Solving the equations of motion

[This section is copied to a large extent from Antti Kuronen's MD course notes, which in turn rely heavily on Allen-Tildesley]

- In MD, what we really want to do is solve the equations of motion of N atoms (or particles in general) interacting via a potential U

- Lagrange equations of motion:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_k}\right) - \frac{\partial L}{\partial q_k} = 0;$$

$$L(\mathbf{q}, \dot{\mathbf{q}}) = K(\mathbf{q}, \dot{\mathbf{q}}) - U(\mathbf{q}, \dot{\mathbf{q}})$$

$\mathbf{q}$ = generalized coordinate

- Now, if we use the cartesian coordinates $q_k = r_k$; $K(\dot{\mathbf{r}}) = \sum_i \frac{1}{2} m_i \dot{r}_i^2$ and $U = U(\mathbf{r})$, we get the equations of motion back to the familiar (Newtonian) form

$$m_i \ddot{\mathbf{r}}_i = \mathbf{f}_i,$$

where $\mathbf{f}_i = \nabla_{\mathbf{r}_i} L = -\nabla_{\mathbf{r}_i} U$ is the force acting in atom i

- We can also start by considering the Hamiltonian equations of motion

$$\dot{q}_k = \frac{\partial H}{\partial p_k}$$

$$\dot{p}_k = -\frac{\partial H}{\partial q_k},$$

where $p_k = \frac{\partial L}{\partial \dot{q}_k}$ is the generalized momentum

and $H(\mathbf{q}, \mathbf{p}) = \sum_k \dot{q}_k p_k - L(\mathbf{q}, \dot{\mathbf{q}})$ the Hamiltonian function

(we assume that $\dot{q}_k$ can be given as a function of $\mathbf{p}$)

- Now, if U does not depend on the velocities, we get quickly back to the familiar form

$$H(\mathbf{q}, \mathbf{p}) = K(\mathbf{p}) + U(\mathbf{q})$$

and if we again use cartesian coordinates the equations of motion will be:

$$\dot{\mathbf{r}}_i = \frac{\mathbf{p}_i}{m_i}$$

$$\dot{\mathbf{p}}_i = -\nabla_{\mathbf{r}_i} U = \mathbf{f}_i$$

- So we have two alternatives:

  1. Solve $3N$ 2:nd order differential equations ($m_i \ddot{\mathbf{r}}_i = \mathbf{f}_i$) derived from the Lagrangian or Newtonian formalism

  2. Solve $6N$ 1:st order equations derived from the Hamiltonian formalism

Numerical solution of equations of motion

- 'finite difference' -method: from a system configuration (atom positions, velocities etc.) on time t we calculate the configuration at time $t + \delta t$
  - δt can be constant or variable (cf. lecture 2).
  - initial conditions $\mathbf{r}(0)$, $\mathbf{v}(0)$ have to be known

- As an example a 'predictor-corrector' -algorithm:

  - Use a Taylor series to **predict** the system configuration at time $t + \delta t$ using the small deviation δt:

$$\mathbf{r}^{\mathrm{p}}(t + \delta t) = \mathbf{r}(t) + \delta t \mathbf{v}(t) + \frac{1}{2}\delta t^2 \mathbf{a}(t) + \frac{1}{6}\delta t^3 \mathbf{b}(t) + \ldots$$

$$\mathbf{v}^{\mathrm{p}}(t + \delta t) = \mathbf{v}(t) + \delta t \mathbf{a}(t) + \frac{1}{2}\delta t^2 \mathbf{b}(t) + \ldots$$

$$\mathbf{a}^{\mathrm{p}}(t + \delta t) = \mathbf{a}(t) + \delta t \mathbf{b}(t) + \ldots$$

$$\mathbf{b}^{\mathrm{p}}(t + \delta t) = \mathbf{b}(t) + \ldots$$

(p=predicted)

  - **v**, **a** and **b** are higher time derivatives of **r,** as usual in a Taylor series. **v** corresponds to velocity, **a** to acceleration and **b** to the time derivative of acceleration.

  - We can instead of $\mathbf{b}$ also use information from previous time steps:

$$\{\mathbf{r}(t), \mathbf{v}(t), \mathbf{v}(t - \delta t), \mathbf{v}(t - 2\delta t)\}$$

  or

$$\{\mathbf{r}(t), \mathbf{v}(t), \mathbf{a}(t), \mathbf{a}(t - 2\delta t)\}$$

  - **correction** step: we now have $\mathbf{r}^{\mathrm{p}}$, from which we can get the forces $t + \delta t$

    $\Rightarrow$ accurate corrected accelerations $\mathbf{a}^{\mathrm{c}}(t + \delta t)$

    $\Rightarrow$ error in accelerations $\Delta\mathbf{a}(t + \delta t) = \mathbf{a}^{\mathrm{c}}(t + \delta t) - \mathbf{a}^{\mathrm{p}}(t + \delta t)$

- Using this known error, one can calculate corrected positions, velocities and so on

$$\begin{aligned}
\mathbf{r}^{c}(t+\delta t) &= \mathbf{r}^{p}(t+\delta t) + c_0 \Delta\mathbf{a}(t+\delta t) \\
\mathbf{v}^{c}(t+\delta t) &= \mathbf{v}^{p}(t+\delta t) + c_1 \Delta\mathbf{a}(t+\delta t) \\
\mathbf{a}^{c}(t+\delta t) &= \mathbf{a}^{p}(t+\delta t) + c_2 \Delta\mathbf{a}(t+\delta t) \\
\mathbf{b}^{c}(t+\delta t) &= \mathbf{b}^{p}(t+\delta t) + c_3 \Delta\mathbf{a}(t+\delta t)
\end{aligned}$$

- The constants c_i depend on how many derivatives of $\mathbf{r}$ we include and the degree of the equation, etc.

- The correction can also be iterated

- However, doing this is almost never a good idea in MD: calculating the forces, especially in modern schemes, takes up the vast majority of the simulation time, so it is best to use an equation which requires only one evaluation of the force per time step (one correction)

- If the correction is not iterated, an obvious choice is $c_2 = 1$ (why?).

- Thus we reach the following approach to solving the MD equations of motion:

(a) predict $\mathbf{r}$, $\mathbf{v}$ and $\mathbf{a}$ for the time $t + \delta t$ using the present values of the same variables

(b) calculate forces and hence $\mathbf{a} = \mathbf{f}/m$ from the new $\mathbf{r}$

(c) correct the predicted $\mathbf{r}$, $\mathbf{v}$ and $\mathbf{a}$ etc. using the new $\mathbf{a}$

- Requirements for a good MD algorithm

(a)	fast	(not that important)
(b)	takes little memory	(important)
(b)	allows a long time step δt	(important)
(c)	reproduces the correct path	(see below)
(d)	conserves energy (and is reversible: $\delta t \to -\delta t \Rightarrow$ back to original state)	(very important)
(f)	easy to implement	(not that important)
(g)	only one force evaluation/time step	(important for complex V)

- fulfilling (c) completely is not possible: any small deviation somewhere will grow exponentially with time. Since all computers have limited floating-point precision, a small round-off error will eventually grow to a large difference

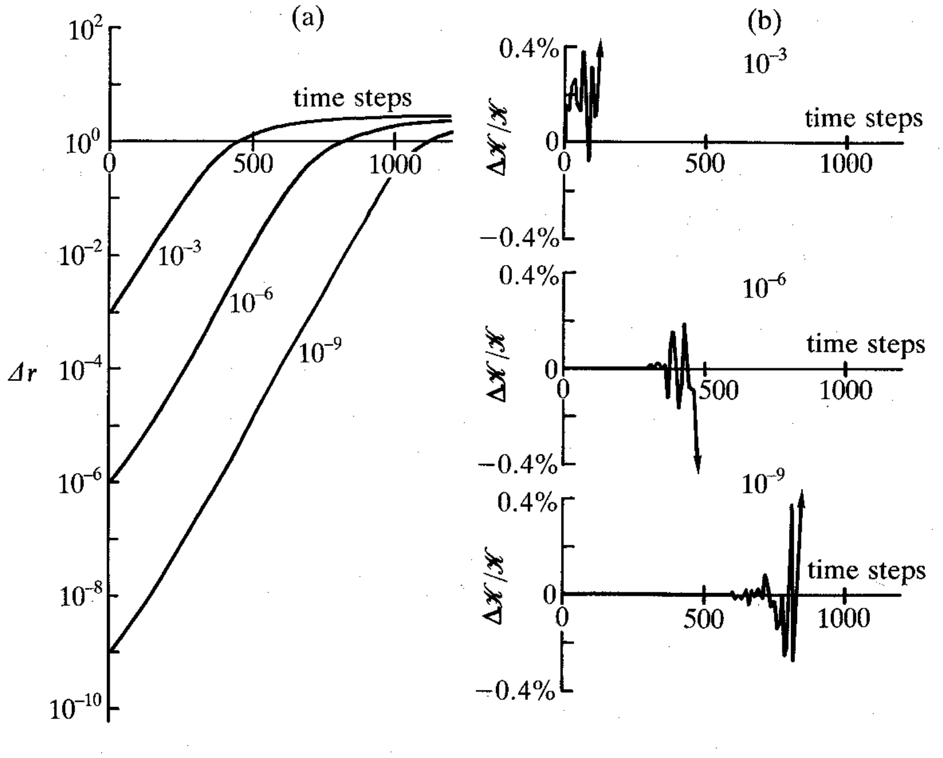


Fig. 3.1 The divergence of trajectories in molecular dynamics. Atoms interacting through the potential $v^{\mathrm{RLJ}}(r)$, eqn (1.10a), were used, and a dense fluid state was simulated ($\rho^* = 0.6$, $T^* = 1.05$, $\delta t^* = 0.005$). The curves are labelled with the initial displacement in units of σ. (a) Δr is the phase space separation between perturbed and reference trajectories. (b) $\Delta\mathcal{K}/\mathcal{K}$ is the percentage difference in kinetic energies.

- A reversible algorithm (see note (d)) has in principle no drift in energy, except for that induced by numerical inaccuracies

- In the following we present some of the most common MD algorithms:
- Verlet

  - Verlet is derived from the following two Taylor series:

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \delta t \mathbf{v}(t) + \frac{1}{2}\delta t^2 \mathbf{a}(t) + \ldots$$

$$\mathbf{r}(t-\delta t) = \mathbf{r}(t) - \delta t \mathbf{v}(t) + \frac{1}{2}\delta t^2 \mathbf{a}(t) + \ldots$$

  sum 'em up and rearrange:

$$\mathbf{r}(t+\delta t) + \mathbf{r}(t-\delta t) = 2\mathbf{r}(t) + \delta t^2 \mathbf{a}(t)$$

$$\mathbf{r}(t+\delta t) = 2\mathbf{r}(t) - \mathbf{r}(t-\delta t) + \delta t^2 \mathbf{a}(t)$$

  - So we have an algorithm which essentially does:
  $\{\mathbf{r}(t), \mathbf{a}(t), \mathbf{r}(t-\delta t)\} \rightarrow \{\mathbf{r}(t+\delta t), \mathbf{a}(t+\delta t)\}$

  - However, the velocities are missing; these can be calculated from

$$\mathbf{v}(t) = \frac{\mathbf{r}(t+\delta t) - \mathbf{r}(t-\delta t)}{2\delta t}$$

  - The error per iteration $O(\delta t^4)$; in the velocities $O(\delta t^2)$

  - memory requirement: $9N$

  - numerical problems, fluctuates heavily

  - Note that we get **r** and **v** at different t

- leap-frog
  - $\left\{\mathbf{r}(t), \mathbf{a}(t), \mathbf{v}\left(t-\frac{1}{2}\delta t\right)\right\} \to \left\{\mathbf{r}(t+\delta t), \mathbf{a}(t+\delta t), \mathbf{v}\left(t+\frac{1}{2}\delta t\right)\right\}$:

$$\mathbf{v}\left(t+\frac{1}{2}\delta t\right) = \mathbf{v}\left(t-\frac{1}{2}\delta t\right) + \delta t\mathbf{a}(t)$$

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \delta t\mathbf{v}\left(t+\frac{1}{2}\delta t\right)$$

  - velocity $\mathbf{v}(t) = \frac{1}{2}\left[\mathbf{v}\left(t-\frac{1}{2}\delta t\right) + \mathbf{v}\left(t+\frac{1}{2}\delta t\right)\right]$ for energies etc.
  - equivalent with Verlet; advantage explicit $\mathbf{v}$
  - memory requirement $9N$
  - but still velocities at different time than the positions
- velocity Verlet:
  - $\{\mathbf{r}(t), \mathbf{v}(t), \mathbf{a}(t)\} \to \{\mathbf{r}(t+\delta t), \mathbf{v}(t+\delta t), \mathbf{a}(t+\delta t)\}$:

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \delta t\mathbf{v}(t) + \frac{1}{2}\delta t^2\mathbf{a}(t)$$

$$\mathbf{v}(t+\delta t) = \mathbf{v}(t) + \frac{1}{2}\delta t[\mathbf{a}(t) + \mathbf{a}(t+\delta t)]$$

  - If we would eliminate $\mathbf{v}$ we'd get back to normal Verlet

- This can also be considered to be a simple predictor-corrector-algorithm: (same as three stage Gear with $\mathbf{r}$ correction = 0):

1. **Predictor stage:**

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \delta t \mathbf{v}(t) + \frac{1}{2}\delta t^2 \mathbf{a}(t)$$

$$\mathbf{v}^p\left(t+\frac{1}{2}\delta t\right) = \mathbf{v}(t) + \frac{1}{2}\delta t \mathbf{a}(t)$$

2. **Corrector stage:**

$$\mathbf{v}^c(t+\delta t) = \mathbf{v}^p\left(t+\frac{1}{2}\delta t\right) + \frac{1}{2}\delta t \mathbf{a}(t+\delta t)$$

- memory requirement $9N$

- Schematic illustration of the progress of different Verlet algorithms:

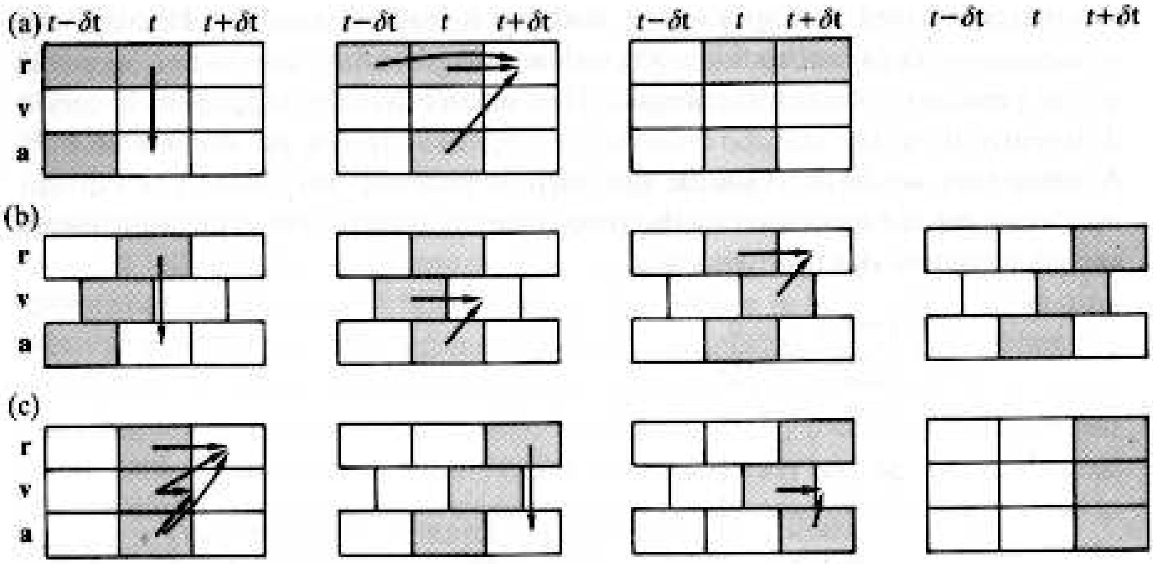


Fig. 3.2 Various forms of the Verlet algorithm. (a) Verlet's original method. (b) The leap-frog form. (c) The velocity form. We show successive steps in the implementation of each algorithm. In each case, the stored variables are in grey boxes.

- Velocity Verlet is a very popular algorithm because it is simple, reversible, yet reasonably accurate.

- Velocity Verlet as pseudocode:

```
do i=1,N
      x(i)=x(i)+deltat*vx(i)+0.5*deltat**2*ax(i)
      vx(i)=vx(i)+0.5*deltat*ax(i)
      ((and same for y and z))
enddo

((get new forces F and hence accelerations ax(i)))

do i=1,N
      vx(i)=vx(i)+0.5*deltat*ax(i)
      ((and same for y and z))
enddo
```

- Beeman algorithm (D. Beeman, J. Comp. Phys. **20** (1976) 130)

  - $\{\mathbf{r}(t), \mathbf{v}(t), \mathbf{a}(t), \mathbf{a}(t-\delta t)\} \rightarrow \{\mathbf{r}(t+\delta t), \mathbf{v}(t+\delta t), \mathbf{a}(t+\delta t)\}$:

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \delta t \mathbf{v}(t) + \frac{2}{3}\delta t^2 \mathbf{a}(t) - \frac{1}{6}\delta t^2 \mathbf{a}(t-\delta t)$$

$$\mathbf{v}(t+\delta t) = \mathbf{v}(t) + \frac{1}{3}\delta t \mathbf{a}(t+\delta t) + \frac{5}{6}\delta t \mathbf{a}(t) - \frac{1}{6}\delta t^2 \mathbf{a}(t-\delta t)$$

  - equivalent with Verlet if $\mathbf{v}$ eliminated, but velocity more accurate

  - memory requirement $12N$

- ion irradiation physics:

  - initially $E_{\max} \sim 1 - 100$ keV ;
  - in the end $E_{\max} \sim k_B T$

    $\Rightarrow$ it is worth using a variable time step (cf. lecture 2).

  - let us mark $\mathbf{r}_n = \mathbf{r}(t_n)$; $\mathbf{r}_{n+1} = \mathbf{r}(t_n + \delta t)$

  - Smith & Harrison (Computers in Physics **3** (1989) 68):

  - $\{\mathbf{r}_n, \mathbf{v}_n, \mathbf{a}_n, \mathbf{a}_{n-1}\} \rightarrow \{\mathbf{r}_{n+1}, \mathbf{v}_{n+1}, \mathbf{a}_{n+1}\}$:

    Taylor : $\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_n \delta t_n + \mathbf{a}_n \frac{\delta t_n^2}{2} + \mathbf{a}'_n \frac{\delta t_n^3}{6} + O(\delta t_n^4)$

    estimate $\mathbf{a}'_n = \frac{\mathbf{a}_n - \mathbf{a}_{n-1}}{\delta t_{n-1}} + O(\delta t_{n-1})$

    $\Rightarrow$ predictor for positions:

    $$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_n \delta t_n + [(3+R)\mathbf{a}_n - R\mathbf{a}_{n-1}]\frac{\delta t_n^2}{6} \quad (*)$$

    $$R = \frac{\delta t_n}{\delta t_{n-1}}$$

    velocity:

    $$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{a}_n \delta t_n + \mathbf{a}'_n \frac{\delta t_n^2}{2} + \mathbf{a}''_n \frac{\delta t_n^3}{6} + O(\delta t_n^4)$$

    force calculation from $\mathbf{r}_{n+1}$:

    $$\Rightarrow \quad \mathbf{a}'_n = \frac{\mathbf{a}_{n+1} - R^2 \mathbf{a}_{n-1} + (R^2 - 1)\mathbf{a}_n}{\delta t_n (1+R)}$$

    $$\mathbf{a}''_n = 2R\left[\frac{\mathbf{a}_{n+1} - R\mathbf{a}_{n-1} + (R+1)\mathbf{a}_n}{\delta t_n^2 (1+R)}\right]$$

    let's insert these into the Taylor series of $\mathbf{v}_{n+1}$:

$$\Rightarrow \mathbf{v}_{n+1} = \mathbf{v}_n + \left[\frac{(3+2R)\mathbf{a}_{n+1}}{1+R} + (3+R)\mathbf{a}_n - \frac{R^2\mathbf{a}_{n-1}}{1+R}\right]\frac{\delta t_n}{6} \quad (**)$$

- algorithm:
  - (a) calculate new positions $\mathbf{r}_{n+1}$ using equation (*)
  - (b) calculate new accelerations $\mathbf{a}_{n+1}$
  - (c) calculate velocities using equation (**)

[(d) correct the positions using

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_n\delta t_n + \left[\frac{(2+R)\mathbf{a}_{n+1}}{1+R} + (4+R)\mathbf{a}_n - \frac{R^2\mathbf{a}_{n-1}}{1+R}\right]\frac{\delta t_n^{\,2}}{12}$$

but this demands two force evaluations per time step]

- Step (d) can be left out

- memory $12N$

- error $O(\delta t_n^{\,4})$

- with a constant time step this reduces to the fairly simple form

$$\Rightarrow \quad \mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_n\delta t_n + [4\mathbf{a}_n - \mathbf{a}_{n-1}]\frac{\delta t_n^{\,2}}{6}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + [5\mathbf{a}_{n+1} + 8\mathbf{a}_n - \mathbf{a}_{n-1}]\frac{\delta t_n}{12}$$

- Finally, we present (without deriving it) one of the Gear algorithms, the so called six-value (fifth-order predictor) Gear algorithm. This is quite often used in MD [G. W. Gear, *Numerical initial value problems in ordinary differential equations,* (Prentice-Hall, Englewook Cliffs, NJ, USA) 1971; Allen-Tildesley].

- Using the notation: $\mathbf{r}_i = \dfrac{\mathbf{r}^{(i)}(\delta t)^i}{i!}$, where $\mathbf{r}^{(i)} = \dfrac{\partial^i}{\partial \mathbf{r}^i}\mathbf{r}$

  we get the **predictor**:

$$\begin{bmatrix} \mathbf{r}_0^{\mathrm{P}}(t+\delta t) \\ \mathbf{r}_1^{\mathrm{P}}(t+\delta t) \\ \mathbf{r}_2^{\mathrm{P}}(t+\delta t) \\ \mathbf{r}_3^{\mathrm{P}}(t+\delta t) \\ \mathbf{r}_4^{\mathrm{P}}(t+\delta t) \\ \mathbf{r}_5^{\mathrm{P}}(t+\delta t) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 3 & 6 & 10 \\ 0 & 0 & 0 & 1 & 4 & 10 \\ 0 & 0 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_0(t) \\ \mathbf{r}_1(t) \\ \mathbf{r}_2(t) \\ \mathbf{r}_3(t) \\ \mathbf{r}_4(t) \\ \mathbf{r}_5(t) \end{bmatrix}$$

  Note that the triangle is simply a Pascal's triangle matrix.

  For 2. order (Newtonian) equations of motion, error term $\delta\mathbf{R}_2 = \mathbf{a} - \mathbf{a}^{\mathrm{p}}$

  corrector: $\mathbf{r}_n^{\mathrm{c}} = \mathbf{r}_n^{\mathrm{P}} + \alpha\delta\mathbf{R}_2$

$$\alpha = \begin{bmatrix} 3/16 \\ 251/360 \\ 1 \\ 11/18 \\ 1/6 \\ 1/60 \end{bmatrix}$$

  Note that if the forces may depend on the velocities, we should have $\alpha_0 = 3/20$ instead.

- The Gear algorithms are not reversible, so theyalways does have an energy drift
- But the drift can be made extremely small for small values of δt

- The fluctuations in energy of different algorithms as a function of the time step is illustrated in the following:

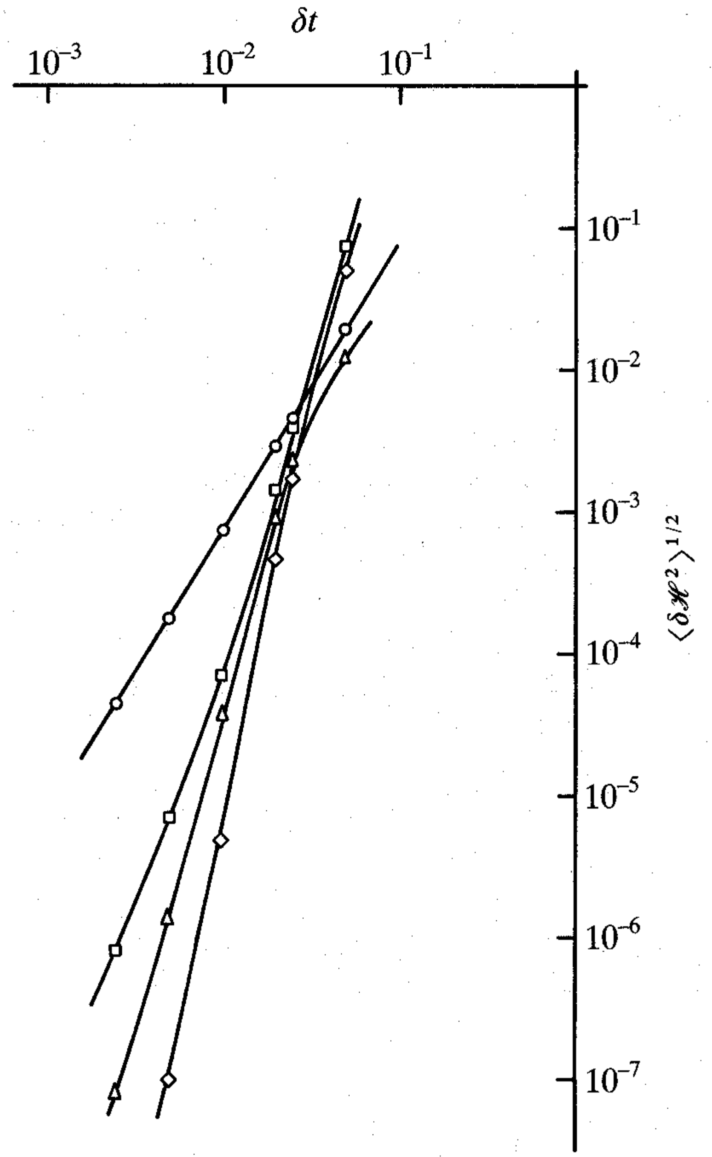
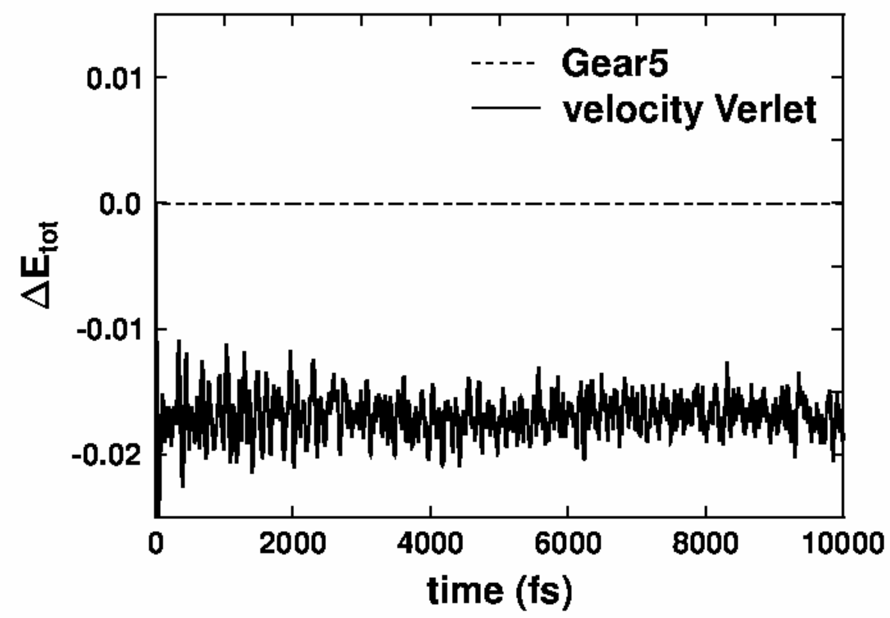


Fig. 3.3 Energy conservation of various algorithms. The system studied is as for Fig. 3.1. We calculate RMS energy fluctuations $\langle \delta \mathcal{H}^2 \rangle^{1/2}$ for various runs starting from the same initial conditions, and proceeding for the same total simulation time t_{run}, but using different time steps δt and corresponding numbers of steps $\tau_{run} = t_{run}/\delta t$. The plot uses log-log scales. The curves correspond to velocity Verlet (circles), Gear fourth-order (squares), Gear fifth-order (triangles), and Gear sixth-order (diamonds) algorithms.

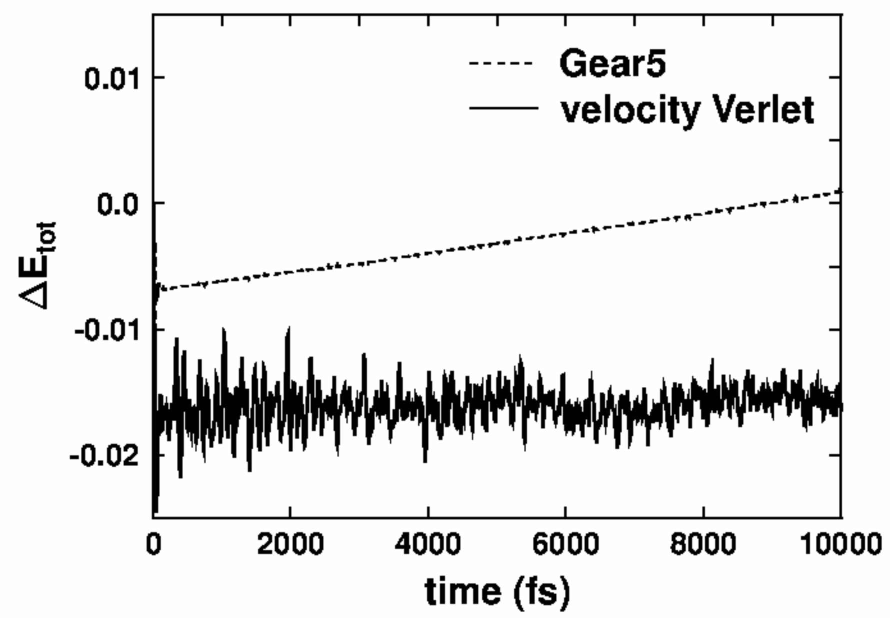
- So the 'better' algorithms have much less fluctuations for very short timesteps.

- Another illustration of this: a 10 ps simulation of a 500 atom Cu lattice at 300 K

  - Gear5 vs. velocity Verlet:
    Time step 1 fs



Time step 10 fs:



- So in Verlet much larger fluctuations, but less drift at 10 ps

- Tuckerman, Berne and Martyna have recently developed new reversible MD-algorithms using a Trotter factorisation of Liouville propagators

  - The method is theoretically very well motivated, and it can be used to derive e.g. the Verlet algorithms [Tuckerman *et al.*, J. Chem. Phys. **97** (1992) 1990]

  - It can also be used to derive a predictor-corrector-type algorithm which is comparable to Gear4 in accuracy but is also time reversible [Martyna and Tuckerman, J. Chem. Phys. **102** (1995) 8071]

- So, what algorithm should one use? My personal feeling is:

  - A quick solution which works well both for short and long time steps: velocity Verlet

  - If one wants minimal oscillations in the total energy: Gear5

  - If one wants great accuracy and minimal energy drift, it is worth looking into Tuckerman's method.