Physics 115/242

Leapfrog method and other "symplectic" algorithms for integrating Newton's laws of motion

Peter Young

I. THE LEAPFROG ALGORITHM

We have already seen in our discussion of numerical differentiation and of numerical integration (midpoint method) that the slope of a chord between two points on a function, (x_0, f_0) and (x_1, f_1), is a much better approximation of the derivative at the midpoint $f'_{1/2}$ than at either end. We can use the same idea in a simple, elegant method for integrating Newton's laws of motion, which takes advantage of the property that the equation for dx/dt does not involve x itself and the equation for dv/dt (v is the velocity) does not involve v. More precisely, for a single degree of freedom, the equations of motion are

$$\frac{dx}{dt} = v \tag{1}$$

$$\frac{dv}{dt} = F(x) \left(= -\frac{dU(x)}{dx} \right) \tag{2}$$

where $F(x)$ is the force on the particle when it is at x, $U(x)$ is the potential energy, and for simplicity we set the mass equal to unity. (To put back the mass replace F by F/m throughout.)

The Euler method would approximate Eq. (1) by

$$x_1 = x_0 + h v_0, \tag{3}$$

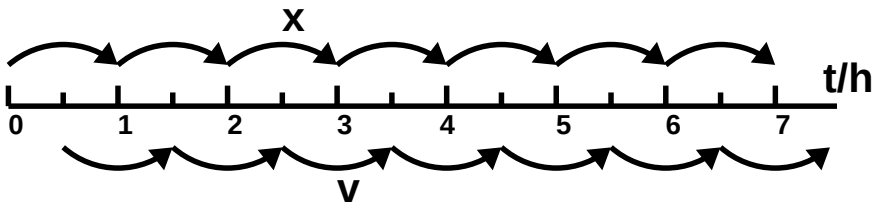
where as usual h is the interval between time steps. A better approximation would be to replace v by its value at the midpoint of the interval, i.e.

$$x_1 = x_0 + h v_{1/2}. \tag{4}$$

Of course, you would protest that we don't yet know $v_{1/2}$ so how can we use this. Let's finesse this for now and assume that we *can* get $v_{1/2}$ in some way. Then we can immediately apply a similar midpoint rule to Eq. (2) to step v forward in time, i.e.

$$v_{3/2} = v_{1/2} + h F(x_1), \tag{5}$$

since we *do* know x_1. Then we can step forward x with $x_2 = x_1 = h v_{3/2}$ and so on. Thus, once we have started off with x_0 and $v_{1/2}$ we can continue with x and v leapfrogging over each other as shown in the figure below.



The basic integration formula for the leapfrog algorithm is therefore

$$x_{n+1} = x_n + hv_{n+1/2} \tag{6}$$

$$v_{n+3/2} = v_{n+1/2} + hF(x_{n+1}). \tag{7}$$

How accurate is this approach? Well $x_1 - x_0$ is of order h, and we showed earlier in the class that the expected leading error $\sim h^2$ vanishes for the midpoint approximation, and so the error for one interval is $\sim h^3$. To integrate over a finite time the number of intervals $\propto 1/h$ and so the overall error is of order h^2. Leapfrog is therefore a second order method, like RK2, and better than Euler, which is only first order. We shall see shortly that, in addition to leapfrog being of higher order than Euler even though it is hardly more complicated, it has other desirable features connected with its *global* properties.

The leapfrog method has a long history. I don't know who first introduced it but there is a nice discussion in the Feynman Lectures on Physics, Vol. I, Sec. 9.6.

II. THE VELOCITY VERLET ALGORITHM

To make leapfrog useful, however, two questions have to be addressed. The first, which we have already mentioned, is how to we start since we need $v_{1/2}$, but we only have the initial velocity v_0 (as well as the initial position x_0). The simplest approximation is just to do a single half step

$$v_{1/2} = v_0 + \tfrac{1}{2}hF(x_0). \tag{8}$$

Although this is not a midpoint method, and so has an error of h^2 we only do this *once* so it does not lower the order of the method, which remains second order. The second question to be addressed is how can we get the velocity at the same time as the position, which is needed, for example, to produce "phase space" plots (see below) and to to compute the energy and angular momentum. The simplest approach is just to consider Eq. (7) to be made up of two equal half steps, which successively relate v_{n+1} to $v_{n+1/2}$ and $v_{n+3/2}$ to v_{n+1}. The leapfrog algorithm with

a means of starting the algorithm and determining x and v at the same times in the way just mentioned, is called *velocity Verlet*. A single time step can be written as

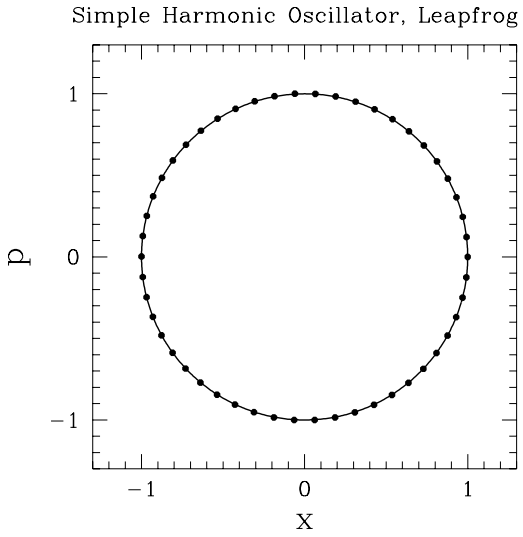
$$\begin{aligned} v_{n+1/2} &= v_n + \tfrac{1}{2}hF(x_n) \\ x_{n+1} &= x_n + hv_{n+1/2} \\ v_{n+1} &= v_{n+1/2} + \tfrac{1}{2}hF(x_{n+1}). \end{aligned} \tag{9}$$

If we are not interested in v at integer times, this is exactly the same as Eqs. (6) and (7), apart from the initial computation of $v_{1/2}$. It looks as though we have to do two force calculations per time step in Eq. (9) but this is not so because the force in the third line is the same as the force in the first line of the *next step*, so it can be stored and reused. Since velocity Verlet is the same as leapfrog, it is a second order method.

It is often useful to show the trajectory as a "phase space" plot i.e. the path in the p-x plane (where $p = mv$ is the momentum). As an example consider the simple harmonic oscillator for which the energy is given by $E = p^2/2m + kx^2/2$, where k is the spring constant. Here we have set $m = 1$ and we will also set $k = 1$ so

$$E = \frac{p^2}{2} + \frac{x^2}{2} \qquad (= \text{const.}) \tag{10}$$

Hence the phase space plot is a circle with radius equal to $\sqrt{2E}$.



The above figure shows a phase space plot for one period of a simple harmonic oscillator using the velocity Verlet method with time step $h = 0.02T$, where $T = 2\pi$ is the period. The starting

values are $x = 1, v = 0$, so $E = 1/2$. The figure shows that the leapfrog/velocity Verlet method correctly follows the circular path in phase space (at least for one period T).

Note that instead of starting with a half step for v followed by full step for x and another half step for v, one could do the opposite: a half step for x followed by full step for v and another half step for x, i.e.

$$\begin{aligned} x_{n+1/2} &= x_n + \tfrac{1}{2}hv_n \\ v_{n+1} &= v_n + hF(x_{n+1/2}) \\ x_{n+1} &= x_{n+1/2} + \tfrac{1}{2}hv_{n+1}. \end{aligned} \tag{11}$$

This is called *position Verlet*. Once the algorithm has been started it is the same as velocity Verlet.

It is trivial to generalize the equations of the leapfrog/Verlet method to the case of more than one position and velocity. For example, for the position Verlet algorithm one has

$$\begin{aligned} x^i_{n+1/2} &= x^i_n + \tfrac{1}{2}hv^i_n \qquad\qquad (i = 1, \cdots, N) \\ v^i_{n+1} &= v^i_n + hF^i(\{x_{n+1/2}\}) \quad (i = 1, \cdots, N) \\ x^i_{n+1} &= x^i_{n+1/2} + \tfrac{1}{2}hv^i_{n+1} \qquad (i = 1, \cdots, N), \end{aligned} \tag{12}$$

where $x^i, v^i (i = 1, 2, \cdots, N)$ are the positions and velocities, and $F^i(\{x\})$ is the force which gives the acceleration of the coordinate x^i, i.e. $F^i(\{x\}) = -\partial U(\{x\})/\partial x^i$, where U is the potential energy. Each force depends, of course, on the set of all positions $\{x\}$. It is important that *all* the positions are updated in the first line of Eq. (12), then all the forces are calculated using the new positions, then all the velocities are updated in the second line, and finally, all the positions are updated again in the last line.

III. THE VERLET ALGORITHM

If we are not interested in the velocities, but just the trajectory of the particle, we can eliminate the velocities from the algorithm since

$$x_{n+2} = x_{n+1} + hv_{n+3/2} \tag{13}$$

$$= x_{n+1} + h(v_{n+1/2} + hF(x_{n+1})). \tag{14}$$

Now $hv_{n+1/2}$ can be written as $x_{n+1} - x_n$ and so we get an equation entirely for the x_n:

$$x_{n+2} = 2x_{n+1} - x_n + h^2F(x_{n+1}). \tag{15}$$

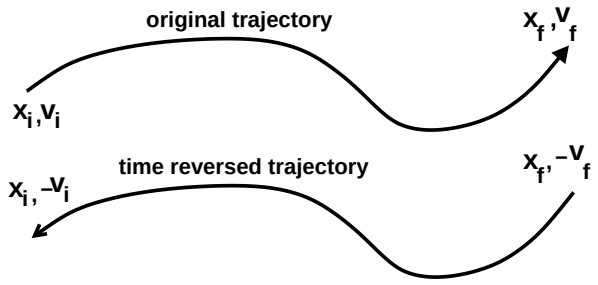
This *Verlet algorithm* is less often used than velocity or position Verlet because: (i) having the velocities is frequently useful, (ii) the Verlet algorithm is not self starting, and (iii) it is more susceptible to roundoff errors because it involves adding a very small term of order h^2 to terms of order unity. By contrast, the velocity or position Verlet schemes only add terms of order h (which is larger than h^2 since h is small) to terms of order unity.

IV. ADVANTAGES OF THE LEAPFROG/(VELOCITY OR POSITION) VERLET ALGORITHM

In addition to combining great simplicity with second order accuracy, the leapfrog/(velocity or position) Verlet algorithm has several other desirable features:

1. It is time reversal invariant.

   Newton's equations of motion are invariant under time reversal. What this means is as follows. Suppose we follow a trajectory from x_i at some initial time t_i (when the particle has velocity v_i) to x_f at a later time t_f (when the velocity is v_f). Now consider the time reversed trajectory which starts, at time t_i, at position x_f but with the *opposite* velocity $-v_f$. Then, at time t_f, the particle will have reached the initial position x_i and the velocity will be $-v_i$, see the figure.



   This time reversed trajectory is what you would see if you took a movie of the original trajectory and ran it backwards. Thus both the trajectory forward in time and the one backwards in time are possible trajectories. Since this is an exact symmetry of the equations, it is desirable that a numerical approximation respect it.

   It is a matter of simple algebra to check that Eqs. (9) or (11) respect time reversal. Start with $x_0 = X, v_0 = V$, say, and determine x_1 and v_1. Then start the time reversed trajectory with $x_0^r = x_1$ and $v_0^r = -v_1$. The three steps of Eqs. (9) or (11) in the reversed trajectory

correspond precisely to the three steps in the forward trajectory (with positions the same and velocities reversed) but in the *reverse order*.

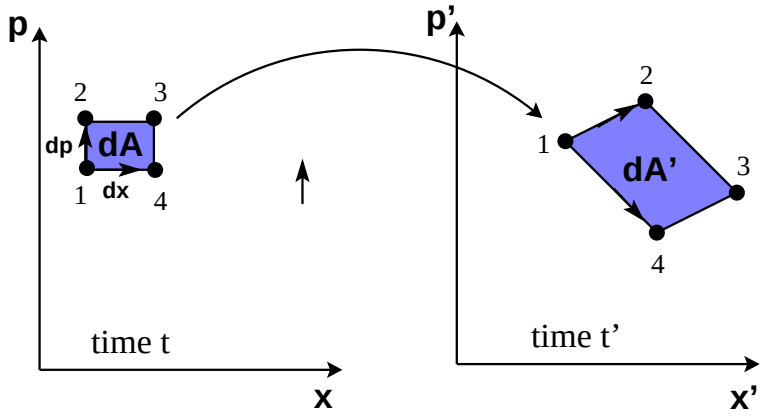
2. In a spherically symmetric potential, angular momentum is conserved and, remarkably, the leapfrog/(velocity or position) Verlet algorithm conserves it *exactly*. If the potential energy U only depends on the magnitude of $\vec{r}$ and not its direction then the force is along the direction of $\vec{r}$, i.e.

$$\vec{F}(r) = -\hat{r}\,\frac{dU(r)}{dr}\,, \tag{16}$$

where $\hat{r}$ is a unit vector in the direction of $\vec{r}$. It is left as a homework exercise to show that the leapfrog algorithm conserves angular momentum for such a force. (Unfortunately energy is *not* exactly conserved in the algorithm.)

It is obviously desirable that a numerical approximation respect symmetries exactly, and I'm not aware of *any other* algorithms which conserve angular momentum, though they may exist. This is obviously a "plus" for the leapfrog algorithm.

3. The leapfrog/(velocity or position) Verlet algorithm is "symplectic", i.e. area preserving. I will now explain this important concept. Consider a small rectangular region of phase space of area dA as shown in the left part of the figure below.



Let the four corners of the square, $(x,p),(x+dx,p),(x,p+dp),(x+dx,p+dp)$ represent four possible coordinates of a particle at time t. These are labeled 1, 2, 3, 4. Then, at a later time t' each of these four points will have changed, to form the corners of a parallelogram, as shown on the right of the figure. Let the area of the parallelogram be dA'. An important theorem (Liouville's theorem) states that the areas are equal, i.e.

$$dA' = dA. \tag{17}$$

I have not been able to find a simple derivation of Liouville's theorem. For a more advanced text which gives a proof, see Landau and Lifshitz, *Classical Mechanics*. Now (x, p) transforms to (x', p'), where x' and p' are some (complicated non-linear) functions of x, and p, i.e.

$$\begin{aligned} x' &= X(x, p) \\ p' &= P(x, p). \end{aligned} \tag{18}$$

A set of equations like (18), in which the values of one set of variables (x, p here), is transformed to new values (x', p' here), is called a *map*. Thus, the result of integration of Newton's laws by a finite amount of time can be represented as an *area preserving map*.

Since the area preserving property is an exact feature of the equations, it is desirable that a numerical approximation preserve it. Such approximations are called *symplectic*.

What is the condition for a map to be symplectic? To see this we need to compute the area dA' in the above figure, and set it equal to $dA = dx\, dp$. The area dA' is given by

$$dA' = |d\vec{e}_1' \times d\vec{e}_2'|, \tag{19}$$

where $d\vec{e}_1'$ and $d\vec{e}_2'$ are the vectors describing the two sides of the parallelogram, $1 \to 4$ and $1 \to 2$. Now the components of $d\vec{e}_1'$ are just the changes in x' and p' when x is changed by dx but p is fixed, i.e.

$$d\vec{e}_1' = \left(\frac{\partial x'}{\partial x}\, \hat{x} + \frac{\partial p'}{\partial x}\, \hat{p} \right) dx, \tag{20}$$

and similarly

$$d\vec{e}_2' = \left(\frac{\partial x'}{\partial p}\, \hat{x} + \frac{\partial p'}{\partial p}\, \hat{p} \right) dp. \tag{21}$$

It is well known that the vector product in Eq. (19) can be represented as a determinant, and so we get

$$dA' = J dA, \tag{22}$$

where

$$J = \begin{vmatrix} \dfrac{\partial x'}{\partial x} & \dfrac{\partial p'}{\partial x} \\ \dfrac{\partial x'}{\partial p} & \dfrac{\partial p'}{\partial p} \end{vmatrix} = \begin{vmatrix} \dfrac{\partial x'}{\partial x} & \dfrac{\partial x'}{\partial p} \\ \dfrac{\partial p'}{\partial x} & \dfrac{\partial p'}{\partial p} \end{vmatrix} \tag{23}$$

is the Jacobian of the transformation from (x', p') to (x, p) which occurs when you change variables in an integral, i.e.

$$\iint \ldots \, dx'dp' = \iint \ldots \, J \, dxdp. \tag{24}$$

In Eq. (23) we have noted the a determinant of the transpose of a matrix is the same as that of the original matrix.

To show that the leapfrog algorithm is symplectic it is convenient to consider each of the three steps in Eq. (9) separately (remember we have $m = 1$ here so p and v can be used interchangeably):

$$\begin{pmatrix} \delta x_1 \\ \delta v_1 \end{pmatrix} = C \begin{pmatrix} \delta x_1 \\ \delta v_{1/2} \end{pmatrix}, \qquad \begin{pmatrix} \delta x_1 \\ \delta v_{1/2} \end{pmatrix} = B \begin{pmatrix} \delta x_0 \\ \delta v_{1/2} \end{pmatrix}, \qquad \begin{pmatrix} \delta x_0 \\ \delta v_{1/2} \end{pmatrix} = A \begin{pmatrix} \delta x_0 \\ \delta v_0 \end{pmatrix}, \tag{25}$$

where

$$C = \begin{pmatrix} 1 & 0 \\ \frac{h}{2}F'(x_1) & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & h \\ 0 & 1 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 0 \\ \frac{h}{2}F'(x_0) & 1 \end{pmatrix}, \tag{26}$$

so

$$\begin{pmatrix} \delta x_1 \\ \delta v_1 \end{pmatrix} = J \begin{pmatrix} \delta x_0 \\ \delta v_0 \end{pmatrix}, \tag{27}$$

where J is given by the matrix product

$$J = CAB\,. \tag{28}$$

It is an important (but not sufficiently well known) theorem that the determinant of a product of matrices is the the product of the determinants, i.e.

$$\det J = \det C \, \det B \, \det A\,. \tag{29}$$

By inspection, $\det A = \det B = \det C = 1$, and so $\det J = 1$, i.e. the leapfrog/velocity Verlet algorithm is symplectic.

However, J is not equal to unity for the other algorithms that we have considered, Euler, RK2 and RK4. (Since RK4 is very accurate the change in area will be small, of order h^4, but not zero.)

The advantage of symplectic algorithms is that they possess a sort of global stability. Since the area bounded by adjacent trajectories is preserved, we can never have the situation

that we clearly saw for the Euler algorithm, where the coordinates (and hence the energy) increase without bound, because this would expand the area. Even in better non-symplectic approximations, such as RK2 and RK4, the energy will eventually deviate substantially from its initial value.

In fact one can show that the results from an approximate symplectic integrator are equal to the *exact* dynamics of a "close by" Hamiltonian, $\mathcal{H}'(h)$ where, for the case of a second order method like leapfrog,

$$\mathcal{H}'(h) = \mathcal{H} + (\ldots)\, h^2 + (\ldots)\, h^3 + \cdots , \tag{30}$$

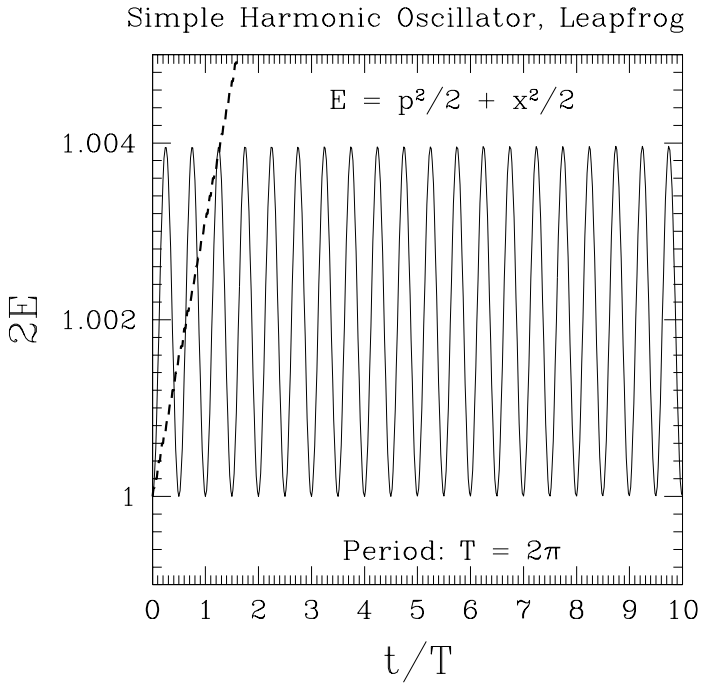
in which

$$\mathcal{H} = \frac{p^2}{2m} + V(x) \tag{31}$$

is the actual Hamiltonian and $(\ldots)$ represent the extra pieces of $\mathcal{H}'(h)$.

V. NUMERICAL RESULTS

We now show some numerical data which illustrates the symplectic behavior of the leapfrog algorithm. As usual, we take the simple harmonic oscillator, with time step $h = 0.02T$ (where T is the period).



The above figure shows that although the energy deviates from the exact value, it never wanders far from the exact result. By contrast, in the RK2 algorithm the energy deviates more and more from the exact value as t increases, as shown by the thick dashed line in the figure. The energy in the leapfrog method oscillates around the correct value because the method is symplectic. Note that for small times (less than a quarter period), the error with leapfrog is actually rather worse than with RK2 (though both are of order h^2). It is in the long time behavior that leapfrog is better since it has "global stability".

VI. HIGHER ORDER SYMPLECTIC ALGORITHMS

Recently there has been interest in *higher order* algorithms which respect time reversal invariance and which are symplectic. The simplest higher order symplectic algorithm is that of E. Forest and R.D. Ruth, Physica D, **43**, 105 (1990), and extensions discussed in e.g. Omelyan, Mryglod and Folk, `http://arxiv.org/abs/cond-mat/0110585`, which are of fourth order. If initially $x = x(t), v = v(t)$ then, in the Forest-Ruth algorithm, the following steps

$$\begin{aligned} x &= x + \theta\frac{h}{2}v \\ v &= v + \theta h F(x) \\ x &= x + (1-\theta)\frac{h}{2}v \\ v &= v + (1-2\theta)hF(x) \\ x &= x + (1-\theta)\frac{h}{2}v \\ v &= v + \theta h F(x) \\ x &= x + \theta\frac{h}{2}v, \end{aligned} \tag{32}$$

with

$$\theta = \frac{1}{2 - \sqrt[3]{2}} \simeq 1.35120719195966, \tag{33}$$

yield $x(t+h), v(t+h)$ and so generate one time step. Note that this method requires three evaluations of the force per time step, as opposed to just one for leapfrog. Note too that the steps are symmetric about the middle one (this ensures time reversal invariance). Since each step involves simply moving forward either the position or the velocity, the Jacobian of the transformation can be written, as for the leapfrog method discussed above, as a the

product of determinants (7 here) each of which trivially has determinant unity. Hence the algorithm is symplectic. The hard work is to show that the algorithm gives an error of order h^5 for one interval (and hence of order h^4 when integrated over $n = \Delta t/h$ time steps for a fixed time increment Δt). This is where the strange value for θ comes from.

It is curious that the middle step of the Forest-Ruth algorithm is larger in magnitude than h (and it, and some of the other steps, go "backwards in time"). This large step turns out out to be necessary to get a 4-th order symplectic algorithm requiring only three force evaluations per time step. If one is willing to accept more than 3 force evaluations one can avoid having a step greater in magnitude than h, e.g. the "PEFRL" algorithm of Omelyan et al. in Eq. (34) below. However, to my knowledge, all higher order symplectic algorithms have some steps which go backwards in time.

One time step in the "Position Extended Forest-Ruth Like" (PEFRL) algorithm of Omelyan et al. is

$$\begin{aligned} x &= x + \xi h v \\ v &= v + (1-2\lambda)\frac{h}{2}F(x) \\ x &= x + \xi h v \\ v &= v + \lambda h F(x) \\ x &= x + (1-2(\chi+\xi))h v \\ v &= v + \lambda h F(x) \\ x &= x + \xi h v \\ v &= v + (1-2\lambda)\frac{h}{2}F(x) \\ x &= x + \xi h v \end{aligned} \tag{34}$$

with

$$\begin{aligned} \xi &= +0.1786178958448091\text{E}{+}00 \\ \lambda &= -0.2123418310626054\text{E}{+}00 \\ \chi &= -0.6626458266981849\text{E}{-}01 \end{aligned} \tag{35}$$

This algorithm requires 4 force evaluations per time step rather than 3 for Forrest-Ruth, but it is more accurate, as we will see below, since it avoids the large time step.

The following table shows results for the maximum error in $2E$ over one period ($2E = 1$ for

the specified initial conditions $x = 1, v = 0$) for the leapfrog algorithm, Forest-Ruth (FR) algorithm, and the PEFRL algorithm of Omelyan et al.

h/T	leapfrog	FR	PEFRL
0.02	3.949×10^{-3}	1.912×10^{-5}	7.206×10^{-7}
0.005	2.468×10^{-4}	7.416×10^{-8}	2.822×10^{-9}

By comparing the errors for the two different values of h/T one can see that the error in the leapfrog algorithm varies as h^2 while that in the FR and PEFRL algorithms varies as h^4. Furthermore the PEFRL algorithm is about 26 times more accurate than the FR algorithm for the same value of h.

To make a fair comparison of the efficiency of the leapfrog and PEFRL algorithms, one should note that the PEFRL algorithm requires 4 times as many function evaluations per step. Hence we compare PEFRL with $h/T = 0.02$ and leapfrog with $h/T = 0.005$, which require the same number of steps; the result is that PEFRL is about still about 340 times more accurate.

One frequently obtains detailed dynamical information about interacting classical systems from "molecular dynamics" (MD) simulations, which require integrating Newton's equations of motion over a long period of time starting from some initial conditions. Since global stability is important for integration over long times, fourth-order symplectic algorithms are likely to play a major role in MD simulations in the future, since these algorithms combine relative simplicity with accuracy and global stability. However I agree with Omelyan et al. that symplectic integrators of order greater than 4 are probably not worth the extra complexity.