ARTICLE NO. 80-4087

Improved Local Linearization Algorithm for Solving the Quaternion Equations

Kenneth Yen* and Gerald Cook†
University of Virginia, Charlottesville, Va.

The objective of this paper is to develop a new and more accurate local linearization algorithm for numerically solving sets of linear time-varying differential equations. Of special interest is the application of this algorithm to the quaternion rate equations. The results are compared, both analytically and experimentally, with previous results using local linearization methods. The new algorithm requires approximately one-third more calculations per step than the previously developed local linearization algorithm; however, this disadvantage could be reduced by using parallel implementation. For some cases the new algorithm yields significant improvement in accuracy, even with an enlarged sampling interval. The reverse is true in other cases. The errors depend on the values of angular velocity, angular acceleration, and integration step size. One important result is that for the worst case the new algorithm can guarantee eigenvalues nearer the region of stability than can the previously developed algorithm.

Introduction

DUE to the extensive use of digital computers in the past decade, people have developed many integration algorithms. These range from the Euler method, which is very simple but less accurate, to the Runge-Kutta method, which is very accurate but time consuming.[1,2] In real-time digital simulation, we desire a numerical algorithm that is both accurate and fast.[3,4] Unfortunately, it is a general rule that there is a tradeoff between speed and accuracy; that is, a more accurate method would be more time consuming.

The second-order Adams-Bashforth method (AB-2) is an algorithm which has been used to solve the quaternion rate equations that are widely used for determining the orientation of missiles and aircraft in real-time, both in actual flight and in simulation problems.[1,4] Here the use of more computation permits the use of simpler instrumentation. The AB-2 method is accurate enough for moderate angular velocity, but with the advent of high-performance aircraft we encounter high angular velocities which make the AB-2 method less accurate. To improve this situation, a more accurate integration algorithm based on local linearization (LL) was developed.[1] Based on this same reasoning, we have developed a new algorithm (NLL) which is an extension of the LL algorithm as an attempt to provide further improvement. In this paper, the NLL algorithm is compared with the LL algorithm both analytically and experimentally. The comparison includes both stability analysis and error analysis. In one of the examples simulated, the NLL algorithm has the more accurate behavior, while in another example the LL algorithm is better. In general, however, the NLL algorithm is expected to provide the better performance because of its superior stability properties.

Development of the NLL

The quaternion equations can be written in matrix form,

$$\dot{X}(t) = A(t)X(t)$$

where

$$A(t) = \frac{1}{2}\begin{bmatrix} 0 & -r(t) & -q(t) & -p(t) \\ r(t) & 0 & -p(t) & q(t) \\ q(t) & p(t) & 0 & -r(t) \\ p(t) & -q(t) & r(t) & 0 \end{bmatrix} \tag{1}$$

and with p, q, and r, being pitch, roll, and yaw rates, respectively.

One might expect the solution of Eq. (1) to be of the form

$$X(t) = \exp\left(\int_{t_0}^{t} A(t)\,dt\right)X(t_0)$$

where

$$e^{At} = I + At + A^2t^2/21,\ t \ldots$$

but, in general, this is not true.

The most general solution of Eq. (1) is $X(t) = \Phi(t,t_0)X(t_0)$, where $\Phi(t,t_0)$ is the solution of $\dot{\Phi}(t,t_0) = A(t)\Phi(t,t_0)$ and $\Phi(t_0,t_0) = I$.[5,6] Unfortunately, even for a very simple time-varying system, it is difficult to determine $\Phi(t,t_0)$ analytically.[7]

For the NLL algorithm, we first expand $A(t)$ about t_k

$$A(t) \approx A(t_k) + \dot{A}(t_k)(t-t_k)$$

Letting $A(t_k) = A_k$ and $\dot{A}(t_k) = \dot{A}_k$, we have

$$\dot{X}(t) = A_kX(t) + \dot{A}_k(t-t_k)X(t) \tag{2}$$

Although A_k, $\dot{A}_k$ are now constants, it is still difficult to find a closed-form analytic solution of Eq. (2) due to the $tX(t)$ term in the second part of the right side. One has to make a further approximation in this term. Let

$$X(t) = X(t_k) + \dot{X}(t_k)(t-t_k) \tag{3}$$

Letting $X(t_k) = X_k$ and $\dot{X}(t_k) = \dot{X}_k$, and substituting Eq. (3) into the second part of the right side of Eq. (2), we get

$$\dot{X}(t) = A_kX(t) + \dot{A}_kX_k(t-t_k) + \dot{A}_k\dot{X}_k(t-t_k)^2 \tag{4}$$

Received July 18, 1979; revision received Jan. 15, 1980.

Index categories: Spacecraft Navigation, Guidance and Flight-Path Control; Simulation.

*Graduate Student, Dept. of Electrical Engineering.
†Professor, Dept. of Electrical Engineering.

The first term in the right side of Eq. (2) has been left exact since it presents no complication.

Because $\dot{X}_k = A_k X_k$, we can rewrite Eq. (4) as

$$\dot{X}(t) = A_k X(t) + \dot{A}_k X_k (t-t_k) + \dot{A}_k A_k X_k (t-t_k)^2 \quad (5)$$

Comparing with the LL algorithm,

$$\dot{X}(t) = A_k X(t) + \dot{A}_k X_k (t-t_k) \quad (6)$$

we find that the NLL algorithm has an extra term, $\dot{A}_k A_k X_k (t-t_k)^2$.

The solution of Eq. (5) is

$$X_{k+1} = e^{A_k T} X_k + A_k^{-2} [e^{A_k T} - I - A_k T] \dot{A}_k X_k + 2A_k^{-3} [e^{A_k T} - I - A_k T - A_k^2 T^2/2] \dot{A}_k A_k X_k \quad (7)$$

where

$$T = t_{k+1} - t_k$$

Equation (7) is the NLL algorithm.

If $A(t)$ is constant, $\dot{A}_k = 0$, Eq. (7) becomes $X_{k+1} = e^{A_k T} X_k$, which corresponds to a time-invariant system.

Simulation

We have used the following two sets of data for testing the NLL and LL algorithm, respectively:

Data set 1

$$\begin{aligned} p &= 10 \sin 0.5t \\ q &= 2 \sin t \\ r &= q \end{aligned}$$

Data set 2

$$\begin{aligned} p &= 5 \sin 0.25t \\ q &= 0.25 \cos 12t \\ r &= 0.25 \sin 12t \end{aligned}$$

Because we did not have angular acceleration data as a direct input, we used the following strategies: For $K=0$ we used $(A_1 - A_0)/T$ to approximate $\dot{A}_0$, and for $K \geq 1$ we used $(A_{k+1} - A_{k-1})/2T$ to approximate $\dot{A}_k$. Note that our solution is for time $k+1$; therefore, A_{k+1} is available to use in forming $\dot{A}_k$.

One may wonder why the angular velocities are not differentiated electronically to obtain the acceleration signals. Certainly this is possible, however, the signals obtained would generally be quite noisy. For this reason the finite differencing operation is proposed.

If $w_k = 0$, it means that no rotation occurs; we omit the calculation and X_{k+1} will be equal to X_k.

The calculations required by the two algorithms were evaluated with respect to computation time. It was assumed that addition requires T seconds, multiplication $2T$ seconds, square roots $4T$ seconds, and trigonometric functions $10T$ seconds. Some economy was achieved through noting the appearance of certain terms several times in both the algorithms. With the above assumptions, it was found that computing X_{k+1} given A_{k+1} and X_k required $154T$ seconds using the LL algorithm and $203T$ seconds using the NLL algorithm, or approximately one third more time using the NLL algorithm. Some of this increase could be eliminated through the use of parallel operations.

The figures compare the accuracy of the simulations in terms of deviation from the "ideal" solution (the one obtained using fourth-order Runge-Kutta with a very small integration interval).

In Figs. 1 and 2, we use approximately three cycles of output data to calculate the mean-square errors (MSE). In Figs. 3 and 4, we illustrate how the mean-square error varied with simulation time. It is obvious that for data set 1 the NLL algorithm yields a better result than the LL algorithm, but for data set 2 the situation is reversed. We will discuss these in the section on stability and error analysis. These calculations were performed at the University of Virginia on a Cyber 170 series computer.

Stability and Error Analysis

The eigenvalues of the system [Eq. (1)] which has the quaternion matrix as its system matrix are purely imaginary with multiplicity two, that is, $\pm(w/2)i$, $\pm(w/2)i$ where $w^2 = p^2 + q^2 + r^2$. One difficulty is that all classical explicit methods which are suitable for real-time simulation have stability boundaries that do not include the imaginary axis except at the origin.[1] Therefore, any truncation or roundoff errors[4] introduced in the solution process will, in general, not die out and the integral curve can diverge.

The NLL algorithm can be written as

$$X_{k+1} = M_k X_k \quad (8)$$

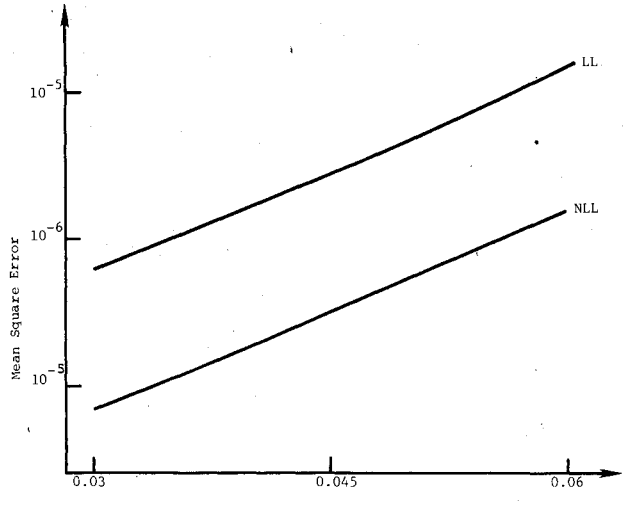

Fig. 1 Relations between MSE and step size for LL and NLL algorithm, data set 1.

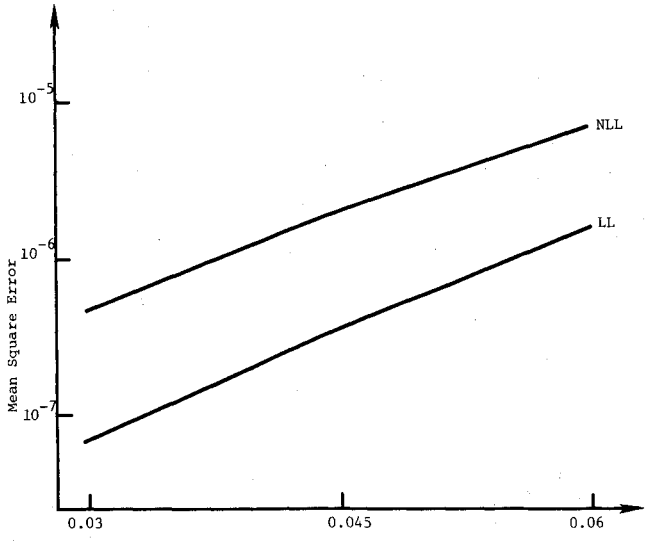

Fig. 2 Relations between MSE and simulation time for LL and NLL algorithm, data set 2.

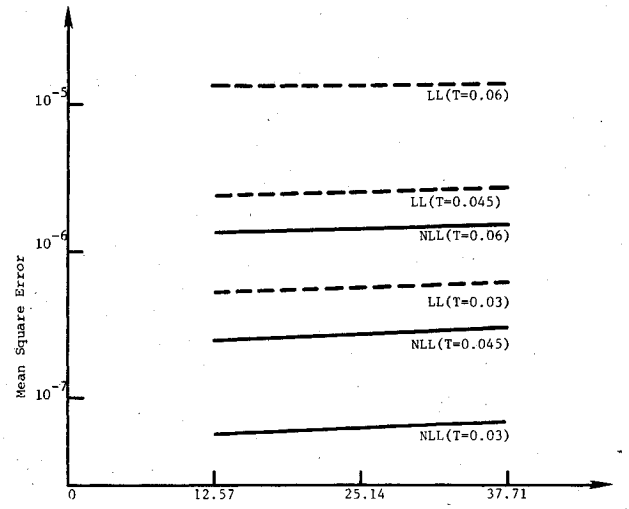

Fig. 3 Relations between MSE and different step size for LL and NLL algorithm for different step size.

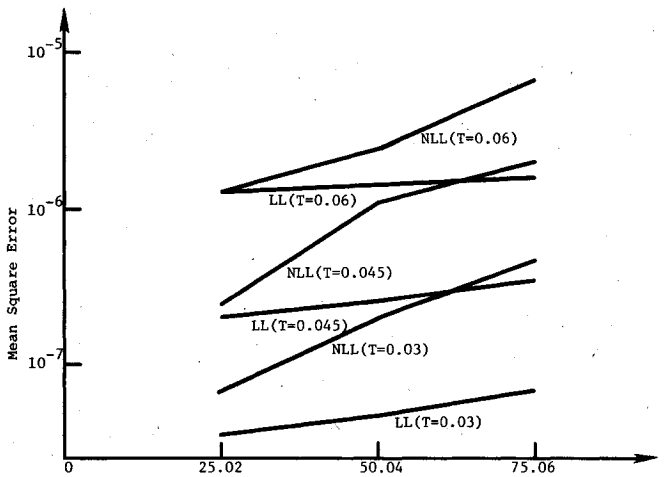

Fig. 4 Relations between MSE and simulation time for LL and NLL algorithm for different step size.

where

$$M_k = H\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} + \begin{bmatrix}0&-G&-J&-K\\G&0&-K&J\\J&K&0&G\\K&-J&-G&0\end{bmatrix} \tag{9}$$

with

$$H = C1 + (C4+C5)S1$$

$$G = \tfrac{1}{2}(C2r_k + C3\dot{r}_k) + (C5-C4)S2 + C6S5$$

$$J = \tfrac{1}{2}(C2q_k + C3\dot{q}_k) + (C5-C4)S3 + C6S6$$

$$K = \tfrac{1}{2}(C2p_k + C3\dot{p}_k) + (C5-C4)S4 + C6S7$$

where

$$C1 = \cos\tfrac{1}{2}W_kT$$

$$C2 = (2/W_k)\sin\tfrac{1}{2}W_kT$$

$$C3 = (4/W_k^2)(1-\cos\tfrac{1}{2}W_kT)$$

$$C4 = (4/W_k^2)[T(2/W_k)\sin\tfrac{1}{2}W_kT]$$

$$C5 = 2C4$$

$$C6 = (4/W_k^2)[T^2 - (8/W_k^2)(1-\cos\tfrac{1}{2}W_kT)]$$

$$S1 = \tfrac{1}{4}(r_k\dot{r}_k + q_k\dot{q}_k + p_k\dot{p}_k)$$

$$S2 = \tfrac{1}{4}(p_k\dot{q}_k - q_k\dot{p}_k)$$

$$S3 = \tfrac{1}{4}(r_k\dot{p}_k - p_k\dot{r}_k)$$

$$S4 = \tfrac{1}{4}(q_k\dot{r}_k - r_k\dot{q}_k)$$

$$S5 = \tfrac{1}{8}[\dot{r}_k(p_k^2+q_k^2-r_k^2) - 2r_kq_k\dot{q}_k - 2r_kp_k\dot{p}_k]$$

$$S6 = \tfrac{1}{8}[\dot{q}_k(p_k^2-q_k^2+r_k^2) - 2q_kp_k\dot{p}_k - 2q_kr_k\dot{r}_k]$$

$$S7 = \tfrac{1}{8}[\dot{p}_k(-p_k^2+q_k^2+r_k^2) - 2p_kq_k\dot{q}_k - 2p_kr_k\dot{r}_k]$$

The eigenvalues of the matrix M_k are

$$\xi_1 = \xi_3 = H + ir \qquad \xi_2 = \xi_4 = H - ir \tag{10}$$

where $r = \sqrt{G^2+J^2+K^2}$.
The magnitude of ξ_j $(j=1,\ldots,4)$ is given as

$$|\xi_j|_k^2 = 1 + [(Y/Z)P(Z) + (Y/Z)^2Q(Z)]_k + (T^2\dot{W}_k^2/4)R^2(Z) - T^2\dot{W}_k(\sin Z)R(Z) \tag{11}$$

where

$$Y = \dot{W}_kT^2 \qquad Z = W_kT/2$$

$$P(Z) = \sin Z/Z - \cos Z - 2\cos Z(1-\sin Z/Z)$$

$$Q(Z) = (5/4Z^2)(1-\cos Z)^2 + (9/4)(1-\sin Z/Z)^2 - \tfrac{1}{2}(1-\cos Z)$$

$$R(Z) = 1 - 2(1-\cos Z)/Z^2$$

Because the eigenvalues of $A(t)$ are on the imaginary axis, a good discrete time algorithm would necessarily have root locations on the unit circle, i.e., $|\xi_j|_k^2 = 1$. The error in $|\xi_j|_k^2$ is defined as

$$\text{Error}_k = (Y/Z)P(Z) + (Y/Z)^2Q(Z)_k + (T^4\dot{W}_k^2/4)R^2(Z) - T^2\dot{W}_k(\sin Z)R(Z) \tag{12}$$

For $W_k \neq 0$ and $\dot{W}_k \neq 0$,

$$\lim_{t-0} \quad \text{error}_k = 0$$

that is, the method is consistent.

For sufficiently small T,

$$|\xi_j|_k^2 = 1 + \dot{W}_k^2T^4/16 + 0(T^5) \tag{13}$$

The corresponding result for LL[1] is

$$|\xi_j|_k^2 = 1 + \dot{W}_k^2T^4/16 + \dot{W}_kW_kT^3/6 + 0(T^5) \tag{14}$$

The implications of these equations will be discussed later. Let us now turn our attention to Eq. (8).

For any rotation step from k to $k+1$, there exists a transition matrix M which exactly represents the rotation step. This is true regardless of whether the step is infinitesimal or finite. In fact, the true M is the matrix equivalent of the

quaternion associated with the rotation. One source of error in both the LL and NLL algorithms is that, for a finite rotation, the matrix M computed is at best a good approximation of the true transition matrix. In the general case, M is the sum of a true term MT and an error term ME; $M=MT+ME$, and also the state X_k is the sum of the true state $X(t_k)$ and an error term E_k. Using these definitions, the state transition equation

$$X_{k+1}=M_kX_k$$

becomes

$$X(t_{k+1})+E_{k+1}=(MT_k+ME_k)(X(t_k)+E_k)+r(t_k,T)$$

Subtracting $X(t_{k+1})=MT_kX(t_k)$ from both sides yields the error propagation equation

$$E_{k+1}=ME_kX(t_k)+M_kE_k+r(t_k,T) \tag{15}$$

It is seen that the behavior of E_k depends not only on M_k but also on ME_k. By making M_k have very small eigenvalues, the homogeneous behavior of Eq. (15) would die out very rapidly. However, if the eigenvalues of M_k are very small, this means that M_k must deviate considerably from the true value for M (since the eigenvalues for the true M have magnitudes of unity) and, therefore, ME_k will be large. Thus the solution is not simply to choose an M_k with arbitrarily small eigenvalues.

On the other hand, if the eigenvalues of M_k are greater than unity, one not only has the presence of the $ME_kX(t_k)$ term as a forcing function in Eq. (15) but also unstable homogeneous behavior.

Thus, one simple measure of accuracy of solution is the deviation of the eigenvalues of M_k from unity in magnitude. (Note that having eigenvalues of unity magnitude is a necessary but not sufficient condition for accurate performance.) Looking at Eqs. (13) and (14), certain observations can be made in this regard.

If W_k and $\dot{W}_k$ are of the same sign, it is obvious that the NLL algorithm is superior to the LL algorithm. In aerospace applications, the angular acceleration $\dot{W}_k$ must eventually change sign; the eigenvalues of NLL algorithm will always lie outside the unit circle; however, the eigenvalues of LL algorithm will not always lie outside the unit circle. In this situation, it is really hard to compare the two algorithms. One may be better some of the time and the other better at other times. This can also be seen from Figs. 1-4.

Based on Eqs. (13) and (14), one can make comparisons on the least upper bounds of the eigenvalues. For the LL algorithm, dropping terms in T of order five and higher, one has

$$|\xi_j|_k^2 \le 1+|\dot{W}_k^2T^4/16|+|\dot{W}_kW_k|T^3/6$$

while for the NLL algorithm one has

$$|\xi_j|_k^2 \le 1+|\dot{W}_k^2T^4/16|$$

Thus, a guarantee of stability in the worst case is more nearly satisfied for the NLL algorithm than for the LL algorithm.

Conclusion

A new local linearization algorithm has been developed for solving the quaternion rate equations. This algorithm is slightly more complex and requires one third more computation time than the previously developed local linearization algorithm. For two examples tested, the new algorithm performed better in one case and poorer in the other. Based on a stability analysis, however, the new method does guarantee better behavior under the worst case.

Acknowledgments

The authors gratefully acknowledge the support of NASA Langley Research Center under Grant No. NSG-1403 for the research reported herein. Also, we appreciate the helpful comments of those who reviewed the paper.

References

[1] Conte, S.D. and de Boor, C., *Elementary Numerical Analysis,* McGraw-Hill Book Co., New York, 1965.

[2] Gerald, C.F., *Applied Numerical Analysis,* Addison-Wesley Book Co., Reading, Mass., 1978.

[3] Lawrence, E.B., Jr., Bowles, R.L., and Williams, L.H., "Development and Application of a Local Linearization Algorithm for the Integration of Quaternion Rate Equations in Real-Time Flight Simulation Problems," NASA TN D-7347, 1973.

[4] Wilson, J.W. and Steinmetz, G.G., "Analysis of Numerical Integration Techniques for Real-time Digital Flight Simulation," NASA TN D-4900, 1968.

[5] Rosko, J.S., *Digital Simulation of Physical Systems,* Addison-Wesley Book Co., Reading, Mass., 1972.

[6] Brogan, W., *Modern Control Theory,* Quantum Press Inc., New York, 1974.

[7] D'Angelo, H., *Linear Time-Varying Systems: Analysis and Synthesis,* Allyn and Bacon, Boston, 1970.

[8] Todd, J. (ed.), *Survey of Numerical Analysis,* McGraw-Hill Book Co., New York, 1962, pp. 323-326.