order and the Simo–Wong explicit algorithms, are quite poor at energy conservation.

In general, all the nonsymplectic algorithms steadily gained energy during the simulations, with the sole exception of the augmented second-order algorithm, which usually steadily lost energy. Since momentum is conserved, steadily gaining energy meant that the rigid body came to rotate more-and-more smoothly, with less and less wobble. For the fixed

TABLE II
Energy after *N* Steps: Small Steps

Algorithm	$N = 10^5$ $h = 0.0001$ $\theta = 0.41^\circ$	$N = 8 \times 10^5$ $h = 0.0005$ $\theta = 2.0^\circ$	$N = 4 \times 10^5$ $h = 0.001$ $\theta = 4.1^\circ$	$N = 2 \times 10^5$ $h = 0.002$ $\theta = 8.1^\circ$	$N = 10^5$ $h = 0.004$ $\theta = 16.3^\circ$
1st order	109.663	150.0	150.0	150.0	150.0
2nd order	82.1054	82.5637	85.8535	115.058	150.0
False 3rd order	82.1054	82.6471	86.5584	121.164	150.0
Augmented 2nd order	82.1053	82.0749	81.8636	80.2222	69.6595
3rd order	82.1053	82.1154	82.1866	82.7686	88.0158
4th order	82.1053	82.1053	82.1056	82.1177	82.4979
Simo–Wong explicit	109.661	150.0	150.0	150.0	150.0
McLachlan–Reich 1–2–3	82.1051	82.1028	82.1040	82.0482	81.8740
McLachlan–Reich 2–3–1	82.1053	82.1053	82.1053	82.1054	82.1057
Runge–Kutta	82.1053	82.3810	84.3364	101.328	149.811
AB–AM	82.1055	82.5159	85.4372	111.305	149.999

Note. Initial energy was 82.1053. The highest possible energy is 150.0 and represents complete failure of energy preservation. N = number of steps; h = time interval; θ = mean rotation.

TABLE III
Energy after *N* Steps: Large Steps

Algorithm	$N = 1000$ $h = 0.004$ $\theta = 16.3^\circ$	$N = 500$ $h = 0.008$ $\theta = 32.5^\circ$	$N = 250$ $h = 0.016$ $\theta = 65.1^\circ$	$N = 200$ $h = 0.020$ $\theta = 81.4^\circ$
Augmented 2nd order	81.9528	81.0002	76.3078	74.5092
3rd order	82.1603	82.6283	89.2196	101.390
4th order	82.1092	82.2288	86.0610	94.9191
McLachlan–Reich 1–2–3	81.9601	82.0779	79.3202	77.6101
McLachlan–Reich 2–3–1	82.1057	82.1069	82.1083	82.1063
Runge–Kutta	83.4575	91.8125	132.808	149.855
AB–AM	84.4914	101.429	149.736	149.999

Note. Initial energy was 82.1053. The highest possible energy is 150.0 and represents complete failure of energy preservation. $N =$ number of steps; $h =$ time interval; $\theta =$ mean rotation.

momentum, the maximum value for the energy is 150.0, and values near 150.0 represent a complete failure of energy preservation.

Table II shows the results of simulations with relatively small simulation steps, with mean rotation per step between 0.41 degrees and 16.3 degrees. The first column shows the results of simulating 10 seconds of rotation, whereas the other columns all show the results of 400 seconds of simulation. (The reason for the shorter simulation time in the first column is that otherwise it was essentially impossible to get any of the poorest algorithms to have any meaningful preservation of energy at all.) Here we see that the Group-3 algorithms exhibit substantial degradation in energy preservation, while the other algorithms do quite well. The Group-0 and Group-1 algorithms performed dramatically better than the other algorithms. The Group-1 algorithms performed consistently better than the Group-2 algorithms in all these experiments—this is shown particularly dramatically in the last two columns of Table II where the mean rotations are between 8.2 degrees and 16.3 degrees per simulation step.

These results are meaningful also because the Group-1 algorithms and, to a lesser extent, the Group-0 algorithms are computationally much faster and easier to code than the Group-2 algorithms.

Table III shows the results of similar experiments for large simulation steps for some of the Group-0, Group-1, and Group-2 algorithms. For these experiments, 4 seconds of motion of the same rigid body was simulated, with mean rotations ranging from 16.3 degrees to 81.4 degrees per time step. Once again the Group-0 methods were outstanding at energy conservation. Also, as expected, the Group-1 algorithms were consistently and substantially more accurate in terms of energy conservation than the Group-2 algorithms.

Table III was designed to show the ranges in which the augmented second-order, the third-order, and the fourth-order algorithms give reasonable energy conservation. However, the McLachlan–Reich method had good energy stability over a much longer period of time. We repeated the experiment from the last column of Table III except with $N = 200{,}000{,}000$. Even with the large step size of 81.4° per step on average, the energy after 200 million simulation steps was equal to 75.2587 (for the 1–2–3 ordering) and 82.1218 (for the 2–3–1), which represents a very modest drift in energy.

Our second set of experiments measured the relative efficiency of the simulation methods. In each experiment, we choose a total time T and a target accuracy ϵ: we then calculated, for each method, what step size h is sufficient for the simulation to yield an answer which is correct to within an error ϵ in orientation. To be precise, we actually calculated the number of steps, N, needed—of course $h = T/N$. The results of our experiments are reported in Tables IV and V. Each algorithm is tested both with and without the use of the energy preservation method of Section 4. The tables report both N and the mean rotation angle, θ, during each step (the values of θ are expressed as degrees). The error value ϵ is measured in terms of radians however; for example, an error of $\epsilon = 10^{-3}$ means that the final error in orientation was less than $1/1000$ of a radian.

In order to analyze the data in Tables IV and V, it is useful to group the simulation methods into three classes, based on their order of accuracy. Group A is the fourth-order-accurate algorithms. Group B contains the third-order-accurate methods, of which the third-order method is the only representative (however, the third-order method with energy preservation showed better than fourth-order accuracy and belongs to Group A). Group C contains the second-order-accurate methods, namely, the second-order method, the false third-order method, the augmented second-order method, the McLachlan–Reich symplectic method, the Runge–Kutta method, and the AB–AM predictor–corrector method. In some cases, the McLachlan–Reich algorithm did sufficiently well to be categorized as a Group B algorithm. Group D contains the first-order-accurate methods, namely, the first-order method and the Simo–Wong explicit method. Generally, the Group A methods outperform the Group B methods, which outperform the Group C methods, which outperform Group D.

The Group D methods performed poorly in comparison to the Group C methods. In fact, without energy preservation, the Group D methods failed to give enough accuracy to even measure their relative performance. With energy preservation, they did much better and, in fact, using the energy preservation caused the Group D algorithms to act like second-order-accurate algorithms. In column 1 of Table IV, we see that the first-order method with energy preservation could keep the overall error rate to 0.001 after 514 steps, with a mean rotation of $\theta = 3.17$ degrees in each step. From column two, we see that the same algorithm needed 15,356 steps, with a mean rotation of 0.11 degrees, to perform the same simulation with an accuracy of 10^{-6}. The Simo–Wong algorithm performed almost identically to the first-order algorithm.

The Group C algorithms all acted like second-order-accurate systems but still encompassed a fairly wide range of performance levels. The worst performance came from the AB–AM method without energy preservation. The AB–AM method with energy preservation, the Runge–Kutta methods, and the second-order methods had very similar performance levels as measured by the number of steps N. Finally, the augmented second-order method was the most efficient method of Group C, requiring only about one-half as many steps as the other Group C algorithms. For example, in the second column of Table IV, we see that the augmented second-order method can achieve in 6513 steps the same accuracy that the Runge–Kutta method can achieve in 12,771 steps. Actually, this grossly understates the performance advantage of the augmented second-order method: the running time to carry out a single Runge–Kutta step is approximately three times as long as the time to compute a single step of the augmented second-order method. Thus, roughly speaking, the augmented second-order method is approximately six times as efficient as the Runge–Kutta method when execution speed is taken into account. Similarly, even the (nonaugmented) second-order method is approximately three times as efficient as the Runge–Kutta method.

The worst-performing McLachlan–Reich implementation, with axis ordering 1–2–3, acted very much like the second-order algorithms of Group C. It also tended to gain in

TABLE IV

Number of Steps (N) and Mean Rotation in Degrees (θ) Required to Achieve a Given Accuracy

Algorithm	$T = 0.4$, $\epsilon = 0.001$	$T = 0.4$, $\epsilon = 10^{-6}$	$T = 4.0$, $\epsilon = 0.001$
1st order	$N > 20{,}480$	$N > 20{,}480$	$N > 20{,}480$
(nonpreserving)	failed	failed	failed
1st order	$N = 514$	$N = 15{,}356$	$N = 16{,}216$
(energy-preserving)	$\theta = 3.17$	$\theta = 0.11$	$\theta = 1.00$
2nd order	$N = 387$	$N = 11{,}509$	$N = 14{,}546$
(nonpreserving)	$\theta = 4.22$	$\theta = 0.14$	$\theta = 1.12$
2nd order	$N = 364$	$N = 11{,}500$	$N = 11{,}505$
(energy-preserving)	$\theta = 4.48$	$\theta = 0.14$	$\theta = 1.41$
False 3rd order	$N = 412$	$N = 12{,}768$	$N = 15{,}056$
(nonpreserving)	$\theta = 3.95$	$\theta = 0.13$	$\theta = 1.08$
False 3rd order	$N = 404$	$N = 12{,}784$	$N = 12{,}773$
(energy-preserving)	$\theta = 4.04$	$\theta = 0.13$	$\theta = 1.27$
Augmented 2nd order	$N = 195$	$N = 6513$	$N = 5573$
(nonpreserving)	$\theta = 210$	$\theta = 0.25$	$\theta = 2.92$
Augmented 2nd order	$N = 210$	$N = 6617$	$N = 6616$
(energy-preserving)	$\theta = 7.78$	$\theta = 0.25$	$\theta = 2.46$
3rd order	$N = 82$	$N = 789$	$N = 3661$
(nonpreserving)	$\theta = 20.09$	$\theta = 2.06$	$\theta = 4.45$
3rd order	$N = 37$	$N = 209$	$N = 652$
(energy-preserving)	$\theta = 45.17$	$\theta = 7.83$	$\theta = 25.00$
4th order	$N = 53$	$N = 236$	$N = 1304$
(nonpreserving)	$\theta = 31.29$	$\theta = 6.93$	$\theta = 12.49$
4th order	$N = 39$	$N = 219$	$N = 688$
(energy-preserving)	$\theta = 42.85$	$\theta = 7.47$	$\theta = 23.69$
Simo–Wong explicit	$N > 20{,}480$	$N > 20{,}480$	$N > 20{,}480$
(nonpreserving)	failed	failed	failed
Simo–Wong explicit	$N = 513$	$N = 15{,}356$	$N = 16{,}209$
(energy-preserving)	$\theta = 3.17$	$\theta = 0.11$	$\theta = 1.00$
McLachlan–Reich 1–2–3	$N = 514$	$N = 16{,}250$	$N = 16{,}131$
(nonpreserving)	$\theta = 3.17$	$\theta = 0.10$	$\theta = 1.01$
McLachlan–Reich 1–2–3	$N = 193$	$N = 6063$	$N = 5210$
(energy-preserving)	$\theta = 8.48$	$\theta = 0.27$	$\theta = 3.12$
McLachlan–Reich 2–3–1	$N = 39$	$N = 1192$	$N = 1191$
(nonpreserving)	$\theta = 42.86$	$\theta = 1.37$	$\theta = 13.68$
McLachlan–Reich 2–3–1	$N = 31$	$N = 955$	$N = 945$
(energy-preserving)	$\theta = 54.19$	$\theta = 1.71$	$\theta = 17.23$
MR 4th-order 2–3–1	$N = 26$	$N = 135$	$N = 431$
(nonpreserving)	$\theta = 64.92$	$\theta = 12.15$	$\theta = 37.85$
MR 4th-order 2–3–1	$N = 18$	$N = 73$	$N = 196$
(energy-preserving)	$\theta = 95.13$	$\theta = 22.60$	$\theta = 83.45$
Runge–Kutta	$N = 405$	$N = 12{,}771$	$N = 13{,}512$
(nonpreserving)	$\theta = 4.03$	$\theta = 0.13$	$\theta = 1.20$
Runge–Kutta	$N = 404$	$N = 12{,}785$	$N = 12{,}773$
(energy-preserving)	$\theta = 4.04$	$\theta = 0.13$	$\theta = 1.27$
AB–AM	$N = 782$	$N > 20{,}480$	$N > 20{,}480$
(nonpreserving)	$\theta = 2.08$	failed	failed
AB–AM	$N = 451$	$N = 14{,}274$	$N = 14{,}279$
(energy-preserving)	$\theta = 3.62$	$\theta = 0.11$	$\theta = 1.14$

TABLE V
The Performance of Algorithms in High-Accuracy Simulations

Algorithm	$T = 4.0$ $\epsilon = 10^{-6}$	$T = 40.0$ $\epsilon = 0.001$	$T = 40.0$ $\epsilon = 10^{-6}$
McLachlan–Reich 2–3–1 (nonpreserving)	$N > 40{,}960$ failed	$N > 37{,}533$ $\theta = 4.34$	$N > 163{,}840$ failed
McLachlan–Reich 2–3–1 (energy-preserving)	$N > 40{,}960$ failed	$N = 29{,}760$ $\theta = 5.47$	$N > 163{,}840$ failed
MR 4th-order 2–3–1 (nonpreserving)	$N = 2364$ $\theta = 6.89$	$N = 7526$ $\theta = 21.63$	$N = 41{,}617$ $\theta = 3.92$
MR 4th-order 2–3–1 (energy-preserving)	$N = 1034$ $\theta = 15.76$	$N = 3282$ $\theta = 49.60$	$N > 163{,}840$ failed
3rd order (nonpreserving)	$N = 36{,}552$ $\theta = 0.45$	$N > 40{,}960$ failed	$N > 163{,}840$ failed
3rd order (energy-preserving)	$N = 3700$ $\theta = 4.40$	$N = 11{,}621$ $\theta = 14.01$	$N = 65{,}631$ $\theta = 2.48$
4th order (nonpreserving)	$N = 5294$ $\theta = 3.07$	$N = 32{,}656$ $\theta = 4.98$	$N = 131{,}473$ $\theta = 1.24$
4th order (energy-preserving)	$N = 3888$ $\theta = 4.19$	$N = 12{,}275$ $\theta = 13.26$	$N = 69{,}254$ $\theta = 2.35$

accuracy when combined with energy preservation. The best-performing McLachlan–Reich implementation, with axis ordering 2–3–1, acted more like a third-order-accurate algorithm than a second-order algorithm. The latter tended to benefit less from the use of energy preservation.

The third-order method when combined with energy preservation was substantially superior to every Group B, C, or D method. For example, in column 3 of Table IV, the third-order method with energy preservation needs only 652 steps to achieve the same accuracy as 5573 steps of the augmented second-order method—a better than seven-fold increase in efficiency. It is impressive to note that this level of accuracy was obtained with a mean rotation of 25 degrees per simulation step. Even better speed improvements are realized with higher levels of accuracy: in column 2, only 209 steps of the third-order algorithm with energy preservation have the same overall accuracy as 6513 steps of the augmented second-order method—a better than 25-fold increase in efficiency!

The closest competitor to the third-order method with energy preservation from Groups B, C, and D is the McLachlan–Reich 2–3–1 method. Here we note that the more accuracy that is required (10^{-6} versus 10^{-3}) and the longer the period of the simulation, then the greater the advantage of the third-order method with energy preservation. Of course, this should be expected when comparing a third-order method with a second-order method, but it should be noted that this advantage is occurring already with fairly large simulation steps (e.g., for the long-term simulations reported in Table V, the simulation performs mean rotations of 4.40 degrees, 14.01 degrees, and 2.48 degrees). From Table I, we see that, modulo implementation issues, the speeds of the two methods are more-or-less comparable; therefore, the third-order method is overall more efficient in terms of accuracy than the McLachlan–Reich second-order methods.

Table V shows more clearly the impressive accuracy of the third-order method with energy preservation. In column 2, we see that 11,621 simulation steps with mean rotation of just over 14 degrees per step still maintain an accuracy of 0.001. And, 65,631 steps of mean rotation 2.48 degrees gives an accuracy of 10^{-6}.

The rest of Table V shows the relative performance of the other best-performing simulation methods. Measuring in terms of number of steps, the symplectic fourth-order MR method with the 2–3–1 ordering (non-energy-preserving) was able to achieve higher accuracy than the third-order method with energy preservation—the symplectic algorithm consistently needed about two-thirds as many steps as the third-order energy-preserving algorithm. However, since the former algorithm has a computational cost of more than twice that of the latter, the third-order algorithm with energy preservation enjoys an advantage in this regard.

The fourth-order method without energy preservation predictably did significantly better than the third-order method without energy preservation. With energy preservation, we observed the surprising result that the third- and fourth-order algorithms have essentially equivalent accuracy. Both experimentally exhibited better than fourth-order accuracy, but we have no theoretical justification for why the third-order algorithm with energy preservation performed as well as the fourth-order algorithm.

One other somewhat odd feature of Table V arises in the comparison of the MR 2–3–1 algorithm with energy preservation to the MR 2–3–1 algorithm without energy preservation. First, since the symplectic algorithms are so good at preserving energy it is surprising how much adding energy preservation helps; furthermore, adding the energy preservation to the symplectic algorithms is somewhat inelegant and presumably destroys the symplectic properties. Second, although about half as many steps were needed by the energy-preserving version, the energy-preserving version failed to converge at all in the most demanding simulation (the last column of the table). This was presumably due to the fact that so many operations are performed per step (13 rotations plus the energy-correction operation), and that roundoff errors accumulated more quickly than in the other algorithms, rendering the algorithm incapable of converging to the target accuracy.

To verify that our higher order algorithms are correct and actually have the expected higher order accuracy, consider a simulation for a total time T, where T is held constant. If N steps are used, then the time step h is proportional to $1/N$; hence a third-order algorithm would have error $O(1/N^3)$ per time step, and so the overall error for the entire simulation is expected to be $O(1/N^2)$. Likewise a fourth-order algorithm should have overall error $O(1/N^3)$. Actually, the overall error should be somewhat lower than this since we expect some cancellation of errors. Our experiments with the third- and fourth-order methods without energy preservation confirmed that these algorithms are indeed third and fourth order (respectively). The third-order method with energy preservation was seen to be substantially better than third order: indeed comparing the last two columns of Table V or comparing the last column of Table IV and the first column of Table V, we see that a six-fold increase in the number of steps yields a 1000-fold increase in overall accuracy! This performance was generally seen in other situations, in that we typically saw that doubling the number of steps caused the overall error to be divided by about 15. Thus the experimental evidence indicates that the third-order method with energy preservation acts like a better-than-fourth-order-accurate algorithm.[7]

[7] A fourth-order algorithm would nominally have the overall error drop by a factor of 8 when the number of steps is doubled, but in fact it would sometimes do better owing to cancellation of errors. The observed factor of 15 means that the algorithm was between fourth-order and fifth-order accurate for our test scenario.

In viewing the data in Table IV, we see that the use of energy preservation improves the simulation accuracy substantially in the case of the Group D algorithms; in fact, our empirical observations suggest that the Group D algorithms with energy preservation act very much like second-order-accurate Group C methods. The AB–AM method, the third-order method, the second-order 1–2–3 McLachlan–Reich method, and the fourth-order 2–3–1 symplectic method also benefited substantially from energy preservation. The remaining methods acheived only a modest improvement with the use of energy preservation, and the augmented second-order method had a small decrease in accuracy when combined with energy preservation.

7. CONCLUSIONS

We have introduced several new algorithms for the simulation of rigid rotations. We derived a third-order term which can be used to improve the energy preservation of a second-order simulation method (in the "augmented second-order algorithm") or which can be used to formulate an third-order method which is correctly third order. We further derived fourth-order correction terms in the general setting of Lie algebras and gave a corresponding fourth-order-accurate algorithm. These algorithms can be readily extended to higher orders and apply to general Lie algebras. Second, we gave a simple and easy-to-calculate method for preserving energy exactly based on reorienting the body slightly so as to preserve both momentum and energy. This reorientation is done at right angles to the direction of movement and thus introduces very little error. First-order methods combined with energy preservation were experimentally observed to act like second-order-accurate methods. The third-order method combined with energy preservation was experimentally observed to act like a better-than-fourth-order-accurate method.

Our experiments were carried out on torque-free bodies, but since the algorithms were designed without any special assumptions regarding torques, our methods should also work well in the presence of torques.

Traditional implementations of algorithms such as Runge–Kutta or the Adams–Bashforth–Moulton predicator–corrector method are not nearly as efficient at simulating rotation as the augmented second-order method or the third-order method or the symplectic methods. The augmented second-order method is approximately six times as efficient as Runge–Kutta when computational speed is accounted for, and the third-order methods are substantially even more efficient for high long-term accuracy.

Let us consider the question of which algorithm is best to use. In general, we recommend the use of one of the following: the augmented second-order, the third-order, the fourth-order, the McLachlan–Reich algorithm, or the fourth-order symplectic algorithm. The first three (especially the third- and fourth-order methods) can be combined with our energy preservation method and also thereby achieve better accuracy as well as stability. The symplectic algorithms can also be combined with our energy preservation method, but this would seemingly destroy the symplectic property and would be a somewhat inelegant choice.

The answer to which algorithm is best will depend largely on the application. First, if one is interested mainly in long-term stability with large time steps, then the McLachlan–Reich second-order method is a strong candidate. This method allow only a small fluctuation in energy and is simple to implement with a reasonable computational cost, especially if Cayley transforms are used. The fourth-order symplectic method has even better energy stability and significantly more accuracy, albeit at a higher computational cost per step. A similar stability can be obtained by using our second-order or third-order methods with the energy-preservation method; these can provide very good accuracy and require somewhat lower computational resources. Second, if one is interested in absolute long-term accuracy, then the third-order method with energy preservation gives the best accuracy per computational cost. Third, if one can estimate the derivatives of the angular momentum, then the second-, third-, and fourth-order algorithms can incorporate these derivatives directly.

As a general observation, if one is simulating the rotation of a single isolated rigid body, then there are a number of ways to do this more directly: one could use elliptic functions for instance, or one could use any of our algorithms to simulate the body through one “wobble” to very high accuracy and then predict the future positions of the rigid body very accurately. Thus, any interesting use of the rigid-body rotation algorithms should work well in the presence of external torques. This is potentially an important advantage for the our second-, third-, and fourth-order algorithms, since if one can estimate the external torque, or more generally, the derivatives of the angular momentum, then this can be directly incorporated into these algorithms.

ACKNOWLEDGMENTS

We thank R. McLachlan and S. Reich for suggestions and for pointing out a error in a preliminary draft of this paper. We also thank Jay Fillmore and Nolan Wallach for discussions on Lie theory. Finally, the referees’s comments on the first draft of the paper helped improve the proofs of our paper considerably.

REFERENCES

1. G. Benettin and A. Giorgilli, On the Hamiltonian interpolation of near-to-the-identity symplectic mappings with applications to symplectic integration algorithms, *J. Stat. Phys.* **74**, 1117 (1994).
2. S. R. Buss and J. Fillmore, Spherical averages and applications to spherical splines and interpolation. Submitted for publication.
3. P. J. Channell and F. R. Neri, An introduction to symplectic integrators, in *Integration Algorithms and Classical Mechanics*, edited by, J. E. Marsden, G. W. Patrick, and W. F. Shadwick, Fields Institute Communications #10 (*Am. Math. Soc.*, Providence, 1996), p. 45.
4. P. J. Channell and J. C. Scovel, Integrators for Lie–Poisson dynamical systems, *Physica D* **50**, 80 (1991).
5. P. E. Crouch and R. Grossman, Numerical integration of ordinary differential equations on manifolds, *J. Nonlinear Sci.* **3**, 1 (1993).
6. A. Dullweber, B. Leimkuhler, and R. I. McLachlan, Symplectic splitting methods for rigid body molecular dynamics, *J. Chem. Phys.* **107**, 5840 (1997).
7. K. Engø, A. Marthinsen, and H. Munthe-Kaas, *DiffMan’s User Guide*, Tech. Rep. 166, University of Bergen, March 1999. http://www.ii.uib.no//diffman.
8. F. Fer, Résolution del l’equation matricielle $\dot{U} = pU$ par produit infini d’exponentielles matricelles, *Bull. Classe Sci. Acad. Roy. Belgium* **44**, 818 (1958).
9. E. Forest and R. D. Ruth, Fourth-order symplectic integration, *Physica D* **43**, 105 (1990).
10. Z. Ge, Equivariant symplectic difference schemes and dynamical systems, *Physica D* **49**, 376 (1991).
11. Z. Ge and J. E. Marsden, Lie–Poisson Hamilton–Jacobi theory and Lie–Poisson integrators, *Phys. Lett. A* **133**, 134 (1988).
12. M. Géradin and D. Rixen, Parameterization of finite rotations in computational dynamics: A review, *Rev. Eur. Éléments Finis* **4**, 497 (1995).
13. H. Goldstein, *Classical Mechanics* (Addison-Wesley, Reading, MA, 1950).

14. E. Hairer and C. Lubich, The life-span of backward error analysis for numerical integrators, *Numer. Math.* **76**, 441 (1997).
15. T. Holder, B. Leimkuhler, and S. Reich, Explicit variable stepsize and time-reversible integration, *BIT*, to appear, 2000.
16. A. Iserles, A. Marthinsen, and S. P. Nørsett, *On the Implementation of the Method of Magnus Series for Linear Differential Equations*, Tech. Rep. Numerics No. 1, The Norwegian University of Science and Technology, Trondheim, Norway, 1998.
17. A. Iserles and S. P. Nørsett, *Linear ODEs in Lie Groups*, Tech. Rep. DAMTP 1997/NA9, Cambridge, 1997. To appear in *Proc. 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics*.
18. A. Iserles and S. P. Nørsett, *On the Solution of Linear Diffential Equations in Lie Groups*, Tech. Rep. DAMTP 1997/NA3, Cambridge, 1997.
19. D. Lewis and J. C. Simo, Conserving algorithms for the *N* dimensional rigid body, in *Integration Algorithms and Classical Mechanics*, edited by, J. E. Marsden, G. W. Patrick, and W. F. Shadwick, Fields Institute Communications #10 (*Am. Math. Soc.*, Providence, 1996), p. 121.
20. W. Magnus, On the exponential solution of diferential equations for a linear operator, *Commun. Pure Appl. Math.* **7**, 649 (1954).
21. A. Marthinsen and B. Owren, *A Note on the Construction of Crouch–Grossman Methods*, Tech. Rep. Numerics No. 2/1998, The Norwegian University of Science and Technology, Trondheim, Norway, 1998.
22. M. F. Massoud and Y. A. Youssef, Digital simulation of the rotation of a moving triad with a known matrix spin vector, *Simulation* **20**, 138 (1973).
23. R. I. McLachlan, Explicit Lie–Poisson integration and the Euler equations, *Phys. Rev. Lett.* **71**, 3043 (1993).
24. R. I. McLachlan, On the numerical integration of ordinary differential equations by symmetric composition methods, *SIAM J. Sci. Comput.* **16**, 151 (1995).
25. R. I. McLachlan and C. Scovel, Equivariant constrained symplectic integration, *J. Nonlinear Sci.* **5**, 233 (1995).
26. H. Munthe-Kaas, Runge–Kutta methods on Lie groups, *BIT* **38**, 92 (1998).
27. H. Munthe-Kaas, High order Runge–Kutta on manifolds, *Appl. Numer. Math.* **29**, 115 (1999).
28. H. Munthe-Kaas and B. Owren, *Computations in a Free Lie Algebra*, Tech. Rep. 148, University of Bergen, 1998.
29. B. Owren and A. Marthinsen, Runge–Kutta methods adapted to manifolds and based on rigid frames, *BIT* **39**, 116 (1999).
30. S. Reich, Momentum conserving symplectic integrators, *Physica D* **17**, 375 (1994).
31. S. Reich, Symplectic integrators for systems of rigid bodies, in *Integration Algorithms and Classical Mechanics*, edited by J. E. Marsden, G. W. Patrick, and W. F. Shadwick, Fields Institute Communications #10 (*Am. Math. Soc.*, Providence, 1996), p. 181.
32. S. Reich, Backward error analysis for numerical integrators, *SIAM J. Numer. Anal.* **36**, 1549 (1999).
33. J. C. Simo and K. K. Wong, Unconditionally stable algorithms for rigid body dynamics that exactly preserve energy and momentum, *Int. J. Numer. Meth. Eng.* **31**, 19 (1991).
34. J. L. Synge and B. A. Griffith, *Principles of Mechanics* (McGraw–Hill, New York, 1959).
35. H. Yoshida, Construction of higher order symplectic integrators, *Phys. Lett. A* **12**, 262 (1990).
36. A. Zanna, *The Fer Expansion and Time Symmetry: A Strang-Type Approach*, Tech. Rep. NA1998/14, DAMTP, Cambridge, 1998.