Ordinary Differential Equations and Molecular Dynamics

Ramses van Zon

SciNet, University of Toronto

Scientific Computing Lecture 12

February 13, 2014

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.

$$\frac{dx_1}{dt} = f_1(x_1, x_2, t); \qquad \frac{dx_2}{dt} = f_2(x_1, x_2, t)$$

- The derivative can be of higher order to, e.g.

$$\frac{d^2x}{dt^2} = f(x, t)$$

  But this can be written, by setting $x_1 = x$, $x_2 = dx/dt$, to

$$\frac{dx_1}{dt} = x_2; \qquad \frac{dx_2}{dt} = f(x_1, t)$$

- This class includes Newtonian dynamics (+ some extensions), which is the what is solved in “Molecular Dynamics.”

Numerical approaches

Start from the general form:

$$\frac{dy_i}{dx} = f(x, \{y_j\})$$

- All approaches will evaluate f at discrete points $x_0, x_1, \ldots$.
- Initial conditions: specify $y_i(x_0)$ and $\frac{dy_i}{dx}(x_0)$.
- Consecutive points may have a fixed step size $h = x_{k+1} - x_k$ or may be adaptive.
- $\{y_j(x_{i+1})\}$ may be implicitly dependent on f at that value.

Stiff ODEs

- A stiff ODE is one that is hard to solve, i.e. requiring a very small stepsize **h** or leading to instabilities in some algoritms.
- Usually due to wide variation of time scales in the ODEs.
- Not all methods equally suited for stiff ODEs

ODE solver algorithms: Euler

To solve:

$$\frac{dy}{dx} = f(x, y)$$

Simple approximation:

$$y_{n+1} \approx y_n + hf(x_n, y_n) \qquad \text{“forward Euler”}$$

Rational:

$$y(x_n + h) = y(x_n) + h\frac{dy}{dx}(x_n) + \mathcal{O}(h^2)$$

So:

$$y(x_n + h) = y(x_n) + hf(x_n, y_n) + \mathcal{O}(h^2)$$

- $\mathcal{O}(h^2)$ is the local error.
- For given interval $[x_1, x_2]$, there are $n = (x_2 - x_1)/h$ steps
- Global error: $n \times \mathcal{O}(h^2) = \mathcal{O}(h)$
- Not very accurate, nor very stable (next): don't use.

Stability

Example: solve harmonic oscillator numerically:

$$\frac{dy^{(1)}}{dx} = y^{(2)}$$

$$\frac{dy^{(2)}}{dx} = -y^{(1)}$$

Use Euler ($y_{n+1} \approx y_n + hf(x_n, y_n)$) gives

$$\begin{pmatrix} y^{(1)}_{n+1} \\ y^{(2)}_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & h \\ -h & 1 \end{pmatrix} \begin{pmatrix} y^{(1)}_n \\ y^{(2)}_n \end{pmatrix}$$

Stability governed by eigenvalues $\lambda_\pm = 1 \pm ih$ of that matrix.
$|\lambda_\pm| = \sqrt{1 + h^2} > 1 \quad \Rightarrow$ Unstable for any h!

ODE solver algorithms: implicit mid-point Euler

To solve:

$$\frac{dy}{dx} = f(x,y)$$

Symmetric simple approximation:

$$y_{n+1} \approx y_n + hf(x_n, (y_n + y_{n+1})/2) \qquad \text{“mid-point Euler”}$$

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

Example (Harmonic oscillator)

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

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

$$\boxed{\text{Implicit methods often more stable and allow larger step size } h.}$$

ODE solver algorithms: Predictor-Corrector

1. Computation of new point
2. Correction using that new point

- Gear P.C.: keep previous values of **y** to do higher order Taylor series (predictor), then use **f** in last point to correct. Can suffer from catestrophic cancellation at very low **h**.
- Runge-Kutta: Refines by using mid-points. Workhorse even behind fancier solvers.

4th order version:

$$\begin{aligned}
k_1 &= hf(x, y) \\
k_2 &= hf(x + h/2, y + k_1/2) \\
k_3 &= hf(x + h/2, y + k_2/2) \\
k_4 &= hf(x + h, y + k_3) \\
y' &= y + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}
\end{aligned}$$

Further ODE solver techniques

Adaptive methods

As with the integration, rather than taking a fixed **h**, vary **h** such that the solution has a certain accuracy.

Don't code this yourself! Adaptive schemes are implemented in libraries such as the `gsl` and `boost::numeric::odeint`.

Geometric, symplectic and variants

Respects hamiltonian form, better energy conservation.
Will discuss in the context of MD.

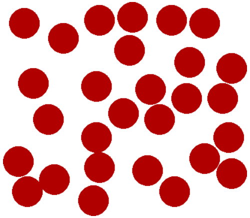
Molecular Dynamics

Molecular Dynamics Simulations

N interacting particles

$$\mathbf{m_i \ddot{r}_i = f_i(\{r\}, \{\dot{r}_j\}, t)}$$

+ initial conditions



What makes this different from other ODEs?

- Hamiltonian dynamics
- Very expensive evaluation of **f** if **N** is large

For both, we will only touch upon some aspects.

Note that **N**-body simulation fall within this class as well; the numerics does not case whether the particles are molecules or stars.

Hamiltonian dynamics

- Molecular Dynamics aims to compute *equilibrium*, *dynamical* and *transport* properties of *classical many body systems*.
- Many classical systems have Newtonian equations of motion:

$$\dot{\mathbf{r}} = \frac{1}{\mathbf{m}}\mathbf{p} \qquad\qquad \dot{\mathbf{p}} = \mathbf{F} = -\frac{\mathbf{dU}}{\mathbf{dr}},$$

  or $\dot{\mathbf{x}} = \mathbf{Lx}$, with $\mathbf{LA} = \{\mathbf{A}, \mathbf{H}\}$, where $\mathbf{x} = (\mathbf{r}, \mathbf{p})$.
- Energy $\mathbf{H} = \frac{|\mathbf{p}|^2}{\mathbf{2m}} + \mathbf{U}(\mathbf{r})$ is conserved under the dynamics.
- Potential energy is typically a sum of pair potentials:

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

  which entails the following expression for the forces **F**:

$$\mathbf{F_i} = -\sum_{\mathbf{j}\neq\mathbf{i}} \frac{\mathbf{d}}{\mathbf{dr_i}}\varphi(\mathbf{r_{ij}}) = \sum_{\mathbf{j}\neq\mathbf{i}} \underbrace{\varphi'(\mathbf{r_{ij}})\frac{\mathbf{r_j} - \mathbf{r_i}}{\mathbf{r_{ij}}}}_{\mathbf{F_{ij}}}$$

Hamiltonian dynamics as disguished sampling

- If the system is ergodic then time average equals the microcanonical average:

$$\lim_{t_{\text{final}}\to\infty} \frac{1}{t_{\text{final}}} \int_0^{t_{\text{final}}} dt\, A(x(t)) = \frac{\int dx\, A(x)\, \delta(E - H(x))}{\int dx\, \delta(E - H(x))}.$$

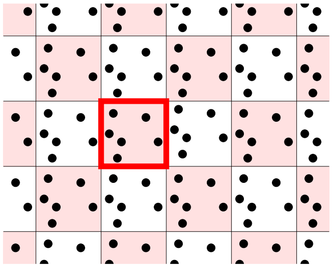
- For large **N**, microcanonical and canonical averages are equal for many quantities **A**.

- Need long times t_{final}!

- Helps to forget initial equilibration time t_{burnin}.

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 $-\mathbf{L}/2$ and $\mathbf{L}/2$.
- 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 is the Lennard-Jones potential

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

  - $\boldsymbol{\sigma}$ 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 $\mathbf{2^{1/6}}\boldsymbol{\sigma}$.
- Computing all forces in an N-body system requires the computation of $\mathbf{N(N-1)/2}$ forces $\mathbf{F_{ij}}$
- Force computation often the most demanding part of MD.

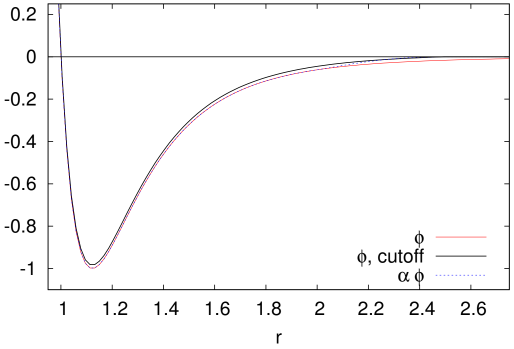
Force calculations

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

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

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

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



Streamlining the force evaluation

Cell divisions

- Divide the simulation box into cells larger than the cutoff $\mathbf{r_c}$.
- 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 $\mathbf{r_c} + \delta\mathbf{r}$.
- 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 $\delta\mathbf{r}/\mathbf{2}$.

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

Force calculations

Long-range interaction

Electrostatics, gravity, are example of long range interactions that cannot be cut off without seriously altering the physics.
Requires special techniques such as

- Barnes-Hut
- Particle Mesh Ewald

Desirable qualities for a molecular dynamics integrator

- *Accuracy*
- *Efficiency*
- *Stability*
- *Respect physical laws:*
  - 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*.

Symplectic integrators

Momentum Verlet Scheme (first version)

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \frac{\mathbf{p}_n}{m}h + \frac{\mathbf{F}_n}{2m}h^2$$
$$\mathbf{p}_{n+1} = \mathbf{p}_n + \frac{\mathbf{F}_{n+1} + \mathbf{F}_n}{2}h$$

The momentum rule appears to pose a problem since $\mathbf{F}_{n+1}$ is required. But to compute $\mathbf{F}_{n+1}$, we need only $\mathbf{r}_{n+1}$, which is computed in the integration step as well.
Equivalent to position Verlet scheme.

Symplectic integrators

Momentum Verlet Scheme (second version)

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

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

Also nice and symmetric:

1. Half momentum step
2. Full position step
3. Half momentum step

First step the same as the last (with updated F).

Symplectic integrators from Hamiltonian splitting methods

- For sampling, one wants a long trajectory (formally $\mathbf{t_f} \longrightarrow \infty$).
- It is therefore important that an integration algorithm be stable.
- The momentum Verlet scheme, on the other hand, is much more stable than, say, the Euler scheme.
- To see why, one should re-derive the momentum Verlet scheme from a completely different starting point, using a so-called *Hamiltonian splitting method* (also known as *Geometric integration*).

Symplectic integrators from Hamiltonian splitting methods

Very, very briefly:

- Any Hamiltonian $\mathbf{H}$ a flow on phase space: $\mathbf{U(t)}$
- Split up Hamiltonian in $\mathbf{k}$ parts, $\mathbf{H_1 \ldots H_k}$.
- Gives $\mathbf{k}$ flows: $\mathbf{U_1 \ldots U_K}$.
- Baker-Campbell-Hausdorff formula gives approximate factorization, e.g. $\mathbf{H = H_1 + H_2 \Rightarrow U(h) \approx U_1(h/2)U_2(h)U_1(h/2)}$
- Symmetric form reduced order and preserves time reversibility.
- This is momentum Verlet!
- Further using BCH, one can derive a shadow Hamiltonian.
- $\Rightarrow$ simulated system retains all hamiltonian properties.

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

Homework

Homework (page 1/2)

Code, git-log, makefile, and text file describing the results due on Feb 20.

Compute the position $\mathbf{(x_0, y_0)}$ of at least one minimum of the function:

$$\mathbf{f(x,y) = (3 - e^{-x^2})(1+y^2)^{1/2}\tanh\cosh\left[e^{-4x^2}(1+y)\right]}$$

for $\mathbf{x} > \mathbf{0}$. Also compute the value $\mathbf{f(x_0, y_0)}$.

You are asked to use two different methods:

1. Steepest Descent
2. Conjugate Gradient

Compare the number of function evaluations between the two.

For this assignment, you have to install and use the GNU scientific library (GSL). You'll have to study the GSL documentation as well. Note that the GSL has 2 conjugate gradient routines: use the Polak-Ribiere variant. . . .

Homework (page 2/2)

Both of these minimizations require derivatives, so here they are.

Let's define 4 intermediate results first:

$$\begin{aligned} \mathbf{A} &= \mathbf{e}^{-\mathbf{x}^2} \\ \mathbf{B} &= (\mathbf{1}+\mathbf{y}^2)^{1/2} \\ \mathbf{C} &= \mathbf{tanh\,cosh}[\mathbf{A}^4(\mathbf{1}+\mathbf{y})] \\ \mathbf{D} &= \mathbf{sinh}[\mathbf{A}^4(\mathbf{1}+\mathbf{y})] \end{aligned}$$

Then

$$\begin{aligned} \mathbf{f} &= (\mathbf{3}-\mathbf{A})\mathbf{BC} \\ \frac{\partial \mathbf{f}}{\partial \mathbf{x}} &= \mathbf{2xAB}\left[\mathbf{C}-\mathbf{4}(\mathbf{3}-\mathbf{A})\mathbf{A}^3(\mathbf{1}-\mathbf{C}^2)\mathbf{D}(\mathbf{1}+\mathbf{y})\right] \\ \frac{\partial \mathbf{f}}{\partial \mathbf{y}} &= (\mathbf{3}-\mathbf{A})\left[\frac{\mathbf{yC}}{\mathbf{B}}+\mathbf{A}^4\mathbf{B}(\mathbf{1}-\mathbf{C}^2)\mathbf{D}\right] \end{aligned}$$

(the **x** in the second line is not a multiplication sign).