A common, avoidable source of error in molecular dynamics integrators

Ross A. Lippert, Kevin J. Bowers, Ron O. Dror, Michael P. Eastwood, Brent A. Gregersen, John L. Klepeis, and Istvan Kolossvary
D. E. Shaw Research, LLC, New York, New York 10036

David E. Shaw[a)]
D. E. Shaw Research, LLC, New York, New York 10036 and Center for Computational Biology and Bioinformatics, Columbia University, New York, New York 10032

(Received 24 October 2006; accepted 11 December 2006; published online 29 January 2007)

[DOI: 10.1063/1.2431176]

The most popular integration methods in molecular dynamics (MD) simulations are the Verlet algorithm[1] and its velocity-explicit brethren, leapfrog and velocity Verlet[2] (Table I), which integrate the differential equation $\ddot{x}=a(x(t))$. In exact arithmetic, from appropriate starting conditions, their approximations $x_i \sim x(i\delta_t)$ are identical, with error proportional to the square of the time step δ_t. In finite-precision arithmetic, on the other hand, the velocity-explicit algorithms are recognized as more accurate than Verlet,[3] and this is one reason why they are used by most widespread MD codes. Many of these codes, however, add a modification like $v_{i+1/2}=(x_{i+1}-x_i)/\delta_t$ to compute a modified velocity from a modified position when run with constraints. We show that this modification degrades the numerical accuracy of such integrators to that of Verlet. Once recognized, the problem can be easily remedied.

The velocity-explicit algorithms have an advantage in finite-precision arithmetic because storing $v_{i+1/2}$ or v_i, instead of x_{i-1}, is a better use of the available precision. In typical simulations, x_i and x_{i-1} usually have some k leading bits in common, so k bits of the difference x_i-x_{i-1} will be zero, and thus carry no information. Demonstrations of this disadvantage are easily made; the amplitude drift of a simulated harmonic oscillator is larger using Verlet than leapfrog or velocity Verlet. Alternatively, consider a one-dimensional system under constant acceleration $a(x)=A>0$ starting from $(x,v)=(1,1)$ at $t=0$. With relative arithmetic precision ϵ (i.e., $1+\delta$ evaluates to 1 if $|\delta|<\epsilon/2$), the acceleration affects the trajectory under Verlet only if $A \geqslant \delta_t^{-2}\epsilon/2$. For the velocity-explicit algorithms, values of A above $\delta_t^{-1}\epsilon/2$ (leapfrog) or $\delta_t^{-1}\epsilon$ (velocity Verlet) affect the trajectory. This suggests that velocity-explicit algorithms are more sensitive to acceleration and this difference is more pronounced in lower precision arithmetic.

By mediating the flow of information from acceleration, $a(x_i)$, to position, x_i, the velocity data accumulate values too small to have an immediate effect on x_i, but which may have a cumulative effect later. Any inversion of this relationship is justly viewed with suspicion.

For illustration, consider a modified leapfrog,

$$\tilde{v}_{i+1/2} = v_{i-1/2} + \delta_t a(x_i), \tag{1}$$

$$x_{i+1} = x_i + \delta_t \tilde{v}_{i+1/2}, \tag{2}$$

$$v_{i+1/2} = \frac{x_{i+1}-x_i}{\delta_t}. \tag{3}$$

Substituting Eqs. (3) and (1) into Eq. (2) gives a numerically equivalent iteration step for x,

$$x_{i+1} = x_i + \delta_t\left(\frac{x_i - x_{i-1}}{\delta_t} + \delta_t a(x_i)\right). \tag{4}$$

In the constant acceleration example, $A<\delta_t^{-2}\epsilon/2$ has no effect, as for Verlet. Moreover, in standard floating point, if $\delta_t=2^k$ for some integer k then Eq. (4) is numerically equivalent to

$$x_{i+1} = x_i + ((x_i - x_{i-1}) + \delta_t^2 a(x_i)),$$

highlighting its similarity to Verlet.[16] Using the equation $x_{i+1}=x_i+\delta_t v_{i+1/2}$ to find $v_{i+1/2}$ removes v from its place in the information flow. Although Eq. (3) is a gratuitous addition, recomputation of velocity from successive positions is a tempting expedient when integrating in the presence of position constraints.

Constraints can be incorporated into the Verlet algorithm with the update rule

$$x_{i+1} = ((2x_i - x_{i-1}) + \delta_t^2 a(x_i)) + \delta_t^2 A(x_i)\lambda_i, \tag{5}$$

where $A(x_i)$ is a matrix whose columns are normal to the constraint surface at x_i and the Lagrange multiplier λ_i is a vector determined by demanding that constraints be satisfied at x_{i+1}. SHAKE (Ref. 4) is an iterative method that determines a corrected x_{i+1} from an uncorrected $\tilde{x}_{i+1}=(2x_i-x_{i-1})+\delta_t^2 a(x_i)$, avoiding explicit calculation of λ_i. M-SHAKE (Ref. 5) is a different iterative solver. SETTLE (Ref. 6) is a closed-form $\tilde{x}_{i+1}\rightarrow x_{i+1}$ correction suitable for rigid water molecules.

A common adaptation of SHAKE to leapfrog replaces Eq. (2) with

$$\tilde{x}_{i+1} = x_i + \delta_t \tilde{v}_{i+1/2}, \tag{2'}$$

$$x_{i+1} = \text{SHAKE}(\tilde{x}_{i+1}), \tag{2''}$$

reducing to Eqs. (1)–(3) in the absence of constraints. The LINCS algorithm[7] also uses Eq. (3) for velocity correction [their Eq. (12)], though it is easily repaired [Eq. (15)].

TABLE I. Common integration methods: parentheses indicate the customary computational ordering (Ref. 3).

Verlet	$x_{i+1}=(2x_i-x_{i-1})+\delta_t^2 a(x_i)$
Leapfrog	$v_{i+1/2}=v_{i-1/2}+\delta_t a(x_i)$ $x_{i+1}=x_i+\delta_t v_{i+1/2}$
Velocity Verlet	$x_{i+1}=x_i+\delta_t\left(v_i+\frac{1}{2}\delta_t a(x_i)\right)$ $v_{i+1}=\left(v_i+\frac{1}{2}\delta_t a(x_i)\right)+\frac{1}{2}\delta_t a(x_{i+1})$

One could avoid Eq. (3) by an alternative discretization, algebraically equivalent to Eq. (5),

$$v_{i+1/2}=v_{i-1/2}+\delta_t a(x_i)+\delta_t A(x_i)\lambda_i, \tag{6}$$

$$x_{i+1}=x_i+\delta_t v_{i+1/2}, \tag{7}$$

where λ_i is determined so that constraints are satisfied at x_{i+1}. RATTLE (Ref. 8) uses the velocity Verlet version of this discretization with an additional velocity correction to ensure tangency to the constraint surface ($v_i \cdot A(x_i)=0$). A textbook[3] supplementary FORTRAN program F.9 demonstrates an iterative RATTLE method similar to SHAKE.

Two of the most widely used MD codes, AMBER (Ref. 9) (Version 9, sander and pmemd) and GROMACS (Ref. 10) (Version 3.3.1), use Eq. (3) whenever constraints are active. Another code, NAMD (Ref. 11) (Version 2.6), incorporates F.9, but the default constraint method uses SETTLE with Eq. (3) for water atoms. Another popular MD code, CHARMM (Ref. 12) (Version 33), uses a more accurate alternative to Eq. (3),

$$v_{i+1/2}=\tilde{v}_{i+1/2}+\frac{x_{i+1}-\tilde{x}_{i+1}}{\delta_t}, \tag{3'}$$

which infers the contact acceleration, $(x_{i+1}-\tilde{x}_{i+1})/\delta_t^2$, rather than the velocity, from the corrected positions. This solution is not as accurate as methods based on the RATTLE discretization; the acceleration can be computed more accurately by directly solving for the λ_i in Eq. (6). MD codes have traditionally run in double precision, where the degradation from Eq. (3) is insignificant, but it may become problematic if other codes follow the lead of GROMACS in using single-precision arithmetic to accelerate MD computations.

To assess the impact of Eq. (3′) on a single-precision simulation, we incorporated Eq. (3′) into GROMACS (a simpler code modification than RATTLE). Starting from an equilibrated 30 Å^3 cube of water (901 molecules, periodic), we ran an *NVE* simulation for 1 ns with $\delta_t=1$ fs, using single-precision GROMACS, our modified version, and corresponding double-precision versions. We selected SETTLE for computation of constrained positions to avoid considerations of iterative methods and parameters. Figure 1 shows that even this partial fix substantially reduces energy drift in single precision. We also plot the same results for single-precision Desmond,[13] which uses the RATTLE discretization with a variant of M-SHAKE. Other differences between Desmond and GROMACS may also contribute to differences in results.

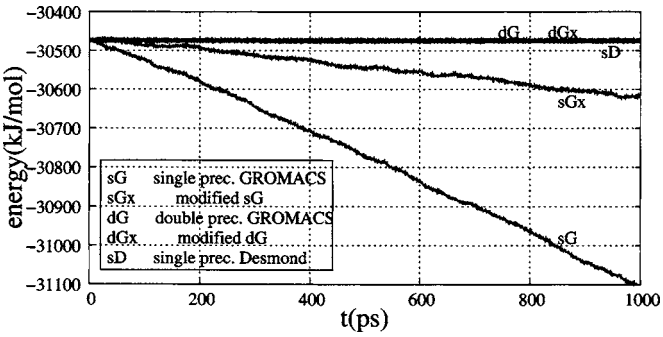

FIG. 1. Energy drift for our test simulations using SPC water (Ref. 14), smooth particle mesh Ewald (Ref. 15) with fourth order B splines, $1/\beta=2.96$ Å, a 1 Å mesh spacing, 10 Å van der Waals and 12 Å Coulomb cutoffs, and neighbor list updates every time step ($\delta_t=1$ fs).

[a] Author to whom correspondence should be addressed. Electronic mail: david@deshaw.com

[1] L. Verlet, Part. Part. Syst. Charact. **159**, 98 (1967).
[2] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson, J. Chem. Phys. **76**, 637 (1982).
[3] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Oxford University Press, Oxford, 1987).
[4] J. -P. Ryckaert, G. Ciccotti, and H. J. C. Berendsen, J. Comput. Chem. **23**, 327 (1977).
[5] V. Kräutler, W. F. van Gunsteren, and P. H. Hünenberger, J. Comput. Chem. **22**, 501 (2001).
[6] S. Miyamoto and P. A. Kollman, J. Comput. Chem. **13**, 952 (1992).
[7] B. Hess, H. Bekker, H. J. C. Berendsen, and J. G. E. M. Fraaije, J. Comput. Chem. **18**, 1463 (1997).
[8] H. C. Andersen, J. Comput. Phys. **52**, 24 (1983).
[9] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. Woods, J. Comput. Chem. **26**, 1668 (2005).
[10] D. van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. C. Berendsen, J. Comput. Chem. **26**, 1701 (2005).
[11] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, J. Comput. Chem. **26**, 1781 (2005).
[12] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, J. Comput. Chem. **4**, 187 (1983).
[13] K. J. Bowers, E. Chow, H. Xu *et al.*, in Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC06), Tampa, Florida (2006).
[14] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, and J. Hermans, in *In Intermolecular Forces*, edited by B. Pullman (Reidel, Dordrecht, 1981), pp. 331–342.
[15] U. Essman, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen, J. Chem. Phys. **103**, 8577 (1995).
[16] The result is similar if δ_t is not such a power, with extra roundoff terms from the arithmetic.