Numerical solution of differential equations

Deriving and solving differential equations (DE) is a common task in computational research.

➢ Many physical laws/relations are formulated in terms of DE.

➢ Most continuum simulation methods are based on solution of DE.

Although the finite difference and the finite element methods typically used to solve DE are often intuitively associated with large-scale macroscopic problems, this association is inadequate. The methods are not calibrated to any physical time or length scale.

Numerical methods for solving DE is a vast and complex subject area. In this lecture we will only scratch the surface by briefly discussing several methods used for solving a system of ordinary DE relevant to Particle Dynamics methods.

Particle Dynamics

Particle Dynamics models: Physical system is represented as a sets of particles rather than densities (fields) evolving over time.

Examples:

- Motion of astronomical bodies under the action of gravitational forces

- Motion of atoms under the action of interatomic forces in Molecular Dynamics (MD) method

- Coarse-grained description of the dynamics of fluids and polymeric systems by Dissipative Particle Dynamics method

- Smoothed Particle hydrodynamics, Particle-in-Cell and other quasi-particle methods as alternatives to continuum approaches
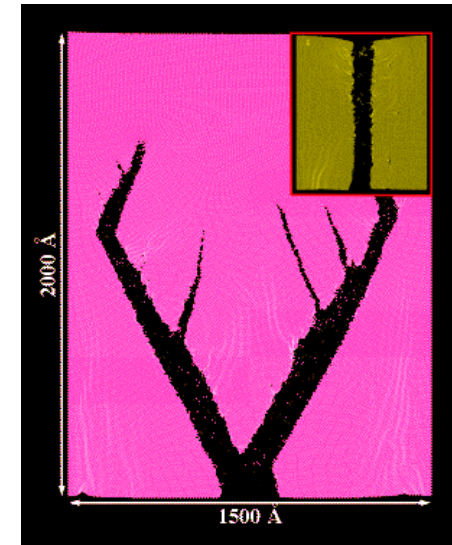
MD simulation of crack propagation
P. Vashista et al.

Dissipative Particle Dynamics simulation of Rayleigh-Taylor instability
Dzwinel, W., Alda, W., Yuen, D.A.
Molecular Simulation, **22**, 1999, 397-418.

Particle Dynamics – Ordinary Differential Equations (ODE)

In Particle Dynamics models laws of motion are typically ordinary differential equations (ordinary means that unknown functions depend only on one variable, typically time).

Consider system of N particles. Trajectory $\vec{r}_i(t)$ of a particle i of mass of m_i, under the action of force $\vec{F}_i$ can be described by the Newton's equation of motion - the second order differential equations (the "order" of a DE is the highest order of any of the derivatives in the equation):

$$m_i \frac{d^2\vec{r}_i(t)}{dt^2} = \vec{F}_i(\vec{r}_1, \vec{r}_2, ..., \vec{r}_N, t) \qquad \text{in 3D space}$$

$$\vec{r}_i = \{x_i, y_i, z_i\}, \quad \vec{F}_i = \{F_i^x, F_i^y, F_i^z\}, \quad i = 1, 2, ..., N$$

3N equations: $\quad m_i \dfrac{d^2 x_i(t)}{dt^2} = F_i^x \qquad m_i \dfrac{d^2 y_i(t)}{dt^2} = F_i^y \qquad m_i \dfrac{d^2 z_i(t)}{dt^2} = F_i^z$

these 3N second order ODE can be rewritten as a system of 6N first order equations:

$$v_i^x = \frac{dx_i(t)}{dt}, \quad v_i^y = \frac{dy_i(t)}{dt}, \quad v_i^z = \frac{dz_i(t)}{dt}, \quad \frac{dv_i^x(t)}{dt} = \frac{F_i^x(t)}{m_i}, \quad \frac{dv_i^y(t)}{dt} = \frac{F_i^y(t)}{m_i}, \quad \frac{dv_i^z(t)}{dt} = \frac{F_i^z(t)}{m_i}$$

Therefore, in general, our task is to solve a system of equations $du/dt = f(u,t)$ with initial conditions given by $u(t_0) = u_0$

Finite difference methods for Particle Dynamics problem

Thus, we need to solve a system of first order ODEs. Analytical solution is difficult and often impossible for a system of more than 2 interacting particles – force acting on a particle depends on positions of all other particles and integration of the equation would involve integrating over a sum. Therefore we have to use a numerical *approximation* for the integral over time.
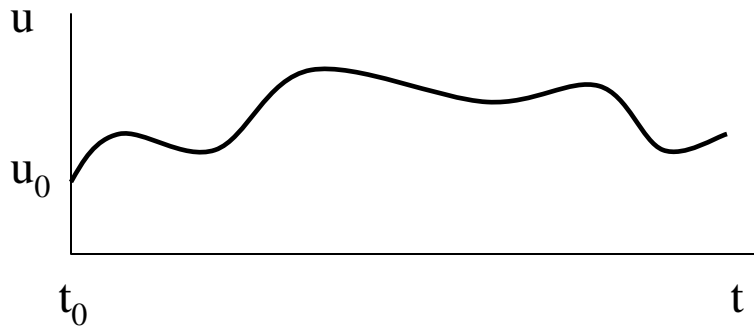
Let's consider, for simplicity only one equation $du/dt = f(u,t)$, $u(t_0) = u_0$
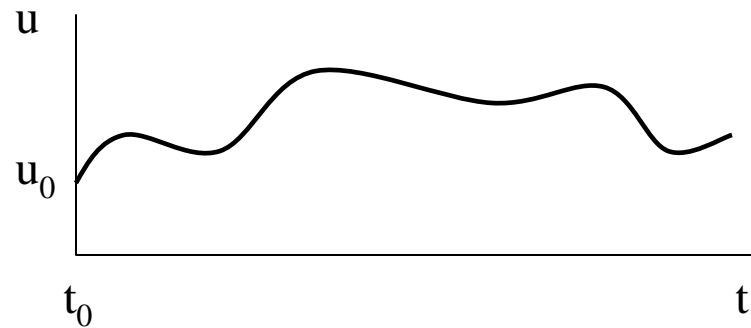
This type of problem is called *initial value problem*, in contrast to *boundary value problem*, in which values of u are given at two end points.

Finite difference methods for Particle Dynamics problem

Thus, we want to solve the following first order ODE equation $du/dt = f(u,t)$, $u(t_0) = u_0$

Finite difference technique is based on two approximations:

1. Discretization of time: a grid of timesteps is introduced.

2. Replace the differential equation by the corresponding finite difference equation:

$$dt \rightarrow h$$
$$du \rightarrow \Delta u = u(t+h) - u(t)$$

$$\frac{du}{dt} = \lim_{h \to 0} \frac{u(t + h) - u(t)}{h} \approx \frac{\Delta u}{h}\bigg|_{h \neq 0}$$

Finite difference method allows us, starting from the initial value u_0, to calculate values of $u(t_0 + n \times h)$ one step at a time.
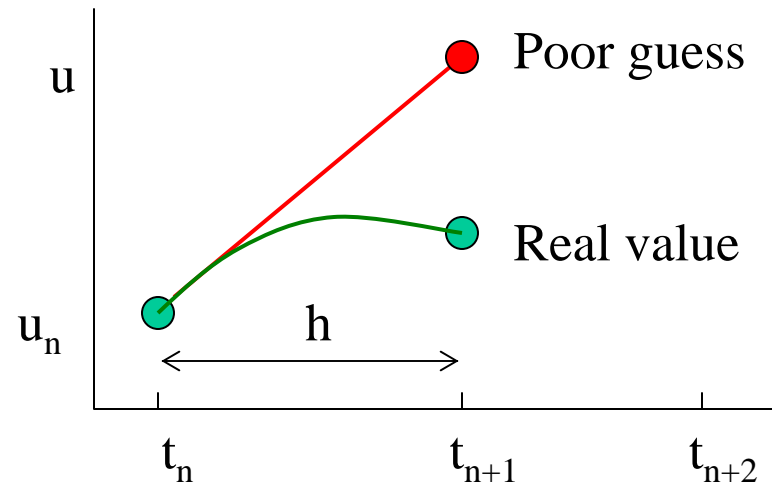
Euler's Method

Inaccurate and can be unstable: should not be used!

For $t_n = t_0 + n \times h, \ 0 \leq n \leq (t_{last} - t_0)/h$ an approximate solution is given by

$$u_{n+1} = u_n + h*f(u_n, t_n)$$

Linear approximation is used to get the next point

The main problem with Euler method is that only information from the beginning of the interval is used to extrapolate the value at the other side of the interval. In other words, **only slope of the function is taken into account, the curvature is ignored**.

Error in Euler's method

Let's use Taylor expansion to get the exact solution:

$$u_{n+1} = u_n + h\frac{du_n(t)}{dt} + \frac{h^2}{2!}\frac{d^2u_n(t)}{dt^2} + \ldots + \frac{h^p}{p!}\frac{d^pu_n(t)}{dt^p} + \ldots$$

Assuming that u_n is exact, the estimation of local error due to the truncation is

$$u_{n+1} = u_n + hf(u_n, t) + O(h^2)$$

This is a local error. As it accumulates over the h^{-1} intervals the global error is $O(h)$. **Thus, this is a first-order method.**

That is, the truncation error is linearly proportional to the time interval h.
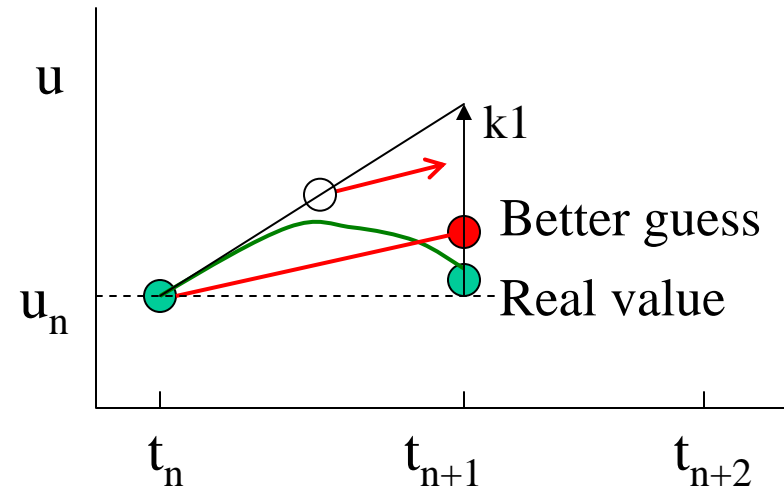
Modified Euler's method or second-order Runge-Kutta method

The accuracy of the approximation can be improved by evaluating the function f at two points (sometimes called the midpoint method):

$$k1 = h* f(u_n, t_n)$$

$$k2 = h* f(u_n+k1/2, t_n+h/2)$$

$$u_{n+1} = u_n + k2 + O(h^3)$$

This better extrapolation uses the fact that the average slope of the function is more closely approximated by the slope at the **center** of the interval (k2/h).

The method is inefficient and only moderately accurate but is used in calculations sometimes.
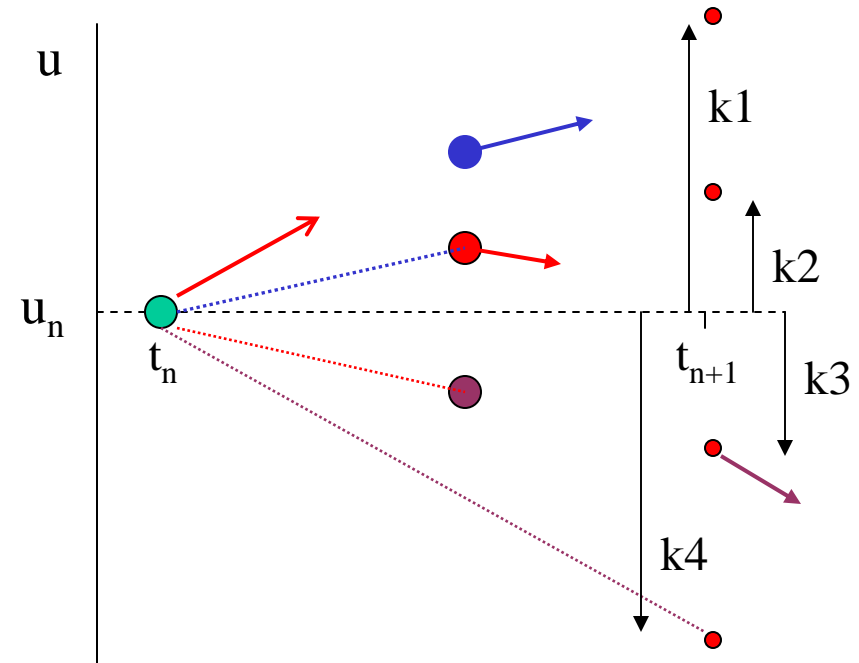
Fourth-order "classical" Runge-Kutta method

$k1 = h* f(u_n, t_n)$

$k2 = h* f(u_n+k1/2, t_n+h/2)$

$k3 = h* f(u_n+k2/2, t_n+h/2)$

$k4 = h* f(u_n+k3, t_n+h)$

$u_{n+1} = u_n + k1/6 + k2/3 + k3/3 + k4/6 + O(h^5)$

1. Use derivative at the beginning of the interval to get first trail midpoint
2. Use derivative at the first midpoint to get second trail midpoint
3. Use derivative at the second midpoint to get a trail endpoint
4. Integrate by Simpson's rule, using average of two midpoint estimates

Fourth-order "classical" Runge-Kutta method

The global "truncation" error of classical Runge-Kutta is $O(h^4)$ and it is achieved by computing function f four times at each timestep. As compared to the Euler's Method, The Runge-Kutta does 4 times as much calculations at each timestep but needs much smaller number of steps for the same time interval to achieve a given accuracy.

This is the most popular method for many systems involving few degrees of freedom. It achieves good accuracy with modest computational complexity.

Numerical Recipes book: "For many scientific users, fourth-order Runge-Kutta is not just the first word on ODE integration, but the last word as well. In fact, you can get pretty far on this old workhorse, especially if you combine it with an adaptive stepsize algorithm."

But… Runge-Kutta is almost never used in particle dynamics methods that involve many degrees of freedom.

The main reason for this – very high computational cost. The right-hand side of the equation has to be evaluated 4 times. Calculation of function f (force) is by far the most time consuming part of any particle method.

An additional minor problem is that the method lacks the time-reversal symmetry of the Newton's equations.

Special methods for particle dynamics

Second-order differential equations in which first-order derivatives do not appear are found so frequently in applied problems, particularly those arising from the law of motion, that special methods have been devised for their solution.

The idea is to go directly from the second derivatives to the function itself without having to use the first order derivatives.

$$m \, \frac{d^2 r(t)}{dt^2} = F(t)$$

Verlet algorithm for $m \dfrac{d^2 r(t)}{dt^2} = F(t)$

Let's write two third-order Taylor expansions for r(t) at t+h and t-h and sum them together

$$r(t+h) = r(t) + h \frac{dr(t)}{dt} + \frac{h^2}{2} \frac{d^2 r(t)}{dt^2} + \frac{h^3}{6} \frac{d^3 r(t)}{dt^3} + O(h^4)$$

$$r(t-h) = r(t) - h \frac{dr(t)}{dt} + \frac{h^2}{2} \frac{d^2 r(t)}{dt^2} - \frac{h^3}{6} \frac{d^3 r(t)}{dt^3} + O(h^4)$$

$$\Longrightarrow \quad r(t+h) + r(t-h) = 2r(t) + h^2 \frac{d^2 r(t)}{dt^2} + O(h^4)$$

$$\boxed{r(t+h) = -r(t-h) + 2r(t) + h^2 F(t)/m + O(h^4)}$$

Verlet algorithm

$$r(t+h) = -r(t-h) + 2r(t) + h^2 F(t)/m + O(h^4)$$

This algorithm is one of the most commonly used particle dynamics computational methods.

- It is simple to implement, accurate and stable (an error made at any given step tends to decay rather than magnify later on).

- It is time reversible (in practice, accumulation of computational roundoff errors eventually breaks reversibility of the calculation anyway – the reversibility of the method is not a major criterion in choosing the method).

Historic note: The algorithm was first used in 1791 by Delambre, and has been rediscovered many times since then, most recently by Verlet in 1960's for molecular dynamics. It was also used by Cowell and Crommelin in 1909 to compute the orbit of Halley's comet, and by Störmer in 1907 to study the motion of electrical particles in a magnetic field.

Verlet algorithm: A few small problems

$$r(t+h) = -r(t-h) + 2r(t) + h^2F(t)/m + O(h^4)$$

➢ The velocity does not enter this algorithm explicitly. We need velocities in MD simulations to compute kinetic energy so that we can check conservation of the total energy.

➢ The initial conditions are often specified as the initial coordinates and initial velocities, while the Verlet involves positions only. How to introduce initial temperature?

➢ One can use $v(t) = [r(t+h) - r(t-h)]/2h$ to compute the velocities, but the local error associated with this expression is of order h^2 rather than h^4.

Velocity Verlet algorithm

Another implementation of the Verlet algorithm that explicitly includes velocities is so-called Velocity Verlet Algorithm.

$$r(t+h) = r(t) + h\ v(t) + h^2\ F(t)/2\ m + O(h^4)$$

$$v(t+h) = v(t) + h\ (F(t)/m + F(t+h)/m)/2 + O(h^3)$$

➢ Self-starting from the positions and velocities at the initial time.

➢ Mathematically identical to the original Verlet algorithm.

➢ Do not need to store the values of r(t) and v(t) at two different times.

Note that you do not need start-up calculation of $r(t_0-h)$ that is needed in the original Verlet.

Note that you have to calculate new forces F(t+h) after calculation of new positions r(t+h) but before calculation of new velocities.

Predictor-corrector algorithms

Another commonly used class of methods to integrate equations of motion are predictor-corrector algorithms. These methods can be used in simulations with velocity-dependent forces, such as in constant-temperature MD methods that will be discussed later in this course.

➢ **Predictor.** From the positions and their time derivatives up to a certain order (all at time t) one "predicts" the same quantities at time t + h using the Taylor expansion.

➢ **Force calculation.** The force acting on a given particle is computed for the predicted positions. The acceleration $\mathbf{a} = \mathbf{F/m}$ will be in general different from the "predicted acceleration". The difference between the two constitutes the "error signal".

➢ **Corrector.** The error signal is used to correct the positions and their derivatives. All the corrections are proportional to the error signal. The coefficients of proportionality are "magic numbers" chosen to maximize the stability of the algorithm.

This method is used in many real MD simulations.

Nordsieck fifth-order predictor-corrector algorithm

(implemented in the MSE627-MD code)

Let us define the following variables: $\quad q_1(t) = h\dfrac{dr(t)}{dt} \qquad q_2(t) = \dfrac{h^2}{2}\dfrac{d^2 r(t)}{dt^2}$

$$q_3(t) = \frac{h^3}{6}\frac{d^3 r(t)}{dt^3} \qquad q_4(t) = \frac{h^4}{24}\frac{d^4 r(t)}{dt^4} \qquad q_5(t) = \frac{h^5}{120}\frac{d^5 r(t)}{dt^5}$$

1. Predicted values (Taylor expansions):

$$r(t+h) = r(t) + q_1(t) + q_2(t) + q_3(t) + q_4(t) + q_5(t)$$

$$q_1(t+h) = q_1(t) + 2q_2(t) + 3q_3(t) + 4q_4(t) + 5q_5(t)$$

$$q_2(t+h) = q_2(t) + 3q_3(t) + 6q_4(t) + 10q_5(t)$$

$$q_3(t+h) = q_3(t) + 4q_4(t) + 10q_5(t)$$

$$q_4(t+h) = q_4(t) + 5q_5(t)$$

$$q_5(t+h) = q_5(t)$$

Nordsieck fifth-order predictor-corrector algorithm

(implemented in the MSE627-MD code)

2. Error in acceleration:
$$\delta q_2(t+h) = \frac{F(t+h)}{2m}h^2 - q_2(t+h)$$

3. Correction (smoothing filter):

$$r^c(t+h) = r(t+h) + C_0 \delta q_2(t+h) \qquad C_0 = 3/20$$

$$q_1^c(t+h) = q_1(t+h) + C_1 \delta q_2(t+h) \qquad C_1 = 251/360$$

$$q_2^c(t+h) = q_2(t+h) + C_2 \delta q_2(t+h) \qquad C_2 = 1$$

$$q_3^c(t+h) = q_3(t+h) + C_3 \delta q_2(t+h) \qquad C_3 = 11/18$$

$$q_4^c(t+h) = q_4(t+h) + C_4 \delta q_2(t+h) \qquad C_4 = 1/6$$

$$q_5^c(t+h) = q_5(t+h) + C_5 \delta q_2(t+h) \qquad C_5 = 1/60$$

Reference: Allen & Tildesley, pp.74-75 and 340-341

Verlet vs. Predictor-Corrector for MD Simulations

Velocity Verlet and predictor-corrector are the most efficient integrators for MD.

Predictor-corrector methods:

➢ the main disadvantage is long-term energy drift (the energy error is increasing linearly with time).

➢ have very good local energy conservation (small fluctuations).

➢ can be used with velocity-dependent forces (e.g. with Nosé-Hoover method for constant temperature simulations).

➢ is easy to use with multiple timestep algorithms.

➢ is not time-reversible.

Verlet method

➢ have a much better long-term energy conservation as compared to predictor-corrector, increase of the timestep typically results in larger local energy fluctuations as compared to predictor-corrector, but not in energy drift.

➢ cannot be used when forces depend on the velocities of the atoms.

➢ is time reversible.

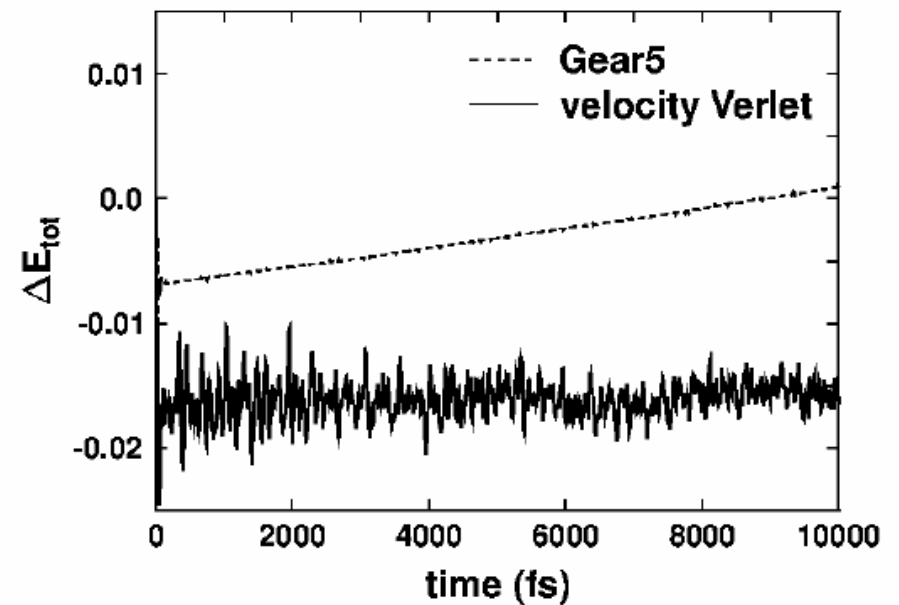# Verlet vs. Predictor-Corrector for MD Simulations

10 ps simulation of a 500 atom Cu lattice at 300 K (by Kai Nordlund)
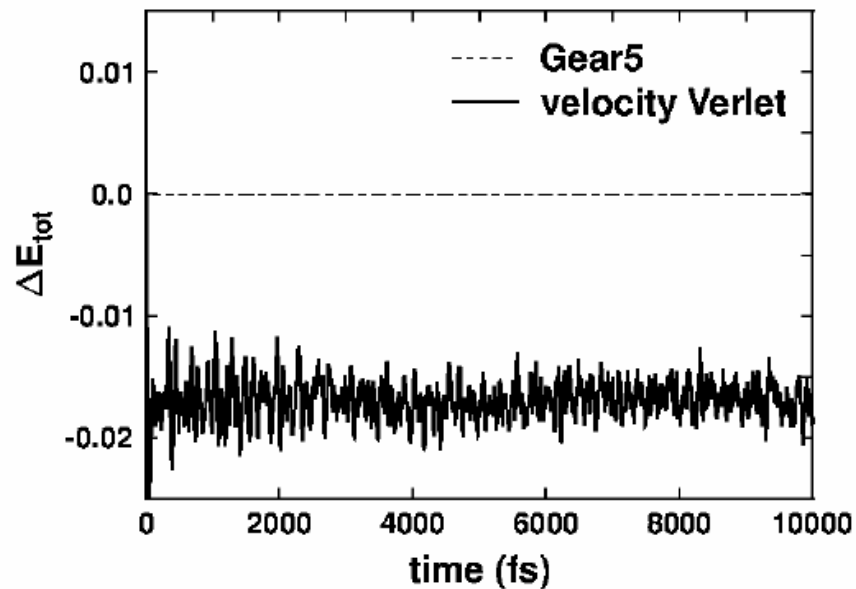
MSE 4270/6270 References for ODE

1. William H. Press, et al., Numerical Recipes in Fortran 90 (or C) : The art of scientific and parallel computing, Cambridge University Press, is available for free in ps/pdf formats from http://www.nr.com/

2. M. P. Allen, D. J. Tildesley, Computer simulation of liquids (Clarendon Press: Oxford, 1990), pages 73-84 and 340-342 – a good description of methods for solution of ordinary differential equations used in molecular dynamics (Verlet, predictor-corrector).

3. D. Frenkel, B. Smit, Understanding molecular simulation from algorithms to applications (Academic Press: San Diego, 1996), pages 59-67 - more on methods used in molecular dynamics.

4. Dierk Raabe, Computational materials science: the simulation of materials, microstructures and properties (Wiley-VCH: Weinheim, 1998), pages 29-40 – short introduction to differential equations.