

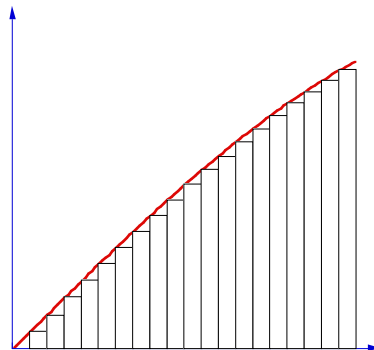
Numerical Integration

RGP I - week 2
Integrating a function
Classical equations of motion

Numerical integration of a function

- ◆ Should be known from the numerics course in the first year
- ◆ is done by replacing the integral by a finite sum, such as:

$$\int_a^b f(x)dx = \frac{b-a}{N} \sum_{i=1}^N f\left(a + i \frac{b-a}{N}\right) + O(1/N)$$



Essential tricks for numerical integration

- ◆ Integrate as much as possible analytically

$$\int_0^1 \int_0^1 yf(x) dx dy = \frac{1}{2} \int_0^1 f(x) dx$$

- ◆ Remove singularities

$$\int_{-1}^1 |x|f(x) dx = \int_{-1}^0 (-x)f(x) dx + \int_0^1 x f(x) dx = \int_0^1 x [f(x) + f(-x)] dx$$

$$\int_0^1 x^{1/3} dx = \int_0^1 3y^3 dy \quad \int_0^1 \frac{f(x)}{\sqrt{1-x^2}} dx = 2 \int_0^1 \frac{f(1-y^2)}{\sqrt{2-y^2}} dy$$

- ◆ Change of variables:

- ◆ Stretch regions with large variations or large values
- ◆ Shrink regions with small variations or small values

$$\int_0^{\infty} f(x) \exp(-x^2) dx = \int_0^1 f(-\ln(x)) \exp(\ln(x)^2) dx$$

Higher order schemes

- ◆ Instead of rectangles

$$\int_a^b f(x) dx = \frac{b-a}{N} \sum_{i=1}^N f\left(a + i \frac{b-a}{N}\right) + O(1/N)$$

- ◆ Use

- ◆ Trapezoidal rule

$$\int_a^b f(x) dx = \frac{b-a}{N} \left(\frac{1}{2} f(a) + \sum_{i=1}^{N-1} f\left(a + i \frac{b-a}{N}\right) + \frac{1}{2} f(b) \right) + O(1/N^2)$$

- ◆ Or parabolas (Simpson rule)

$$\int_a^b f(x) dx = \frac{b-a}{3N} \left(f(a) + \sum_{i=1}^{N-1} (3 - (-1)^i) f\left(a + i \frac{b-a}{N}\right) + f(b) \right) + O(1/N^4)$$

- ◆ for higher order schemes and adaptive integrators review your numerics course or look into text books

The classical equations of motion

- ◆ Newtonian mechanics:

$$m \frac{d^2 \vec{x}}{dt^2} = \vec{F}$$

- ◆ Rewrite as two ordinary differential equations

$$\begin{aligned} \frac{d\vec{x}}{dt} &= \vec{v} \\ \frac{d\vec{v}}{dt} &= \frac{1}{m} \vec{F} \end{aligned}$$

- ◆ Numeric Integration:
 - ◆ replace differential by difference

The forces

- ◆ Often potential forces (no dissipation, no friction)

$$m \frac{d^2 \vec{x}_i}{dt^2} = \vec{F}_i = -\nabla_i U(\vec{x}_1, \dots, \vec{x}_N)$$

- ◆ Often only two-body forces, dependent only on distance plus external potential:

$$U(\vec{x}_1, \dots, \vec{x}_N) = \sum_{i \neq j} V_{ij}(|\vec{x}_i - \vec{x}_j|) + \sum_i V_i^{\text{ext}}(\vec{x}_i)$$

- ◆ However sometimes 3- and many body terms as phenomenological, simplified potentials for complicated and sometimes unknown forces (details in summer)
 - ◆ vibrations around fixed angles between atomic bonds
 - ◆ rotation barriers
 - ◆ phenomenological, effective potentials

Common potentials

- ◆ Gravity and Coulomb potentials

$$V_{ij}^{Gravity}(r) = -G \frac{m_i m_j}{r} \quad V_{ij}^{Coulomb}(r) = \frac{q_i q_j}{r}$$

- ◆ Hard sphere

$$V_{ij}^{hs}(r) = \begin{cases} 0 & r > a \\ \infty & r \leq a \end{cases}$$

- ◆ Lennard-Jones

$$V_{ij}^{LJ}(r) = 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r} \right)^{12} - \left(\frac{\sigma_{ij}}{r} \right)^6 \right]$$

- ◆ Ionic crystal

$$V_{ij}^{ionic}(r) = b_{ij} r^{-n} + \frac{q_i q_j}{r}$$

Forward Euler method

- ◆ 1. Discretize the time

$$t_n = t_0 + n\Delta t \quad \vec{x}_n = \vec{x}(t_n) \quad \vec{v}_n = \vec{v}(t_n) \quad \vec{a}_n = \vec{F}(\vec{x}, t_n) / m$$

- ◆ 2. Taylor expand the solution to 1st order

$$\frac{d\vec{x}}{dt} = \vec{v} \Rightarrow \vec{x}_{n+1} - \vec{x}_n = \vec{v}_n \Delta t + O(\Delta t^2)$$

$$\frac{d\vec{v}}{dt} = \frac{1}{m} \vec{F} \Rightarrow \vec{v}_{n+1} - \vec{v}_n = \vec{a}_n \Delta t + O(\Delta t^2)$$

- ◆ 3. Rearrange to get 1st order forward-Euler integrator

$$\begin{aligned} \vec{x}_{n+1} &= \vec{x}_n + \vec{v}_n \Delta t \\ \vec{v}_{n+1} &= \vec{v}_n + \vec{a}_n \Delta t \end{aligned}$$

- ◆ Problem: unstable for periodic problems

- ◆ See the Mathematica notebook on the web page

Stable first order integrators

◆ Backward Euler:

- ◆ in Euler method replace derivative for positions

$$\frac{d\vec{x}}{dt} = \vec{v} \Rightarrow \vec{x}_{n+1} - \vec{x}_n = \vec{v}_n \Delta t + O(\Delta t^2)$$

$$\frac{d\vec{x}}{dt} = \vec{v} \Rightarrow \vec{x}_n - \vec{x}_{n-1} = \vec{v}_n \Delta t + O(\Delta t^2)$$

- ◆ The second method gives slightly modified but stable method

$$\vec{v}_{n+1} = \vec{v}_n + \vec{a}_n \Delta t$$

$$\vec{x}_{n+1} = \vec{x}_n + \vec{v}_{n+1} \Delta t$$

◆ Midpoint method is a related method which is also stable

$$\vec{v}_{n+1} = \vec{v}_n + \vec{a}_n \Delta t$$

$$\vec{x}_{n+1} = \vec{x}_n + \frac{1}{2}(\vec{v}_n + \vec{v}_{n+1}) \Delta t$$

Order of integration methods

◆ Assume an approximation

$$y(t + \Delta t) - y(t) = f(t) + O(\Delta t^n)$$

◆ then the integrator

$$y(t + \Delta t) = y(t) + f(t)$$

◆ is **locally n-th order**

◆ but **globally (n-1)-th order**,

since the errors add up when integrating over a time T:

$$\begin{aligned} y(T) - y^{\text{numeric}}(T) &= \sum_{n=1}^{T/\Delta t} y(n\Delta t) - y((n-1)\Delta t) - f((n-1)\Delta t) \\ &= \frac{T}{\Delta t} O(\Delta t^n) = O(\Delta t^{n-1}) \end{aligned}$$

A common choice: leap-frog

- ◆ Updates velocities at different times than positions

$$\begin{aligned}\dot{\vec{v}}_{n+1/2} &= \dot{\vec{v}}_{n-1/2} + \dot{\vec{a}}_n \Delta t \\ \vec{x}_{n+1} &= \vec{x}_n + \vec{v}_{n+1/2} \Delta t\end{aligned}$$

- ◆ Looks simple, but surprisingly good
 - ◆ Is actually globally 2nd order and not 1st order!
 - ◆ Exercise: show that it is 2nd order!
- ◆ Is not self starting
 - ◆ Initial step by Euler method:

$$\vec{v}_{1/2} = \vec{v}_0 + \frac{1}{2} \vec{a}_0 \Delta t$$

Velocity-dependent forces

- ◆ Euler Richardson algorithm is better for velocity-dependent forces
- ◆ first calculates the acceleration at an estimated midpoint

$$\vec{a}_{n+1/2} = \frac{1}{m} F\left(\vec{x}_n + \vec{v}_n \frac{\Delta t}{2}, \vec{v}_n + \vec{a}_n \frac{\Delta t}{2}, t_n + \frac{\Delta t}{2}\right)$$

- ◆ then uses this acceleration to update positions and velocities

$$\begin{aligned}\dot{\vec{v}}_{n+1} &= \dot{\vec{v}}_n + \dot{\vec{a}}_{n+1/2} \Delta t \\ \vec{x}_{n+1} &= \vec{x}_n + \left(\vec{v}_n + \vec{a}_{n+1/2} \frac{\Delta t}{2}\right) \Delta t\end{aligned}$$

Verlet algorithm

- ◆ Is a rather simple algorithm which is
 - ◆ Globally 3rd order in the positions
 - ◆ Globally 2nd order in the velocities
- ◆ Good choice since usually positions more important
- ◆ very common choice in molecular dynamics

$$\vec{v}_{n+1} = \vec{v}_n + \frac{1}{2}(\vec{a}_n + \vec{a}_{n+1})\Delta t + O(\Delta t^3)$$

$$\vec{x}_{n+1} = \vec{x}_n + \vec{v}_n\Delta t + \frac{1}{2}\vec{a}_n\Delta t^2 + O(\Delta t^4)$$

Other algorithms

- ◆ The Beeman is another also often used choice

$$\vec{v}_{n+1} = \vec{v}_n + \frac{1}{6}(2\vec{a}_{n+1} + 5\vec{a}_n - \vec{a}_{n-1})\Delta t$$

$$\vec{x}_{n+1} = \vec{x}_n + \vec{v}_n\Delta t + \frac{1}{6}(4\vec{a}_n - \vec{a}_{n-1})\Delta t^2$$

- ◆ especially good at energy conservation for Lennard Jones potentials
- ◆ Other methods will be discussed later
 - ◆ Predictor-Corrector
 - ◆ Runge-Kutta

The classical N -body problem

- ◆ Is very hard for large N
 - ◆ Boundary conditions need to be dealt with effectively
 - ◆ Force calculation
 - ◆ Easy for short range forces
 - ◆ Hard for long range forces
- ◆ Here we will develop a very simple molecular dynamics program for the classical N -body problem
 - ◆ First in C++
 - ◆ Later visualization in Java
- ◆ In-depth discussion in the summer semester
- ◆ This week's exercise:
 - ◆ Write a program for shooting of canon balls

Boundary conditions

- ◆ In a simulation of N particles they will just fly apart to infinity
- ◆ Solution 1:
 - ◆ put particles into a box
 - ◆ problem: **boundary effects**
- ◆ Solution 2:
 - ◆ use periodic boundary conditions with the box
 - ◆ no boundary effects, but still some finite size effects
 - ◆ problems
 - ◆ how to calculate distances? nearest neighbor only, OK if short range forces
 - ◆ how to move particles? translate particles that cross the border back into box
 - ◆ Bigger problem are long range forces -> summer semester

Short range forces

- ◆ fall off faster than r^{-d} (e.g. hard spheres, Lennard Jones)
- ◆ can be cut off at some distance without introducing big errors in the calculation

- ◆ only particles in the vicinity contribute to forces
- ◆ effort is
 - ◆ $O(1)$ for the calculation of force on one particle
 - ◆ $O(N)$ for all forces

- ◆ The only problem is the determination of nearby particles
 - ◆ This will be an exercise problem

Long range forces: $O(N^2)$?

- ◆ Long range forces (Coulomb, Gravity) are a problem
- ◆ all particles contribute!
 - ◆ $O(N)$ for the calculation of force on one particle
 - ◆ $O(N^2)$ for all forces in direct method

- ◆ However special methods exist that scale like $O(N \log N)$ they will be discussed in detail in the summer semester

- ◆ Question:
 - ◆ can you think of an $O(N \log N)$ algorithm?
 - ◆ can you think of an $O(N)$ algorithm?

Applications

- ◆ Will be discussed in detail in the summer semester
- ◆ Some examples, spanning all length scales
 - ◆ 10^{-12}m :
 - ◆ Dynamics of heavy nuclei
 - ◆ 10^{-7}m :
 - ◆ Folding of proteins
 - ◆ $10^{-3}\text{m} \dots 10^6\text{m}$
 - ◆ Kinetic simulation of gases
 - ◆ Fluid dynamics
 - ◆ $10^{10}\text{m} \dots 10^{14}\text{m}$:
 - ◆ Evolution of the solar system
 - ◆ $10^{15}\text{m} \dots 10^{22}\text{m}$:
 - ◆ Galactic dynamics
 - ◆ $10^{23}\text{m} \dots 10^{26}\text{m}$:
 - ◆ Cosmology simulations