LECTURE 2 28.3.2006

FINITE DIFFERENCE METHODS AND MD ALGORITHMS

EULER AND ODHO

The simplest method: Euler, time step h, advance positions and velocities from one instant of time t to t+h as

$$x(t+h) = x(t) + h\dot{x}(t)$$
$$v(t+h) = v(t) + h\dot{v}(t)$$

A simple model system: one-dimensional harmonic oscillator (ODHO)

$$V(x) = \frac{1}{2}kx^2 \;\; ; \;\; F(x) = -\frac{d}{dx}V(x) = -kx$$

Euler equations for ODHO

$$x(t+h) = x(t) + hv(t)$$
$$v(t+h) = v(t) - h\omega^2 x(t) \;\; ; \;\; \omega^2 = k/m$$

Define the initial state: {x(0),v(0)}, apply algorithm → an (approximate) discretized trajectory {x(t),v(t)} by increments of time by h

NUMERICAL ERRORS

A. Truncation error (algorithm)

Taylor series

$$x(t+h) = \sum_{n=0}^{\infty} \frac{1}{n!}\frac{d^n}{dx^n}x(t)$$

has to be truncated at some n → truncation error $O(h^{n+1})$, convention: algorithmic order = n (e.g. 1 for Euler)

B. Round-off error

Originates from the application of the algorithm in computer. It is affected by many factors such as the chosen computational accuracy (single, double precision), order of the statements in the code (← optimization of code by the compiler,…), approximations made by the computer in evaluating mathematical functions (e.g. trigonometric functions from their series),…

C. Local and global errors

Local error is generated during a single time step.
Global error is accumulated during the full length of the simulation.
It is easy to show (by using the mean-value theorem, Haile p. 152) that the global error is always of the order (n-1) if the local error is of the order (n).

ALGORITHMIC STABILITY AND STABILITY ANALYSIS

An *unstable* algorithm accumulates errors from single time steps such that after a certain period, the trajectory cannot be reliably calculated any more.

A *stable* algorithm does not accumulate errors.

Most MD algorithms are *conditionally stable*, i.e. stable for all $h < h_{critical}$

Example: take ODHO and Euler's algorithm

Let

$$x'(t) \text{ ja } x(t);\ v'(t) \text{ ja } v(t)$$

be the calculated and "real" positions and velocities at time t.

Define errors in the same way:

$$e_x(t) = x'(t) - x(t);\ e_v(t) = v'(t) - v(t)$$

The errors will accumulate through the Euler algorithm:

$$e_x(t+h) = e_x(t) + he_v(t)$$
$$e_v(t+h) = e_v(t) - h\omega^2 e_x(t)$$

Define a *stability matrix*

$$e(t+h) = Ae(t) \;;\; e = (e_x, e_v)^T$$
$$A = \begin{pmatrix} 1 & h \\ -\omega^2 h & 1 \end{pmatrix}$$

If matrix A has an eigenvalue λ such that | λ | > 1 , the algorithm is unstable.

Now **| A – λ*I* | = 0** → $\lambda = 1 \pm h\sqrt{-\omega^2}$

So, Euler algorithm is ALWAYS unstable for ODHO independent of h !!

A simple modification to Euler's algorithm makes it *conditionally stable*:

(for positions, sum up 1. order Taylor backwards in time!)

$$x(t+h) = x(t) + [v(t) + h\dot{v}(t)]h$$
$$v(t+h) = v(t) + h\dot{v}(t)$$

The stability matrix is now

$$A = \begin{pmatrix} 1-\omega^2 h^2 & h \\ -\omega^2 h^2 & 1 \end{pmatrix}$$

which has **| λ | < 1 when** $|\omega h| < 2$

VERLET AND "VELOCITY-VERLET" ALGORITHMS

Good practical, stable choices for MD

For positions, sum up Taylor series for x(t+h) + x(t-h) up to 3rd order

$$x(t+h) = 2x(t) - x(t-h) + h^2\ddot{x}(t)$$

+ Effectively third order algorithm (3rd order Taylor terms cancel out)

- Multi-step algorithm (no self-starting)

- No explicit term for velocities, they may be estimated by

$$v(t) \approx \frac{x(t+h) - x(t-h)}{2h}$$

A recommended version is the so-called velocity form:

$$x(t+h) = x(t) + hv(t) + \frac{1}{2}h^2 a(t)$$

$$v(t+h) = v(t) + \frac{1}{2}h\left[a(t) + a(t+h)\right]$$

Identical with the original algorithm (check by reducing velocities)

+ One-step algorithm (self-starting)

+ Positions and velocities known at the same instant of time.

Practical application in a *predictor-corrector* way:

1. **define x(0), v(0), a(0)**
2. **advance x to x(h)**
3. **calculate "half-step velocity" (or "*predicted velocity*")**
   v_{pred} = v(0) + h * a(0) / 2
4. **calculate potential and forces by using x(h); a(h) = F(x(h)) / m**
5. **calculate the *corrected velocity* v(h) = v_{pred} + h * a(h) / 2**
6. **return to 2. Advance x to x(2h) and so on…**

In the sample code below, steps 2 and 3 are done in VERLET **in the first call, step 5 in the second call.**

(real*8 RX(N), RY(N), RZ(N), etc…)

```
DO MDSTEP = 1, NSTEPS
   CALL VERLET(1,DT,N,K,RX,RY,RZ,VX,VY,VZ,AX,AY,AZ)
   CALL LJONES(BOX,RCUT,V,VC,W,N,RX,RY,RZ,FX,FY,FZ)
   AX(:) = FX(:) / MASS
   AY(:) = FY(:) / MASS
   AZ(:) = FZ(:) / MASS
   CALL VERLET(2,DT,N,K,RX,RY,RZ,VX,VY,VZ,AX,AY,AZ)
    …other code… calculation of desired properties at this time
    step… periodic output of data and trajectory…
ENDDO
```