Ordinary Differential Equations (PHY1610 lecture 10)

Ramses van Zon

February 10, 2022

Ordinary Differential Equations

- Are equations with derivatives with respect to 1 variable, e.g.

$$\frac{dx}{dt} = f(x, t)$$

- There can be more than one such equation, e.g.

$$\frac{dx^{(1)}}{dt} = f^{(1)}(x^{(1)}, x^{(2)}, t); \quad \frac{dx^{(2)}}{dt} = f^{(2)}(x^{(1)}, x^{(2)}, t)$$

- The derivative can be of higher order to, e.g. $\dfrac{d^2 x}{dt^2} = f(x, t)$

  But this can be written, by setting $x^{(1)} = x$, $x^{(2)} = dx/dt$, to

$$\frac{dx^{(1)}}{dt} = x^{(2)}; \quad \frac{dx^{(2)}}{dt} = f(x^{(1)}, t)$$

SciNet

ODE Examples

Lotka–Volterra (predator/pray)

$$\frac{dx^{(1)}}{dt} = x^{(1)}(\alpha - \beta x^{(2)})$$

$$\frac{dx^{(2)}}{dt} = -x^{(2)}(\gamma - \delta x^{(1)})$$

Harmonic oscillator

$$\frac{dx^{(1)}}{dt} = x^{(2)}$$

$$\frac{dx^{(2)}}{dt} = -x^{(1)}$$

Rate equations (chemistry))

$$\frac{dx^{(1)}}{dt} = -2k_1[x^{(1)}]^2 x^{(2)} + 2k_2[x^{(3)}]^2$$

$$\frac{dx^{(2)}}{dt} = -k_1[x^{(1)}]^2 x^{(2)} + k_2[x^{(3)}]^2$$

$$\frac{dx^{(3)}}{dt} = 2k_1[x^{(1)}]^2 x^{(2)} - 2k_2[x^{(3)}]^2$$

Lorenz system (weather)

$$\frac{dx^{(1)}}{dt} = \sigma(x^{(2)} - x^{(1)})$$

$$\frac{dx^{(2)}}{dt} = x^{(1)}(\rho - x^{(3)}) - x^{(2)}$$

$$\frac{dx^{(3)}}{dt} = x^{(1)}x^{(2)} - \beta x^{(3)}$$

Numerical approaches

Start from the general form:

$$\frac{dx^{(i)}}{dt} = f(x^{(1)}, x^{(2)}, ..., t)$$

- Algorithms for numerically solving ODEs are called integrators.
- All integrators will evaluate f at discrete points t_0, t_1, ...
- Initial conditions: specify $x^{(i)}(t_0)$.
- The time step is typically denoted with h.
- Consecutive points may have a fixed step size $h = t_{k+1} - t_k$ or may be adaptive.

Desirable qualities for an integrator

- *Accuracy*

- *Efficiency*

- *Stability*

- Respect physical laws, e.g.

  Time reversal symmetry
  Conservation of energy
  Conservation of linear momentum
  Conservation of angular momentum
  Conservation of phase space volume

The most efficient algorithm is then the one that allows the largest possible time step for a given level of **accuracy**, while maintaining **stability** and preserving **conservation laws**.

ODE solvers: Forward Euler

To solve:

$$\frac{dx}{dt} = f(t, x)$$

we could take the simple approximation:

$$x_{n+1} \approx x_n + h f(x_n, t_n) \qquad \text{"forward Euler"}$$

Why?

$$x(t_n + h) = x(t_n) + h\frac{dx}{dt}(t_n) + \mathcal{O}(h^2)$$

So:

$$x(t_n + h) = x(t_n) + h f(x_n, t_n) + \mathcal{O}(h^2)$$

So when taking small time steps, this should be accurate.

Accuracy of the forward Euler method

$$x(t_n + h) = x(t_n) + h f(x_n, t_n) + \mathcal{O}(h^2)$$

- $\mathcal{O}(h^2)$ is the local error, i.e., the error in each time step.

- For given trajectory from $t = t_1$ to t_2, we need $n = (t_2 - t_1)/h$ steps.

- The global error, i.e., the error accumulated over the trajectory, is therefore:
  $n \times \mathcal{O}(h^2) = \mathcal{O}(h)$

- Not very accurate.

Stability of the forward Euler method

To solve harmonic oscillator:
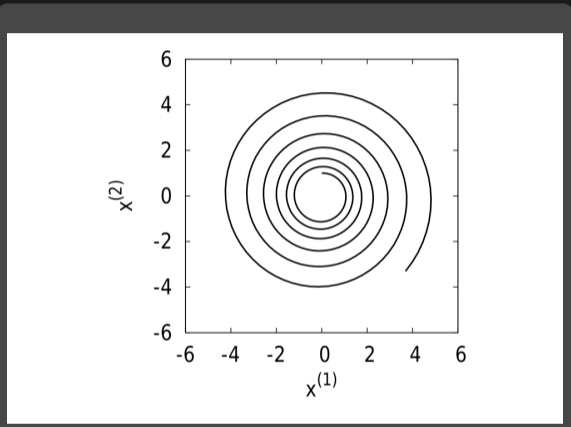
$$\frac{dx^{(1)}}{dt} = x^{(2)}$$

$$\frac{dx^{(2)}}{dt} = -x^{(1)}$$

with forward Euler gives:

$$\begin{pmatrix} x_{n+1}^{(1)} \\ x_{n+1}^{(2)} \end{pmatrix} = \begin{pmatrix} 1 & h \\ -h & 1 \end{pmatrix} \begin{pmatrix} x_n^{(1)} \\ x_n^{(2)} \end{pmatrix}$$

Stability governed by eigenvalues.

$\lambda_{\pm} = 1 \pm ih$ of that matrix.

$|\lambda_{\pm}| = \sqrt{1 + h^2} > 1$ Unstable for any h!

Monitoring Stability

- For the harmonic oscillator, we know the exact answer, so it's easy to see that the forward Euler integrator is unstable.

- For systems without an exact solution, one may still know that some quantities should be bounded.

- Many physical systems have conserved energy, so we can monitor the energy as a function of time.
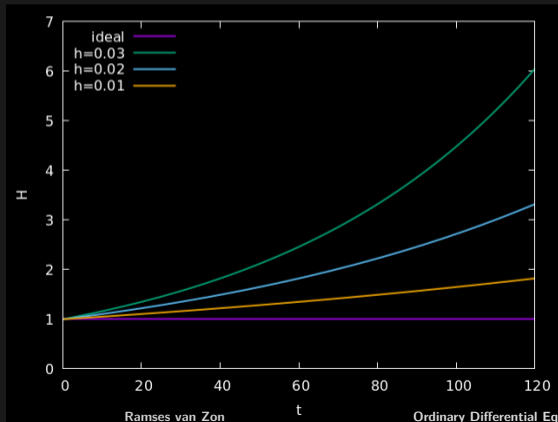
Harmonic oscillator:

$$H = \frac{1}{2}[x^{(1)}]^2 + \frac{1}{2}[x^{(1)}]^2$$

So smaller h does help, but in the long run ($t \sim \mathcal{O}(1/h)$), unstable.

ODE solvers: implicit mid-point Euler

Equation to solve:

$$\frac{dx}{dt} = f(x, t)$$

Symmetric simple approximation:

$$x_{n+1} \approx x_n + h f((x_n + x_{n+1})/2, t_n) \qquad \text{"mid-point Euler"}$$

This is an implicit formula, i.e., has to be solved for x_{n+1}.

Example: Harmonic oscillator

$$\begin{bmatrix} 1 & -\frac{h}{2} \\ \frac{h}{2} & 1 \end{bmatrix} \begin{bmatrix} x_{n+1}^{[1]} \\ x_{n+1}^{[2]} \end{bmatrix} = \begin{bmatrix} 1 & \frac{h}{2} \\ -\frac{h}{2} & 1 \end{bmatrix} \begin{bmatrix} x_n^{[1]} \\ x_n^{[2]} \end{bmatrix} \Rightarrow \begin{bmatrix} x_{n+1}^{[1]} \\ x_{n+1}^{[2]} \end{bmatrix} = M \begin{bmatrix} x_n^{[1]} \\ x_n^{[2]} \end{bmatrix}$$

Eigenvalues M are $\lambda_{\pm} = \frac{(1 \pm ih/2)^2}{1 + h^2/4}$ so $|\lambda_{\pm}| = 1$

Stable for all h!

> Implicit methods often more stable and allow larger step size h.

ODE solvers: implicit mid-point Euler
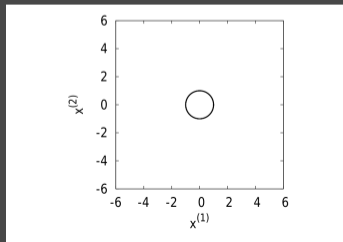
Equation to solve:

$$\frac{dx}{dt} = f(x, t)$$

Symmetric simple approximation:

$$x_{n+1} \approx x_n + hf((x_n + x_{n+1})/2, t_n) \qquad \text{"mid-point Euler"}$$

This is an implicit formula, i.e., has to be solved for x_{n+1}.

Example: Harmonic oscillator

ODE solvers: Predictor-Corrector

- Computation of new point

- Correction using that new point

- Gear P.C.: keep previous values of x to do higher order Taylor series (predictor), then use f in last point to correct.

  Can suffer from catastrophic cancellation at very low h.

- Runge-Kutta: Refines by using mid-points. 4th order version:

$$k_1 = hf(t, x)$$
$$k_2 = hf(t + h/2, x + k_1/2)$$
$$k_3 = hf(t + h/2, x + k_2/2)$$
$$k_4 = hf(t + h, x + k_3)$$
$$x' = y + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

Further ODE solvers techniques

Adaptive methods

Rather than taking a fixed h, we can vary h such that the solution has a certain accuracy.

Methods that adjust the time step as the computation proceeds are known as adaptive methods.

Don't code this yourself!

Adaptive schemes are implemented in libraries such as the `gsl` and `boost::numeric::odeint`.

GSL ODE example: Van der Pol equation (GSL docs)

```c
#include <stdio.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
int func(double t, const double y[], double f[],
         void *params) {
  double mu = *(double *)params;
  f[0] = y[1];
  f[1] = -y[0] - mu*y[1]*(y[0]*y[0] - 1);
  return GSL_SUCCESS;
}
int jac(double t, const double y[], double *dfdy,
        double dfdt[], void *params) {
  double mu = *(double *)params;
  gsl_matrix_view dfdy_mat
    = gsl_matrix_view_array (dfdy, 2, 2);
  gsl_matrix * m = &dfdy_mat.matrix;
  gsl_matrix_set (m, 0, 0, 0.0);
  gsl_matrix_set (m, 0, 1, 1.0);
  gsl_matrix_set (m, 1, 0, -2.0*mu*y[0]*y[1] - 1.0)
  gsl_matrix_set (m, 1, 1, -mu*(y[0]*y[0] - 1.0));
  dfdt[0] = dfdt[1] = 0.0;
  return GSL_SUCCESS;
}
```

```c
int main() {
  const gsl_odeiv_step_type * T
    = gsl_odeiv_step_rk8pd;
  gsl_odeiv_step * s
    = gsl_odeiv_step_alloc (T, 2);
  gsl_odeiv_control * c
    = gsl_odeiv_control_y_new (1e-6, 0.0);
  gsl_odeiv_evolve * e
    = gsl_odeiv_evolve_alloc (2);
  double mu = 10;
  gsl_odeiv_system sys = {func, jac, 2, &mu};
  double t = 0.0, t1 = 100.0;
  double h = 1e-6;
  double y[2] = { 1.0, 0.0 };
  while (t < t1) {
      int status = gsl_odeiv_evolve_apply (e,c,s,
                        &sys, &t, t1, &h, y);
      if (status != GSL_SUCCESS) break;
      printf("%.5e %.5e %.5e\n", t, y[0], y[1]);
    }
  gsl_odeiv_evolve_free(e);
  gsl_odeiv_control_free(c);
  gsl_odeiv_step_free(s);
}
```

1

# Special case #1: Molecular Dynamics

Molecular Dynamics Simulations

Used in chemical physics, materials science and the modelling of bio-molecules.

- N interacting particles
- $m_i \ddot{r}_i = F_i(r_1, r_2, \vdots t)\}$
  - $+$ initial conditions

What makes this different from other ODEs?

- Hamiltonian dynamics
- Very expensive evaluation of F if N is large
- Large simulation times needed

N-body simulation fall within this class as well; the numerics does not care whether the particles are atoms or stars.
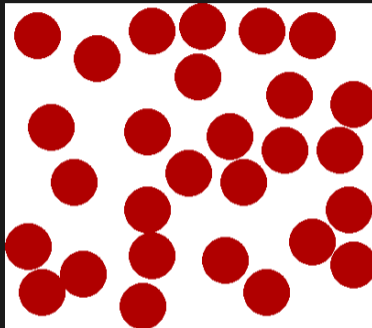
Hamiltonian dynamics

- Molecular Dynamics aims to compute *equilibrium*, *thermodynamic* and *transport* properties of *classical many body systems*.

- Often, the energy is of the form $H = \frac{|p|^2}{2m} + \Phi(r)$ (a.k.a. the Hamiltonian), and is conserved under the dynamics.

- In that case, the systems follows Newton's equations of motion:

$$\dot{r} = \frac{1}{m}p \qquad\qquad \dot{p} = F = -\frac{\partial \Phi}{\partial r},$$

- Potential energy Φ is typically a sum of pair potentials:

$$\Phi(r) = \sum_{(i,j)} \varphi(r_{ij}) = \sum_{i=1}^{N} \sum_{j=1}^{i-1} \varphi(r_{ij}),$$

which entails the following expression for the forces F:

$$\mathbf{F}_i = \sum_{j \neq i} \varphi'(r_{ij}) \frac{\mathbf{r}_j - \mathbf{r}_i}{r_{ij}}$$

Boundary conditions

- When simulating finite systems, a wall potential would give finite size effects and destroy translation invariance.

- More benign: *Periodic Boundary Conditions*

- All particles in box have coordinates between $-L/2$ and $L/2$.

- A particle exiting simulation box is put back at the other end.

- The box with thick red boundaries is our simulation box.

- Other boxes are copies, or "periodic images".
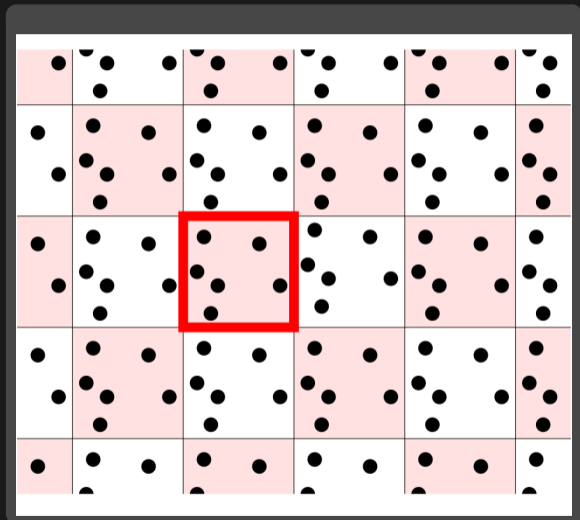
Force calculations

- A common pair potential between neutral, spherical particles (atoms) is the Lennard-Jones potential

$$\varphi(r) = 4\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right],$$

σ is a measure of the range of the potential.

ε is its strength.

The potential is positive for small r: repulsion.

The potential is negative for large r: attraction.

The potential goes to zero for large r: short-range.
   The potential has a minimum of $-\varepsilon$ at $2^{1/6}\sigma$.

Force calculations

- Avoid infinite sums: modify the potential such that it becomes zero beyond a certain *cut-off* distance r_c:

$$\varphi'(r) = \begin{cases} \varphi(r) - \varphi(r_c) & \text{if } r < r_c \\ 0 & \text{if } r \geq r_c \end{cases}$$

  where the subtraction of $\varphi(r_c)$ prevents discontinuities.

- Computing all forces in an N-body system requires the computation of $N(N-1)/2$ forces $\mathbf{F}_{ij}$

- Force computation often the most demanding part of MD.

Cutoff Lennard-Jones potentials, $\varepsilon=\sigma=1$, $r_c = 2.5$, $r'_c = 2$

ϕ
ϕ, cutoff
$\alpha\,\phi$
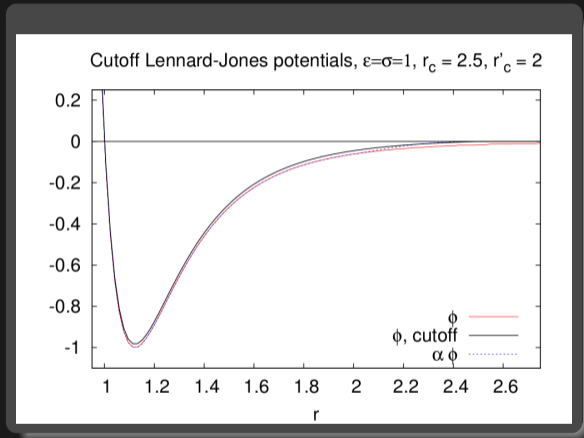
Streamlining the force evaluation

Cell divisions

- Divide the simulation box into cells larger than the cutoff r_c.

- Make a list of all particles in each cell.

- In the sum over pairs in the force computation, only sum pairs of particles in the same cell or in adjacent cells.

Neighbour lists

- Make a list of pairs of particles that are closer than $r_c + \delta r$.

- Sum over the list of pairs to compute the forces.

- The neighbour lists are to be used in subsequent force calculations as long as the list is still valid.

- Invalidation criterion: a particle has moved more than $\delta r / 2$.

For systems with short-range interactions: $\mathcal{O}(N^2) \rightarrow \mathcal{O}(N)$.

Symplectic integrators

MD applications typically contain their own specialized integrator(s).

Reaching long times is paramount, so the stability of the integrator is the most important criteria.

So-called **symplectic integrators** turn out to be particularly stable.

These consists of substeps, each generated by its own Hamiltonian.

Verlet Scheme (first version)

$$r_{n+1} = r_n + \frac{p_n}{m}h + \frac{F_n}{2m}h^2$$
$$p_{n+1} = p_n + \frac{F_{n+1} + F_n}{2}h$$

The momentum rule appears to make this an implicit rule since F_{n+1} is required, but not if F only depends on r!

Verlet Scheme (second version)

The extra storage step can be avoided by introducing the half step momenta as intermediates:

$$p_{n+1/2} = p_n + \frac{1}{2}F_n h$$
$$r_{n+1} = r_n + \frac{p_{n+1/2}}{m}h$$
$$p_{n+1} = p_{n+1/2} + \frac{1}{2}F_{n+1}h$$

Where are the MD libraries?

There typically aren't any. MD packages are usually applications with a lot of parameters, that used other libraries. Examples:

- Gromacs
- NAMD
- LAMMPS

which all differ in intended usages, available force fields, serial speed (platform dependent), parallel scalability, etc.

Frameworks

Some MD packages come more as frameworks, which could be used as a library, within e.g. a C++ program. OpenMM out of Stanford is a prime example which is still actively maintained

(and in fact used in the GPU implementation of Gromacs).

https://simtk.org/home/openmm

2

Special case #2: N-Body Gravitation

Long-range interaction

- Long-range interactions are those that cannot be cut off without seriously altering the physics.

- Gravity is an example of a long range interaction.

- Electrostatic is another, so much of this section applies to MD with charged molecules or atoms as well.

- Let's assume we have a good integrator already, so the bottleneck is the force computation.

- Without a cut-off, computing the sum over pairs, or "Particle-Particle", $\mathcal{O}(N^2)$ methods seem unavoidable.

Luckily, there exists special techniques such as

- Barnes-Hut
- Particle Mesh
- Particle-Particle/Particle Mesh (P3M) or Ewald Sums.

Barnes-Hut

- Cells with too many particles are subdivided (recursive)

- Leads to a quad (2d) or octal (3d) tree

- Pretend each box is one massive particle at the centre to compute the force. (or, replace by a multipole expansion.)
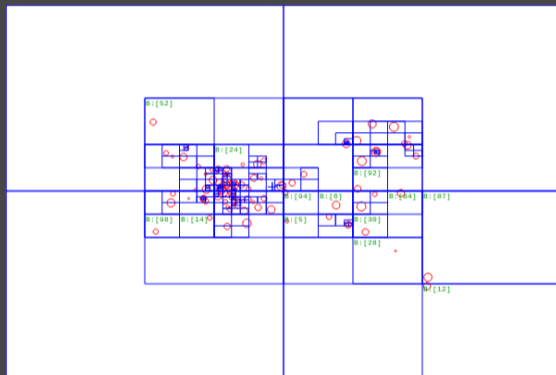
2D Barnes-Hut tree partitioning of 100 bodies for gravitational interactions.

commons.wikimedia.org/wiki/File:2D_Quad-Tree_partitioning_of_100_bodies.png
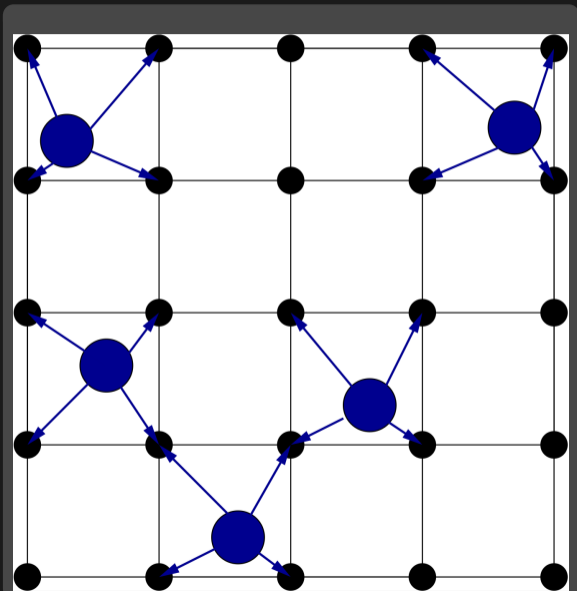
Particle-Mesh

- Choose a fixed-size rectangular mesh

- Distribute masses (blue large circles) to mesh vertices (little black circles)

- Determine gravitational potential using FFT:

$$\nabla^2 \Phi = 4\pi G\rho \Rightarrow \hat{\Phi} = -\frac{4\pi G\hat{\rho}}{k^2}$$

- The forces on the lattice are given by the $\nabla \Phi$ in real space, i.e, the fourier inverse of

$$\hat{F} = i\mathbf{k}\hat{\Phi} = -i\mathbf{k}\frac{4\pi G\hat{\rho}}{k^2}$$

- The inverse FFT gives the real force to move the particles with.

- $\mathcal{O}(N \log N)$.

Ewald Sums and P3M

- Particle-Mesh is fast, but not very accurate.

- This is because the short range part of the forces is poorly represented.

- One can do better.

- Idea of P3M or Ewald summation is to do an exact summation of forces with bodies nearby, and perform an approximate calculation for bodies further away.

- Ewald does not assign to grid, but pays for this: $\mathcal{O}(N^{3/2})$.

- P3M still assigns masses to a regular grid, allowing for $\mathcal{O}(N \log N)$ scaling.

- It relies on being able to translate this separation of local and further-away in fourier space.

- Many choices possible, some better than others: quite outside the scope of this lecture, best stop.