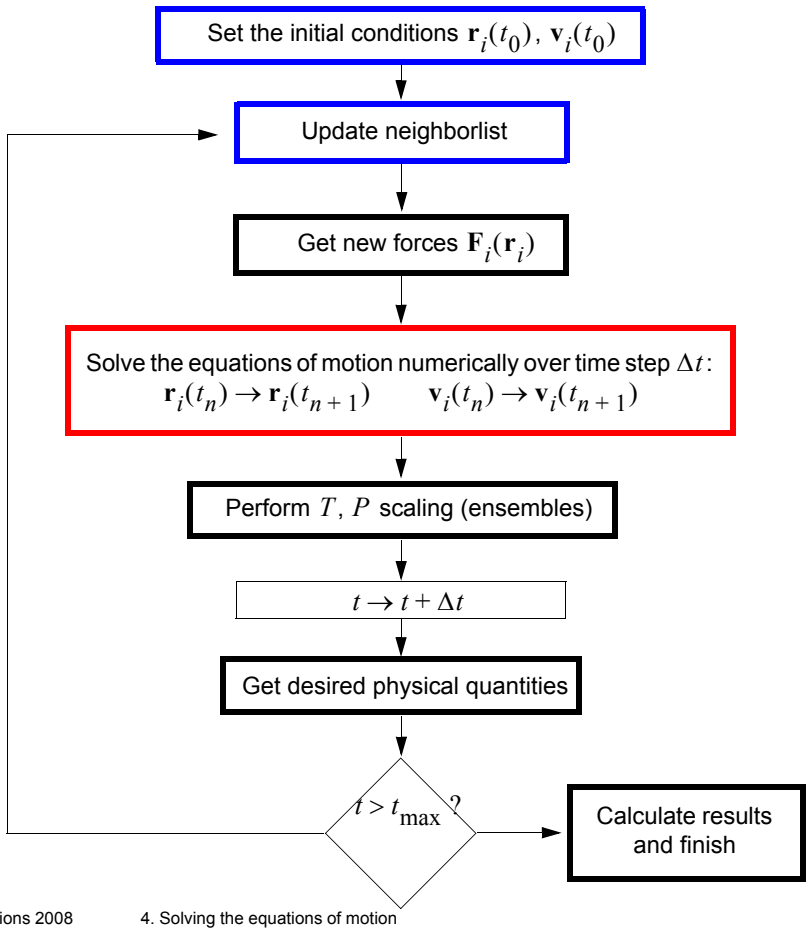


Solving the equations of motion

[Main source: 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 V
- Lagrange equations of motion:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0;$$

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

$\mathbf{q}$ = generalized coordinate

- By using the cartesian coordinates

$$q_i = r_i$$

$$K(\dot{\mathbf{r}}) = \sum_i \frac{1}{2} m_i \dot{r}_i^2,$$

$$V = V(\mathbf{r}),$$

we get the familiar (Newtonian) form

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

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

Solving the equations of motion

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

$$\dot{q}_i = \frac{\partial H}{\partial p_i}, \quad \dot{p}_i = -\frac{\partial H}{\partial q_i},$$

where $p_i = \frac{\partial L}{\partial \dot{q}_i}$ is the generalized momentum

and $H(\mathbf{q}, \mathbf{p}) = \sum_i \dot{q}_i p_i - L(\mathbf{q}, \dot{\mathbf{q}})$ the Hamiltonian function (we assume that $\dot{q}_i$ can be given as a function of $\mathbf{p}$)

- If V does not depend on the velocities, we get quickly back to the familiar form

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

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

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

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

- So we have two alternatives:
  1. Solve a system of $3N$ 2nd order ODE's ($m_i \ddot{\mathbf{r}}_i = \mathbf{f}_i$) derived from the Lagrangian or Newtonian formalism
  2. Solve a system of $6N$ 1st order ODE's derived from the Hamiltonian formalism

Numerical solution of equations of motion

- Finite difference method: from a system configuration (atom positions, velocities etc.) at time t we calculate the configuration at time $t + \delta t$
  - δt can be constant or variable
  - initial conditions $\mathbf{r}(0)$, $\mathbf{v}(0)$ have to be known (initial value problem)

- 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:

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

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

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

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

Equations of motion not (yet) used.

  - $\mathbf{v}$, $\mathbf{a}$ and $\mathbf{b}$ are higher time derivatives of $\mathbf{r}$:
    $\mathbf{v}$ = velocity, $\mathbf{a}$ = acceleration and $\mathbf{b}$ = the time derivative of acceleration.

Numerical solution of equations of motion

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

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

    Equations of motion now used.

    $\Rightarrow$ 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

$$\mathbf{r}^{\mathrm{c}}(t+\delta t) = \mathbf{r}^{\mathrm{p}}(t+\delta t) + c_0\Delta\mathbf{a}(t+\delta t)$$

$$\mathbf{v}^{\mathrm{c}}(t+\delta t) = \mathbf{v}^{\mathrm{p}}(t+\delta t) + c_1\Delta\mathbf{a}(t+\delta t)$$

$$\mathbf{a}^{\mathrm{c}}(t+\delta t) = \mathbf{a}^{\mathrm{p}}(t+\delta t) + c_2\Delta\mathbf{a}(t+\delta t)$$

$$\mathbf{b}^{\mathrm{c}}(t+\delta t) = \mathbf{b}^{\mathrm{p}}(t+\delta t) + c_3\Delta\mathbf{a}(t+\delta t)$$

  - 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
  - But no in MD: calculating the forces expensive $\Rightarrow$ use an algorithm requiring only *one evaluation of the force* per time step (one correction)
  - If the correction is not iterated, an obvious choice is $c_2 = 1$.

Numerical solution of equations of motion

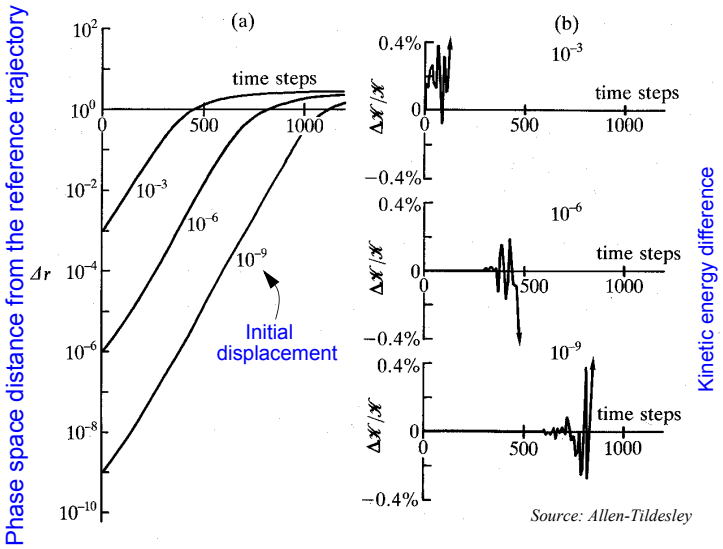
- Thus we reach the following approach to solving the MD equations of motion:
  (a) predict $\mathbf{r}$, $\mathbf{v}$ and $\mathbf{a}$ for the time $t+\delta t$ using the present values of the same variables
  (b) calculate forces and hence $\mathbf{a} = \mathbf{f}/m$ from the new $\mathbf{r}$
  (c) 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: $\delta t \rightarrow -\delta t \Rightarrow$ back to original state)	(very important)
(f) easy to implement	(not that important)
(g) only one force evaluation/time step	(important for complex V)

Numerical solution of equations of motion

- 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 (Lennard-Jones system; in reduced units $\rho^* = 0.6$, $T^* = 1.05$):



- A reversible algorithm has in principle no drift in energy, except for that induced by numerical inaccuracies.

Common algorithms

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

- **Verlet**
  - 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 them up and rearrange:

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

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

  - 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

Common algorithms

- **Leap-frog**
  - Mathematically equivalent with Verlet (not numerically)

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

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

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

  - 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.

  - Advantage: explicit $\mathbf{v}$.
  - Memory requirement $9N$.
  - But still velocities at different time than the positions.

Common algorithms

- **Velocity Verlet**
  - Eliminates the half-step velocity problem

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

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

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

  - If we would eliminate $\mathbf{v}$ we would 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:

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

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

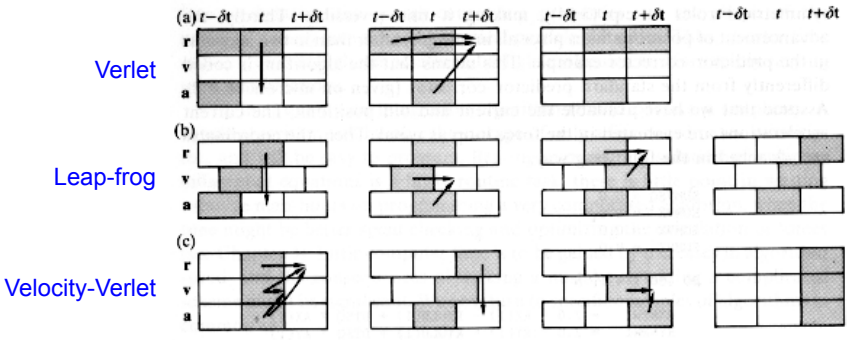
  2. Corrector stage:

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

  - Memory requirement $9N$.

Common algorithms

- Schematic illustration of the progress of different Verlet algorithms:



Source: Allen-Tildesley

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

Common algorithms

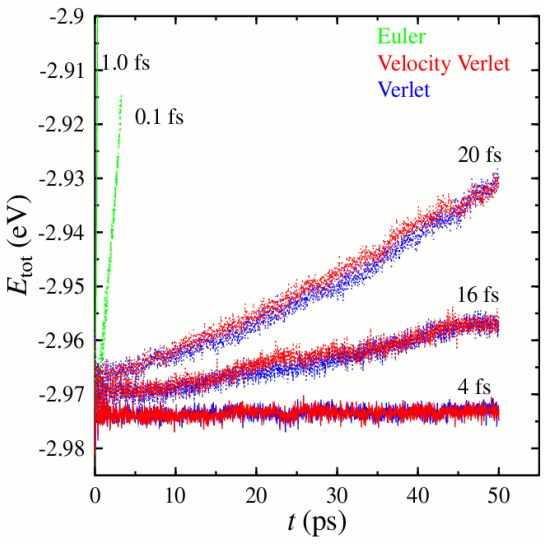
- Velocity Verlet as pseudocode:

```
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 accelerations ax(i)))

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

- Comparison of performance
  - 500 Cu atoms at 300 K
  - Euler: $\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \delta t \mathbf{v}(t)$
    $\mathbf{v}(t+\delta t) = \mathbf{v}(t) + \delta t \mathbf{a}(t)$



Common algorithms

- Beeman algorithm (D. Beeman, *J. Comp. Phys.* 20 (1976) 130.)
  - Equivalent with Verlet if $\mathbf{v}$ eliminated, but velocity more accurate

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

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

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

  - Memory requirement $12N$

Common algorithms

- **Ion irradiation physics**
  - Initially $E_{\max} \sim 1 - 100$ keV ;
  - In the end $E_{\max} \sim k_B T \Rightarrow$ variable time step
  - 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 $\mathbf{a}'_n = \frac{\mathbf{a}_n - \mathbf{a}_{n-1}}{\delta t_{n-1}} + O(\delta t_{n-1})$

Time step ratio

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

$\Rightarrow$ Predictor for positions:

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

Velocity:

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

Common algorithms

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

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

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

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

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

- Algorithm:
  (a) calculate new positions $\mathbf{r}_{n+1}$ using equation (1)
  (b) calculate new accelerations $\mathbf{a}_{n+1}$
  (c) calculate velocities using equation (2)

[(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]

- Memory $12N$, error $O(\delta t_n{}^4)$.
- With a constant time step this reduces to the fairly simple form.

$$\Rightarrow \quad \mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_n\delta t_n + [4\mathbf{a}_n - \mathbf{a}_{n-1}]\frac{\delta t_n{}^2}{6}, \qquad \mathbf{v}_{n+1} = \mathbf{v}_n + [5\mathbf{a}_{n+1} + 8\mathbf{a}_n - \mathbf{a}_{n-1}]\frac{\delta t_n}{12}$$

Common algorithms

- Six-value (fifth-order predictor) Gear algorithm (Gear5). This is quite often used in MD[1].
  - Using the notation: $\mathbf{r}_i = \frac{\mathbf{r}^{(i)}(\delta t)^i}{i!}$, where $\mathbf{r}^{(i)} = \frac{\partial^i}{\partial t^i}\mathbf{r}$

we get the **predictor** $\mathbf{r}_i^{\mathrm{p}}$:

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

- Note that the triangle is simply a Pascal's triangle matrix.
- For 2nd order (Newtonian) equations of motion, error term is $\delta\mathbf{r}_2 = \mathbf{r}_2 - \mathbf{r}_2^{\mathrm{p}}$.

1. G. W. Gear, *Numerical initial value problems in ordinary differential equations,* (Prentice-Hall, Englewook Cliffs, NJ, USA) 1971; Allen-Tildesley

Common algorithms

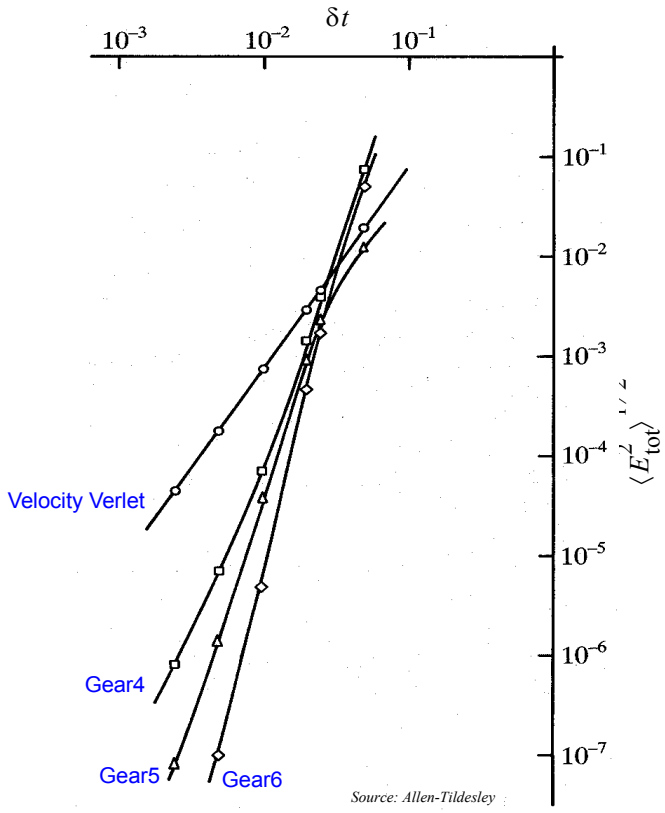
- Corrector: $\mathbf{r}_n^{\mathrm{c}} = \mathbf{r}_n^{\mathrm{P}} + \alpha\delta\mathbf{r}_2,\ \alpha = \begin{bmatrix} 3/16 \\ 251/360 \\ 1 \\ 11/18 \\ 1/6 \\ 1/60 \end{bmatrix} = \begin{bmatrix} 0.1875 \\ 0.6972 \\ 1.0000 \\ 0.6111 \\ 0.1667 \\ 0.0167 \end{bmatrix}$

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

Common algorithms

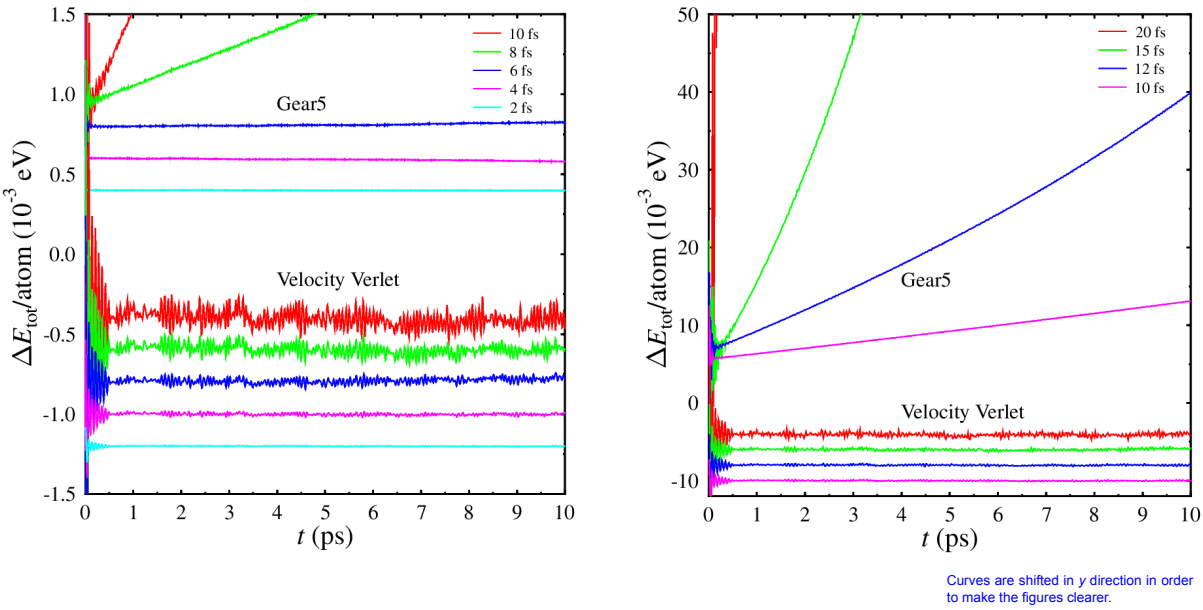
- The fluctuations in energy of different algorithms as a function of the time step is illustrated on the right (Lennard-Jones system; in reduced units $\rho^* = 0.6,\ T^* = 1.05$)

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



Common algorithms

- Another illustration of this: a 10 ps simulation of a 4000 atom Cu lattice at 300 K. Potential = EAM



Curves are shifted in *y* direction in order to make the figures clearer.

Newer algorithms

- 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.]
  - New algorithms $\Rightarrow$ not much experience.
- So, what algorithm should one use?
  - A quick solution which works well with short 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**.