Proceedings of the 31st Conference
on Decision and Control
Tucson, Arizona • December 1992

On the Numerical Integration of the Dynamic Attitude Equations

P.E. Crouch and Y. Yan
Center for Systems Science and Engineering
Arizona State University
Tempe AZ 85287 - 7606*

R. Grossman
Laboratory for Advanced Computing
University of Illinois at Chicago
Chicago, IL 60680**

Abstract

We describe new types of numerical integration algorithm developed by the authors. The main aim of the algorithms is to numerically integrate differential equations which evolve on geometric objects, such as the rotation group. The algorithms provide iterates which lie on the prescribed geometric object, either exactly, or to some prescribed accuracy, independent of the order of the algorithm. In this sense the algorithms can be called exact integration algorithms. This paper describes the application of these new algorithms to the differential equations describing the evolution of the attitude of a rigid body, such as a spacecraft, when the body is modeled by the corresponding dynamical equations. Such exact integration algorithms can be important in control applications, and indeed their analysis has employed exactly the same tools used in contemporary analysis of nonlinear control systems.

1. Introduction to a New Class of Numerical Integration Algorithms

The authors have recently developed new types of numerical integration algorithm [4] and [5], inspired in part by the classical multistep and single step algorithms, such as the Adams Bashforth, and the Runge Kutta algorithms respectively. The new algorithms depend upon the existence of an oracle which can integrate certain "simple" differential equations to any degree of accuracy, or indeed can give an analytic expression for the solution. The new algorithms then generate numerical solutions of a given differential equation by suitably approximating its solutions by solutions of the "simple" differential equations. One application lies in the case where the "simple" differential equations describe evolution on some geometric object, such as the rotation group, describing the orientation of a rigid body. The solution of a given differential equation on this geometric object is then approximated numerically by iterates generated as solutions of the "simple" differential equations, and which therefore remain on the geometric object to any prescribed accuracy, independent of the order of the numerical integration procedure. The new algorithms have the property that in the case that the "simple" differential equations are just those with constant left hand side, then the new algorithms degenerate to their classical counterpart. The "order" of one of the new algorithms, is based on an analysis reported in earlier work by the authors [5], which is intimately related to contemporary analysis of nonlinear control systems. Moreover the "exact" integration methods described here are important to nonlinear control algorithms which require accurate models of the systems to construct suitable feedback controls.

We first briefly describe the new of class of algorithm introduced by the authors in [4] and [5]. We wish to numerically integrate solutions of differential equations evolving on a N dimensional submanifold M of $\mathbf{R}^n$. Thus as a set of differential equations on M we write:

$$\dot{x}(t) = F(x(t)), \qquad x(0) = p \in M, \tag{1}$$

where

$$F(x) = \sum_{\mu=1}^{N} a^{\mu}(x)\, A_{\mu}(x), \quad x \in M \tag{2}$$

We assume however that the system of equations is in fact presented as a set of differential equation in $\mathbf{R}^n$, so that F and A_μ are vector fields on $\mathbf{R}^n$ and a^μ are functions on $\mathbf{R}^n$. For the system to evolve on M we insist that A_μ are everywhere tangent to M, so that the same is true of F. Denoting the flow of an arbitrary vector field Z on $\mathbf{R}^n$ by $(t, x) \to \text{expt } Z(x)$, exp: $\mathbf{R} \times \mathbf{R}^n \to \mathbf{R}^n$, it follows that if $p \in M$ then expt $Z(p) \in M$ for all t, for which the flow is defined. We make the following critical assumption:

Assumption. There is an oracle which computes the flow $(t, x) \to \text{expt } Z(x)$, either exactly, or to any desired accuracy, for any vector field of the form:

$$Z(x) = \sum_{\mu=1}^{N} \alpha^{\mu} A_{\mu}(x)$$

where α^μ are real numbers.

We also make fundamental use of the vector field F^p:

$$F^{p}(x) = \sum_{\mu=1}^{N} a^{\mu}(p)\, A_{\mu}(x)$$

This vector field is simply the vector field F with coefficients "frozen" at the point p. The assumption above may be interpreted as: F with coefficients frozen at an arbitrary point p can either be explicitly integrated or, integrated to arbitrary accuracy.

Numerical integration algorithms are traditionally described as either single step algorithms, such as the class of Runge Kutta algorithms, or multistep algorithms, such as the Adams Bashforth algorithms. The algorithms are also classified as being explicit or implicit, depending upon whether or not the up date equations give explicit or implicit formulas for the next iterate x^{n+1}. The new algorithms described here are all explicit and may also be classified in this way:

New Single Step Algorithms
Define vector fields inductively by setting:

$$F_1^p(x) = \sum_{\mu=1}^{N} a^{\mu}(p)\, A_{\mu}(x)$$

$$F_2^p(x) = \sum_{\mu=1}^{N} a^{\mu}(\text{exphc}_{21} F_1^p(p))\, A_{\mu}(x)$$

$$F_3^p(x) = \sum_{\mu=1}^{N} a^{\mu}(\text{exphc}_{32} F_2^p(\text{exphc}_{31} F_1^p(p)))\, A_{\mu}(x)$$

... etc.
Clearly these are just the vector fields F with coefficients frozen at various points. The algorithm is then defined by the update relation:

$$x^{n+1} = \text{exphc}_k F_k^{x^n} \text{o} \text{exphc}_{k-1} F_{k-1}^{x^n} \text{o} \ldots \text{o} \text{exphc}_1 F_1^{x^n}(x^n) \tag{3}$$

*Partially Supported by NSF Grants Nos. DMS-910-1964 and INT 89 - 14643

**Partially Supported by NASA Grant NAG2-513, NSF grant DMS 910-1089

New Multistep Algorithms

The multistep algorithm is simply defined by the update relations:

$$x^{n+1} = \exp hc_0 F^{x_n} o \exp hc_1 F^{x_{n-1}} o \dots o \exp hc_k F^{x_{n-k}} u_1$$

$$u_1 = \exp hc_0^1 F^{x_n} o \exp hc_1^1 F^{x_{n-1}} o \dots o \exp hc_k^1 F^{x_{n-k}} u_2 \quad 4$$

followed by similar expressions terminating with $u_{r+1} = x_n$, for some integer r.

These algorithms are defined by the "step length" h and various sets of constants: c_i and c_{ij}, $1 \le i, j \le k$, $i < j$, in the case of the single step algorithms and c_i, c_i^j, $1 \le i \le n$, $1 \le j \le r$, in the case of multistep algorithms. If the update equations (3) and (4) were satisfied exactly by solutions to the differential equation (1) then they would be satisfied exactly by the relations:

$$x_{n-k} = \exp(-hkF)(x_n), \quad k = -1, 0, 1 \dots \quad 5$$

In general the equations are never satisfied exactly by the relations (5). However, after substituting the relations (5) into the equations (3) and (4), the Taylor expansions in h, about h = 0, of both sides of the equations may be compared. The coefficients of powers of h, depend upon the constants c_i, c_{ij}, and c_i^j, and the algebraic equations in these constants obtained by equating powers of h are called the consistency equations. If the coefficients of h^k agree up to $k = M$, then M is said to be *order* of the resulting algorithm.

Remark 1. It is interesting to view the algorithms in the (trivial) special case where $n = N$, $M = \mathbf{R}^N$ and $A_i = e_i$ is the standard i th basis vector in R^N. In this case we have the following:

$$F(x) = \sum_{i=1}^{N} f^i(x)\, e_i\,, \quad F^p = \sum_{i=0}^{N} f^i(p) e_i = F(p)$$

The single step algorithm (3) reduces to:

$$x^{n+1} = x^n + h(c_1 F_1^{x^n} + c_2 F_2^{x^n} + \dots + c_k F_k^{x^n}) \quad 6$$

where

$$F_1^p = F(p),\ F_2^p = F(p + hc_{21}F_1^p),\ F_3^p = F(p + hc_{31}F_1^p + hc_{32}F_2^p)$$

etc. This is now in the standard form of an explicit Runge-Kutta algorithm and we can always take k = M, the order of the algorithm.

The multistep algorithm (4) reduces to:

$$x^{n+1} = x_n + h(c_0 F(x_n) + c_1 F(x_{n-1}) + \dots + c_k F(x_{n-k})) \quad 7$$

This is now in the form of a standard explicit multistep algorithm and we can always take k = M, the order of the algorithm.

Remark 2. The consistency equations may be derived in different ways. The paper [4] uses the algebra of Cayley trees generated by labeled, ordered trees to derive the equations while the paper [5] derives the consistency equations via a careful geometric analysis of the equations (3) and (4). The results show that new third order Runge Kutta algorithm M = 3 and k = 3, has five independent consistency equations, in the six constants defining the algorithm. Four of these equations are identical to the four consistency equations for the classical Runge Kutta algorithm.

Consistency equations for classical third order Runge Kutta algorithms:

$$c_1 + c_2 + c_3 = 1,\ c_2 c_{21} + c_3(c_{31} + c_{32}) = 1/2$$

$$c_2 c_{21}^2/2 + c_3(c_{31} + c_{32})^2/2 = 1/6,\ c_3 c_{32} c_{21} = 1/6$$

Extra consistency equations for new third order Runge Kutta algorithms:

$$c_2^2 c_{21} + c_3^2 (c_{32} + c_{31}) + 2 c_2 c_3 c_{21} = 1/3$$

This set of five equations has multiple solutions, all of which are solutions to the first four equations, and hence also define classical algorithms, but these solutions are not traditionally used. We detail one set of solutions in the next section, and compare the performance of the resulting new algorithm with that of a classical algorithm.

A new third order multistep algorithm can be obtained with k = 3 and r = 2. This yields four consistency equations in six constants, again with multiple solutions. Three of the consistency equations again correspond to the equations for the classical multistep algorithm, once we set the constants $c_i^j = 0$. Although we have implemented the resulting algorithms we do not detail the results in this paper.

Remark 3. In general, because of numerical round off errors, if traditional algorithms such as those defined by equations (6) and (7) are applied to differential equations which do have the structure of equation (1), the iterates x^n drift off the manifold M. One can modify such a scheme by projecting the iterates x^n back onto M, before computing the next approximation x^{n+1}. The new algorithms described here are more closely associated with the class of "exact" algorithms which have been developed in the past decade to integrate Hamiltonian systems numerically while preserving the "Energy" exactly, see for example the papers [1], [6] and [7].

Remark 4. In the paper [3], the authors analyze two non trivial special cases of the single step algorithm (3), to determine the computational effort required, and make comparisons with a standard single step algorithm. In particular [3] analyzes the cases: M= G, a Lie group where A_μ are left (or right) invariant vector fields on G, and M = G/H, a homogeneous space where A_μ are the vector fields induced by the action of G on M.

Remark 5. The algorithms (3) and (4) (and the analysis described in [5]) do not require that the vector fields F and A_μ are tangent to some submanifold M of $\mathbf{R}^n$. A useful application of this observation is provided by the situation where one wishes to integrate a vector field on $\mathbf{R}^n$ given by an expression:

$$F(x) = \sum_{\mu=1}^{N} b^\mu(x)\, B_\mu x$$

where B_μ are matrices and $b\mu$ are functions. That is we simply require that the vector fields A_μ are in fact linear. The oracle in our assumption must then be capable of solving (or solving to any required accuracy) the linear equations:

$$\dot{x} = (\sum_{\mu=1}^{N} \beta^\mu B_\mu)\, x$$

Both views of the new algorithms are exploited in a paper by the authors [3], to integrate a set of equations of the from:

$$\dot{R} = S(\omega(R))R,\ R \in SO(3) \subset \mathbf{R}^{3\times3}$$

where SO(3) is the subgroup of the 3×3 nonsingular matrices consisting of the rotation matrices and S(a) is a 3×3 skew symmetric matrix for every $a \in \mathbf{R}^3$, satisfying $S(a)b = b \times a$. ($\times$ is the cross product of vectors a, $b \in \mathbf{R}^3$.) The oracle must solve the linear equation with frozen coefficients, $\omega(R) = \alpha$:

$$\dot{R} = S(\alpha)R,\ \alpha \in R^3, R \in SO(3) \quad 8$$

This equation has an explicit solution:

$$R(t) = e^{S(\alpha)t} R(0) = e^{S(c)\phi t} R(0),\ \alpha = c\phi,\ c^T c = 1 \quad 9$$

where

$$e^{S(c)\phi t} = (I_3 + S(c)\sin(\phi t) + S(c)^2(1 - \cos(\phi t))) \quad 10$$

2. Numerical Integration of the Attitude Equations

We are concerned in this paper with the numerical integration of the attitude equations, describing the dynamic evolution of a rigid body, as described say, in the paper by Crouch [2]. The kinematic equations are:

$$\dot{R} = S(\omega)R,\ R \in SO(3) \subset \mathbf{R}^{3\times3} \quad 11$$

where R denotes the attitude as described in the previous section. The dynamic equations are:

$$\dot{\omega} = J^{-1} S(\omega) J \omega,\ \omega \in \mathbf{R}^3 \quad 12$$

where ω is the angular velocity and J is the inertia tensor.

To apply the algorithms (3) and (4) described in the previous section we must first identify a suitable structure as in equation (2). Both the total angular momentum and kinetic energy are conserved quantities in equation (12) and so we could view equation (12) as evolving on either the constant energy surface or the constant momentum surface. However in most control problems we are also faced with the integration of the controlled system [2], in which the dynamics take the following form:

$$\dot{\omega} = J^{-1} S(\omega) J\omega + u$$

where u is the control. In this case there are no conserved quantities in general and so it is appropriate to integrate the dynamic equations (12) using a traditional algorithm. However the kinematic equations (11) always evolve on SO(3), a submanifold of the 3×3 nonsingular matrices and so it is appropriate to integrate these equations using the new algorithms, as is explained in remark 5. To make use of this observation, if x denotes the state (R, ω), then we write the equations (11) and (12) defining the vector field F in the from:

$$\dot{x} = \sum_{i=1}^{4} a^i(x)\, A_i(x) \qquad 13$$

We set $a^0 = 1$ and let A_0 to be the vector field defining the differential equation:

$$\dot{R} = S(\omega)R, \quad \dot{\omega} = 0$$

We let A_i, $1 \le i \le 3$, be the vector fields defining the differential equations:

$$\dot{R} = 0, \quad \dot{\omega} = e_i$$

and finally the coefficients $a^i(x)$, $1 \le i \le 3$, are simply the three components of the three vector:

$$J^{-1} S(\omega) J \omega$$

By freezing the coefficients of F the differential equation (13) becomes:

$$\dot{R} = S(\omega)R, \quad \dot{\omega} = \alpha, \quad R(0) = R_0, \ \omega(0) = \omega_0 \qquad 14$$

Clearly we have:

$$\omega(t) = \alpha t + \omega_0 \qquad 15$$

and hence:

$$\dot{R} = S(\alpha t + \omega_0)R, \quad R(0) = R_0$$

Unlike the equation (8) we are not aware of any explicit solution to this differential equation, except where $\omega_0 = 0$. However we can obtain a series solution to the equation in the following form:

$$R(t) = \left(I + \int_0^t \Theta(s_1)ds_1 + \int_0^t \int_0^{s_1} \Theta(s_1)\Theta(s_2)ds_1 ds_2 + \ldots \right) R_0$$

$$\Theta(t) = S(\alpha t + \omega_0)$$

We have computed the first Q terms in this expression as a symbolic expression in t, using a simple "Mathematica" routine, which we denote by $R^Q(t)$. It follows that we have described an oracle that can compute the flow of F with frozen coefficients, solving the system of equations (14), to any desired accuracy. Using this oracle we may compute solutions of the attitude equations (11) and (12), using the new algorithms described in section 1.

A new third order single step algorithm applied to the equations (11) and (12) is denoted algorithm 3. We measure the performance of our algorithms relative to the fifth order Runge Kutta algorithm available in the IMSL library, which we call algorithm 1. A classical third order Runge Kutta algorithm is tested as a comparison and denoted by algorithm 2. We note that our new third order single step algorithm simply reduces to the classical algorithm for the angular velocity ω (equation 15). Thus it is to be expected that the new third order algorithm will give bad results when compared with the fifth order IMSL routine. However, since the angular velocity ω in equation (12) evolves independently of the angular position R in equation (11), we construct an hybrid algorithm, denoted algorithm 4, in which we integrate equation (12) by the IMSL routine, while we integrate equation (11) using the new third order algorithm (we use the IMSL computed value of the angular velocity ω to initiate the computation of the angular position R at each step in the algorithm.) Summarizing we compare the performance of four algorithms:

Algorithm 1. The fifth order Runge-Kutta algorithm found in the I.M.S.L. package. We always use the step length of h = 0.01, in algorithm 1, unless otherwise stated.

Algorithm 2. The classical "Kutta" algorithm, which is an example of a third order classical Runge-Kutta algorithm, using the coefficients:

$$c_1 = 1/6, c_2 = 2/3, c_3 = 1/6,$$

$$c_{21} = 1/2, c_{31} = -1, c_{32} = 2.$$

Algorithm 3. The new single step third order algorithm defined by the constants:

$$c_1 = 1, c_2 = -2/3, c_3 = 2/3,$$

$$c_{21} = -1/24, c_{31} = 161/24, c_{32} = -6.$$

We use the oracle defined by the approximate solution $R(t) = R^Q(t)$, $\omega(t) = \alpha t = + \omega_0$, to compute the solution of the system with frozen coefficients (14).

Algorithm 4. A hybrid algorithm which uses the fifth order IMSL Runge Kutta algorithm to compute an update to the angular velocity ω, and algorithm 3 to compute an update to the orientation R. In the R update we use the value of ω computed from the previous application of the IMSL routine and the value of R computed in the previous R update step.

3. Numerical Results

Extensive numerical tests were carried out on the numerical integration of the attitude equations (11) and (12) using the four algorithms described above. All numerical experiments were performed on a VAX 6000-420 and implemented in Fortran in double precision. Equation (12) was simulated with J, the inertia matrix being diagonal with entries (1, 3, 2). To simulate the kinematic equations (11) it is sufficient to simulate the evolution of three orthonormal vectors making up the initial state R_0. Thus we detail the results of the evolution of only one vector r which therefore remains constrained to lie on a sphere S^2. As the initial state of r we took:

$$r_1 = r_2 = r_3 = 1$$

so that an iterate r^n remains on S^2 as long as

$$e^n = (r_1^n)^2 + (r_2^n)^2 + (r_3^n)^2$$

satisfies $e^n = 3$. We took the initial state of ω, to be the same as that of r.

In the tables below, for each algorithm we give the error e^n - 3, and the differences Δr_1, $\Delta \omega_1$ between the first components of r and ω, respectively, computed using algorithm 1 and the algorithm in question.

4. Discussion

Table 1 shows the degradation in performance of the IMSL code due to an increase in the step size from 0.01 to 0.05. Not shown in these results is the fact that algorithm 1 yields no error in the computation of e^n (= 3). Table 2 shows the much worse performance of the classical third order Kutta algorithm. Tables 3 and 4 demonstrate the effectiveness of the new third order single step algorithm. The computation of the flow of the frozen coefficients equations was done using Q = 10. This was roughly the minimum truncation which ensured that the $e^n = 3$ during the course of the simulation. Notice that this is achieved even as the errors in the state variables build up quite significantly. Tables 5, 6 and 7 demonstrate the effectiveness of the hybrid algorithm. For Q = 8, errors in the value of e^n are just evident, while for Q = 10 they are absent as in algorithm 3. Clearly the hybrid algorithm is

Table 1. Difference between Algorithm 1 (h=0.01) and Algorithm 1 (h=0.05)

t (secs.)	Δr_1	$\Delta \omega_1$	e^n -3
5	-.10E-8	.00E+0	-.40E-9
10	-.20E-9	.00E+0	-.60E-9
15	-.31E-8	.00E+0	-.20E-8
20	-.70E-9	-.10E-9	-.17E-8
25	.49E-8	.00E+0	-.19E-8
30	-.10E-9	.00E+0	-.70E-9
35	.63E-8	-.10E-9	.03E-9
40	.85E-8	.00E+0	.30E-9
45	-.57E-8	.10E-9	.60E-9
50	.50E-9	.00E+0	-.14E-8
55	-.39E-8	.00E+0	-.80E-9
60	-.18E-7	.00E+0	-.19E-8
65	-.13E-8	.00E+0	-.16E-8
70	-.60E-9	.00E+0	-.60E-9
75	-.78E-8	-.10E-9	.20E-9
80	.21E-7	.10E-9	.80E-9
85	.13E-7	.20E-9	-.50E-9
90	.50E-9	-.20E-9	.50E-9
95	.20E-7	-.40E-9	-.18E-8
100	-.84E-8	.10E-9	-.11E-8

Table 2. Difference between Algorithm 1 and Algorithm 2 (h=0.05)

t (secs.)	Δr_1	$\Delta \omega_1$	e^n -3
5	.21E-3	.79E-5	-.54E-3
10	-.12E-3	.88E-4	-.12E-2
15	.89E-5	.82E-4	-.18E-2
20	-.10E-3	-.15E-3	-.25E-2
25	-.28E-2	-.30E-3	-.33E-2
30	-.11E-2	-.27E-4	-.39E-2
35	-.13E-2	.65E-3	-.45E-2
40	-.39E-2	.47E-3	-.51E-2
45	.33E-2	-.10E-2	-.58E-2
50	.16E-2	-.10E-2	-.63E-2
55	.41E-2	.11E-2	-.70E-2
60	.15E-1	.17E-2	-.77E-2
65	.20E-2	-.32E-3	-.83E-2
70	-.53E-2	-.24E-2	-.91E-2
75	.31E-2	-.12E-2	-.96E-2
80	-.18E-1	.32E-2	-.10E-1
85	-.15E-1	.28E-2	-.11E-1
90	-.37E-2	-.36E-2	-.11E-1
95	-.23E-1	-.41E-2	-.12E-1
100	.14E-1	.27E-2	-.13E-1

Table 3. Differences between Algorithm 1 and Algorithm 3 (h=0.05, Q = 10)

t (secs.)	Δr_1	$\Delta \omega_1$	e^n -3
5	.67E-4	-.56E-5	.00E+0
10	.75E-4	.99E-4	.00E+0
15	.78E-3	.12E-3	.00E+0
20	.48E-4	-.15E-3	.00E+0
25	-.21E-2	-.37E-3	.00E+0
30	-.72E-3	-.61E-4	.00E+0
35	-.27E-2	.73E-3	.00E+0
40	-.39E-2	.56E-3	.00E+0
45	.25E-2	-.11E-2	.00E+0
50	-.11E-2	-.12E-2	.00E+0
55	.47E-2	.11E-2	.00E+0
60	.16E-1	.18E-2	.00E+0
65	.25E-2	-.31E-3	.00E+0
70	-.17E-2	-.26E-2	.00E+0
75	.39E-2	-.13E-2	.00E+0
80	-.19E-1	.34E-2	.00E+0
85	-.13E-1	.30E-2	.00E+0
90	-.66E-2	-.38E-2	.00E+0
95	-.28E-1	-.44E-2	.00E+0
100	.14E-1	.29E-2	.00E+0

Table 4. Differences between Algorithm 1 and Algorithm 3 (h=0.1, Q = 10)

t (secs.)	Δr_1	$\Delta \omega_1$	e^n -3
5	.53E-3	-.26E-4	.00E+0
10	.56E-3	.77E-3	.00E+0
15	.62E-2	.94E-3	.00E+0
20	.36E-3	-.12E-2	.00E+0
25	-.16E-1	-.28E-2	.00E+0
30	-.57E-2	-.42E-3	.00E+0
35	-.21E-1	.57E-2	.00E+0
40	-.31E-1	.43E-2	.00E+0
45	.21E-1	-.87E-2	.00E+0
50	-.60E-2	-.91E-2	.00E+0
55	.37E-1	.89E-2	.00E+0
60	.13E+0	.14E-1	.00E+0
65	.19E-1	-.26E-2	.00E+0
70	-.23E-1	-.20E-1	.00E+0
75	.25E-1	-.97E-2	.00E+0
80	-.15E+0	.27E-1	.00E+0
85	-.11E+0	.23E-1	.00E+0
90	-.33E-1	-.30E-1	.00E+0
95	-.19E+0	-.34E-1	.00E+0
100	.12E+0	.23E-1	.00E+0

Table 5. Differences between Algorithm 1 and Algorithm 4 (h=0.05, Q = 8)

t (secs.)	Δr_1	$\Delta \omega_1$	e^n -3
5	.43E-6	.00E+0	.00E+0
10	.41E-5	.00E+0	.00E+0
15	.41E-5	.00E+0	-.10E-9
20	-.12E-5	-.10E-9	-.10E-9
25	-.32E-5	.00E+0	-.10E-9
30	-.35E-6	.00E+0	-.20E-9
35	-.38E-5	-.10E-9	-.20E-9
40	-.63E-5	.00E+0	-.20E-9
45	.10E-5	.10E-9	-.20E-9
50	-.18E-5	.00E+0	-.30E-9
55	.67E-5	.00E+0	-.30E-9
60	.10E-4	.00E+0	-.30E-9
65	.73E-6	.00E+0	-.30E-9
70	.71E-6	.00E+0	-.40E-9
75	.49E-5	-.10E-9	-.40E-9
80	-.98E-5	.10E-9	-.40E-9
85	-.72E-5	.20E-9	-.40E-9
90	-.46E-5	-.20E-9	-.50E-9
95	-.12E-4	-.40E-9	-.50E-9
100	.78E-5	.10E-9	-.50E-9

Table 6. Differences between Algorithm 1 and Algorithm 4 (h=0.05, Q = 10)

t (secs.)	Δr_1	$\Delta \omega_1$	e^n -3
5	.43E-6	.00E+0	.00E+0
10	.41E-5	.00E+0	.00E+0
15	.41E-5	.00E+0	.00E+0
20	-.12E-5	-.10E-9	.00E+0
25	-.32E-5	.00E+0	.00E+0
30	-.35E-6	.00E+0	.00E+0
35	-.38E-5	-.10E-9	.00E+0
40	-.63E-5	.00E+0	.00E+0
45	.10E-5	.10E-9	.00E+0
50	-.18E-5	.00E+0	.00E+0
55	.67E-5	.00E+0	.00E+0
60	.10E-4	.00E+0	.00E+0
65	.73E-6	.00E+0	.00E+0
70	.71E-6	.00E+0	.00E+0
75	.49E-5	-.10E-9	.00E+0
80	-.98E-5	.10E-9	.00E+0
85	-.72E-5	.20E-9	.00E+0
90	-.46E-5	-.20E-9	.00E+0
95	-.12E-4	-.40E-9	.00E+0
100	.78E-5	.10E-9	.00E+0

Table 7. Differences between Algorithm 1 and Algorithm 4 (h=0.1, Q = 10)

t (secs.)	Δr_1	$\Delta \omega_1$	$e^n - 3$
5	.11E-4	.40E-9	.00E+0
10	.34E-4	.00E+0	.00E+0
15	.46E-4	-.11E-8	.00E+0
20	-.12E-4	-.26E-8	.00E+0
25	-.47E-4	.00E+0	.00E+0
30	-.21E-5	-.60E-9	.00E+0
35	-.56E-4	.26E-8	.00E+0
40	-.88E-4	.73E-8	.00E+0
45	.32E-4	-.54E-8	.00E+0
50	-.14E-4	-.14E-7	.00E+0
55	.72E-4	.56E-8	.00E+0
60	.15E-3	.22E-7	.00E+0
65	.82E-5	.36E-8	.00E+0
70	.79E-5	-.32E-7	.00E+0
75	.69E-4	-.27E-7	.00E+0
80	-.16E-3	.41E-7	.00E+0
85	-.11E-3	.51E-7	.00E+0
90	-.41E-4	-.44E-7	.00E+0
95	-.17E-3	-.69E-7	.00E+0
100	.10E-3	.26E-7	.00E+0

superior to the new algorithm on its own. The better performance of the hybrid algorithm (table 6) in ensuring that $e^n = 3$ must be weighed against the better performance of the classical 5 th order algorithm (table 1), in the error made in computing the state variable r. Note however that the hybrid algorithm simply uses a third order algorithm, to integrate the evolution of the state r.

The results demonstrate the effectiveness of the new class of algorithms in integrating differential equations evolving on a geometric structure. The extra computational costs involved in these algorithms must be weighed against the need to preserve the geometric structure. Nevertheless such algorithms may prove beneficial in certain applications.

5. References

[1] M. Austin, P.S. Krishnaprasad and L-S Wang, "Almost Poisson Integration of Rigid Body Systems," Systems Research Center Technical Report, SRC TR 91-45, (1991).

[2] P. E. Crouch, "Spacecraft Attitude Control and Stabilization: Application of Geometric Control to Rigid Body Models," IEEE Trans. Automatic Control, Vol. AC - 29, pp. 321 - 331, (1986).

[3] P. E. Crouch, R. Grossman and Y. Yan, "A Third Order Runge-Kutta Algorithm on a Manifold," Submitted, (1992).

[4] P. E. Crouch, R. Grossman and R. Larson, "Computations Involving Differential Operators and Their Actions on Functions," Proc. 1991 Int. Sym. on Symbolic and Algebraic Computation, ACM, (1991).

[5] P. E. Crouch and R. Grossman, "Numerical Integration of Ordinary Differential Equations on Manifolds," To appear in J. of Nonlinear Science, (1991).

[6] Ge-Zhong and J. Marsden, "Lie-Poisson Hamiltonian-Jacobi Theory and Lie-Poisson Integration," Phys. Lett. A., Vol.133, pp. 134-139, (1988).

[7] J.M. Sanz-Serna, "Runge-Kutta Schemes for Hamiltonian Systems," Bit, Vol 28, pp. 877 - 883, (1988).