N-body simulation

Part 2: N-body methods

Outline

- Time discretisation / integration schemes
  - Derivation
  - Understanding accuracy (errors)
  - Checking correctness & stability
- Force calculation schemes
  - Direct (brute force)
  - Neighbour lists
  - Hierarchical / Tree based
  - Particle-mesh, P^{3}M
  - Periodic boundary conditions
- N-body force “fixes”

N-body dynamics

- Assume, e.g. at t=0
  - position $\vec{r} = (x, y, z)$ of each particle known
  - velocity $\vec{v} = (v_x, v_y, v_z)$ of each particle known
  - total force $\vec{F} = (F_x, F_y, F_z)$ on each particle has been computed
- Acceleration, velocity, and position of particle *i* obey:

$$\boxed{\vec{a_i} = \frac{\vec{F_i}}{m_i} \qquad \frac{d\vec{v_i}}{dt} = \vec{a_i} \qquad \frac{d\vec{r_i}}{dt} = \vec{v_i}}$$

- How do we compute the trajectories of all particles?

N-body dynamics

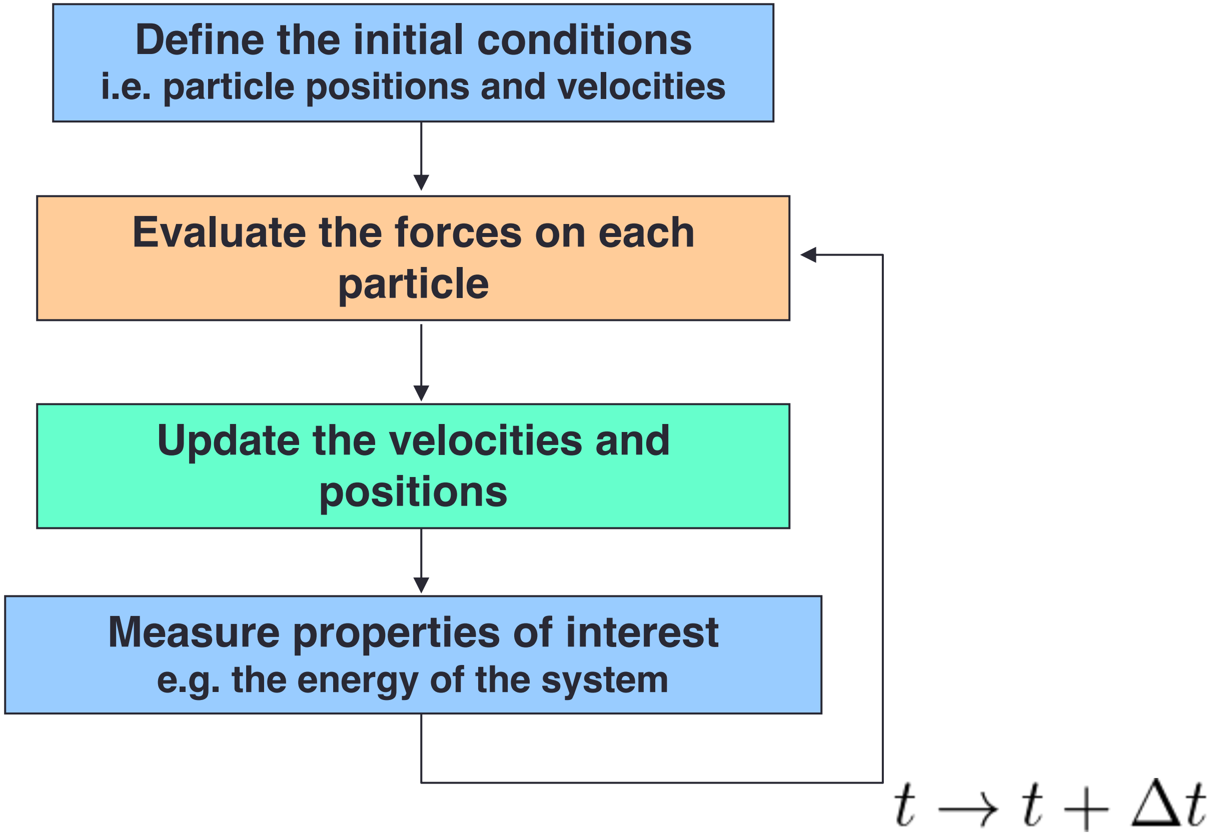
- Suppose $\vec{F_i}$ constant, then after time t

$$\vec{v_i}(t) = \vec{v_i}(0) + \vec{a_i}t$$

$$\vec{r_i}(t) = \vec{r_i}(0) + \vec{v_i}(0)t + \frac{1}{2}\vec{a_i}t^2$$

- Actually $\vec{F_i}$ changes as particles move!
  - Above equations approximate exact continuous-time dynamics
  - Improve by dividing t into intervals of size Δt
  - For each Δt interval:
    - Update particle velocities and positions
    - Recompute forces using updated particle positions
  - Smaller Δt → better approximation (but more iterations needed)

General N-body simulation algorithm



Define the initial conditions
i.e. particle positions and velocities

Evaluate the forces on each particle

Update the velocities and positions

Measure properties of interest
e.g. the energy of the system

$t \to t + \Delta t$

N-body dynamics

- To solve differential equations of classical mechanics numerically, introduce time discretisation / integration:
  - Discretisation: evaluate forces, velocities, positions at discrete times separated by small time intervals
  - Integration: compute how the dynamics evolve by accumulating changes over many discrete time steps
- Next:
  - Mathematical derivation of numerical integration schemes
  - Understand accuracy

Time discretisation / integration schemes

Taylor Expansion

- Express a function at the point $x + h$ as an infinite series of its derivatives evaluated at x :

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

Brook Taylor
1685-1731

- Use Taylor expansion to re-express dynamical equations
  - i.e. instead of f(x), consider v(t) and x(t)
- Truncate series at some chosen order m
  - i.e. only include terms with n < m
  - Use the results directly (or combine in clever ways) to get different numerical time integration schemes

The Euler scheme – forward difference

- Simplest is Euler integration:
- Consider Taylor expansion to first order

Leonhard Euler
1707-1783

$$f(x+h) = f(x) + \underline{f'(x)}h + \underline{\mathcal{O}(h^2)}$$

$$f'(x) \equiv f^{(1)} \equiv \frac{df(x)}{dx}$$

Ignore terms of this power (and higher)

$$\boxed{\begin{array}{l} x \rightarrow t \quad h \rightarrow \delta t \\ f'(x) \rightarrow \dfrac{f(t)}{dt} \end{array}}$$

derivative of displacement is velocity
derivative of velocity is acceleration

- This gives:

$$\boxed{\begin{aligned} v(t+\delta t) &= v(t) + a(t)\delta t \\ x(t+\delta t) &= x(t) + v(t)\delta t \end{aligned}}$$

Deriving the Euler (forward difference) scheme

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \mathcal{O}(h^3)$$

$$\begin{aligned} v(t+\delta t) &= v(t) + v'(t)\delta t + \frac{1}{2}v''(t)(\delta t)^2 + \mathcal{O}((\delta t)^3) \\ &= v(t) + a(t)\delta t + \mathcal{O}((\delta t)^2) \end{aligned}$$

$$\begin{aligned} x(t+\delta t) &= x(t) + x'(t)\delta t + \frac{1}{2}x''(t)(\delta t)^2 + \mathcal{O}((\delta t)^3) \\ &= x(t) + v(t)\delta t + \frac{1}{2}a(t)(\delta t)^2 + \mathcal{O}((\delta t)^3) \\ &= x(t) + v(t)\delta t + \mathcal{O}((\delta t)^2) \end{aligned}$$

Ignoring $\mathcal{O}((\delta t)^2)$ gives:

$$\boxed{\begin{aligned} v(t+\delta t) &= v(t) + a(t)\delta t \\ x(t+\delta t) &= x(t) + v(t)\delta t \end{aligned}}$$

Euler scheme error

- Truncated Taylor Expansion is an approximation
  - What is the error if we use it in simulation?
- One time step:
  - From Euler scheme derivation, error (the ignored terms) is $\mathcal{O}\left((\delta t)^2\right)$
  - Error after just one time step is called “local error”
- N time steps, covering 1 unit of time: $N\delta t = 1$
- Total error after N time steps (“global error”):

$$\mathcal{O}(N(\delta t)^2) = \mathcal{O}(\frac{1}{\delta t}(\delta t)^2) = \mathcal{O}(\delta t)$$

- Applies to position as well as velocity
- Global error linear in step size → “accurate to first order”
- Doesn’t conserve energy!

Verlet – second order scheme

Loup Verlet
b. 1931

- Better, **time-reversal symmetric** and **more accurate** scheme

- Combines forward and backward Taylor expansions:

$$\vec{x}(t+\delta t) = 2\vec{x}(t) - \vec{x}(t-\delta t) + \vec{a}(\delta t)^2 + \mathcal{O}((\delta t)^4)$$

  - (truncate $\mathcal{O}((\delta t)^4)$ to give the equation actually used in the scheme)

- Velocities do not appear explicitly, but can be found using:

$$\vec{v}(t+\delta t) = \frac{x(t+\delta t) - x(t-\delta t)}{2\delta t}$$

- Conserves average energy for conservative forces

Verlet – second order scheme

Loup Verlet
b. 1931

Discretisation errors:

Position error per timestep is $\mathcal{O}((\delta t)^4)$

global error = $\mathcal{O}((\delta t)^2)$ multiplied by an exponential function of number of steps!

Velocity error per timestep is $\mathcal{O}((\delta t)^2)$, can show this also = global error

This algorithm is not "self starting"

Requires one Euler step to be performed first to compute $\vec{x}(0+\delta t)$

Other forms exist to address these shortcomings:

"Velocity Verlet" and **"Leapfrog"**

Leapfrog – second order scheme

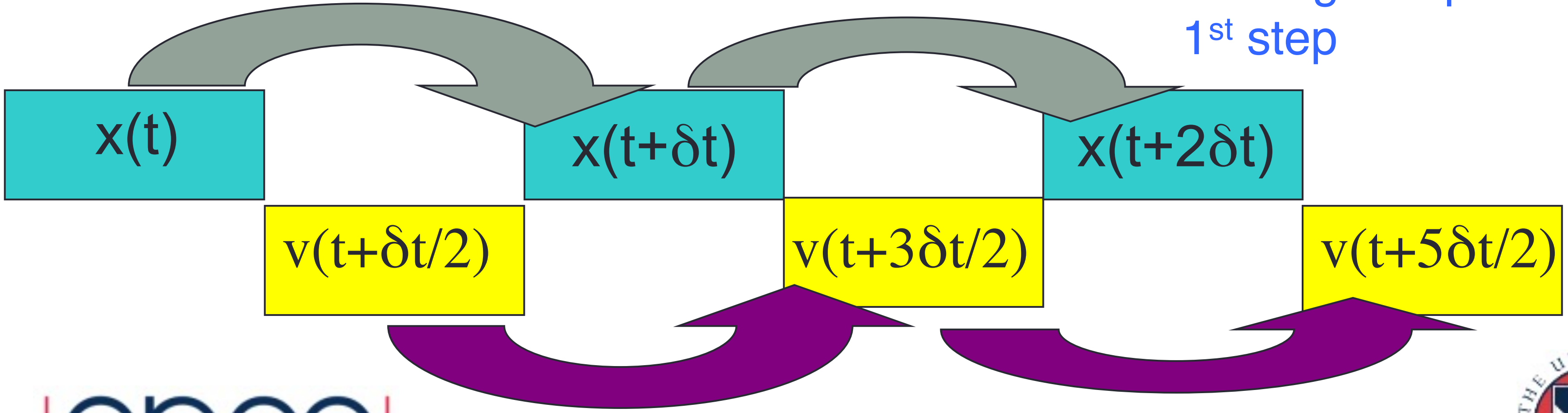
- Uses half-step velocities:

$$\vec{x}(t+\delta t) = \vec{x}(t) + \vec{v}(t + {}^1\!/\!_2\delta t) \times \delta t$$

$$\vec{v}(t + {}^1\!/\!_2\delta t) = \vec{v}(t - {}^1\!/\!_2\delta t) + \vec{a}(t) \times \delta t$$

- Combines forward and backward 2nd order Taylor expansion
  - Local error in position and velocity is $\mathcal{O}((\delta t)^3)$

Algorithm is not self-starting: Require Euler 1st step

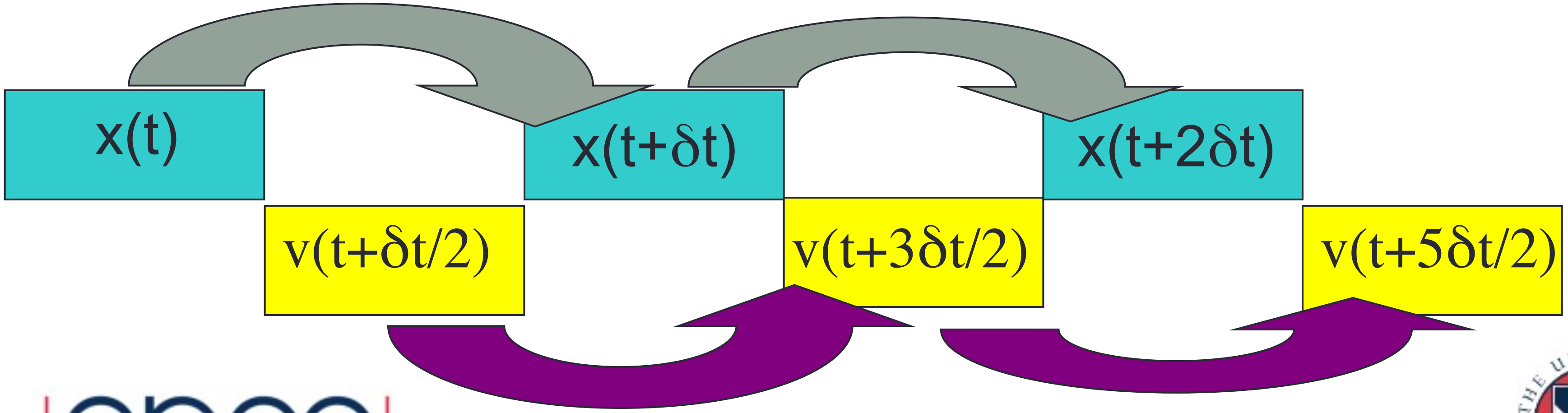


Leapfrog – second order scheme

- Error for each step is $\mathcal{O}((\delta t)^3)$

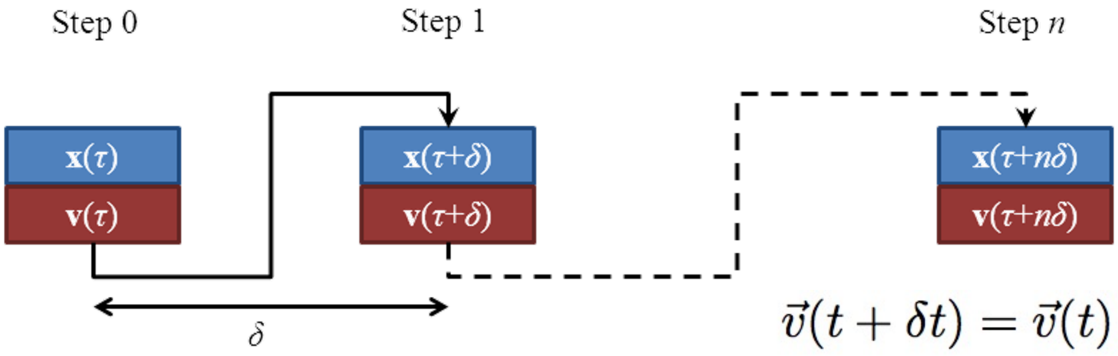
N steps of size δt → Global error is $O((\delta t)^2)$

- **time-reversal symmetric** (like Verlet)
- **more accurate** than Euler (and than Verlet for positions)



Time discretisation / integration schemes

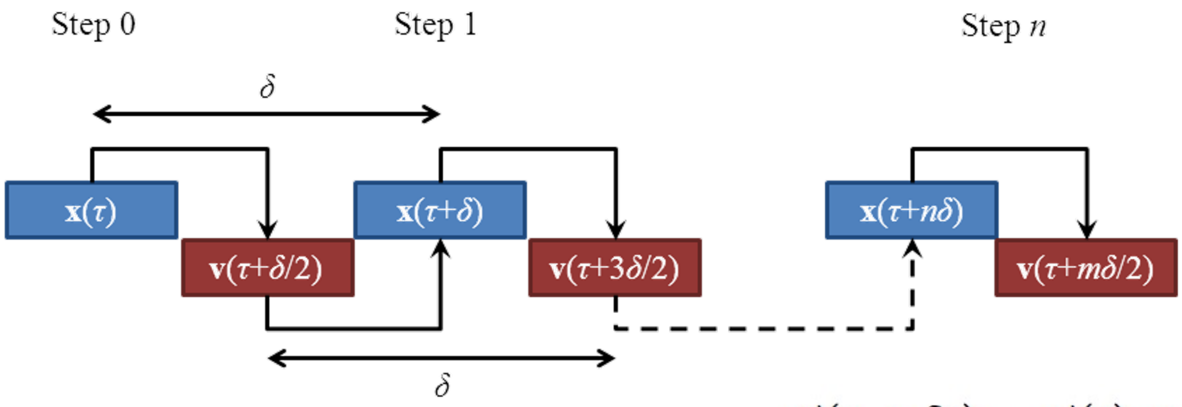
- Euler scheme


Step 0
Step 1
Step n
$\mathbf{x}(\tau)$
$\mathbf{v}(\tau)$
$\mathbf{x}(\tau+\delta)$
$\mathbf{v}(\tau+\delta)$
$\mathbf{x}(\tau+n\delta)$
$\mathbf{v}(\tau+n\delta)$
δ

$$\vec{v}(t+\delta t) = \vec{v}(t) + \vec{a}(t) \times \delta t$$

$$\vec{x}(t+\delta t) = \vec{x}(t) + \vec{v}(t) \times \delta t$$

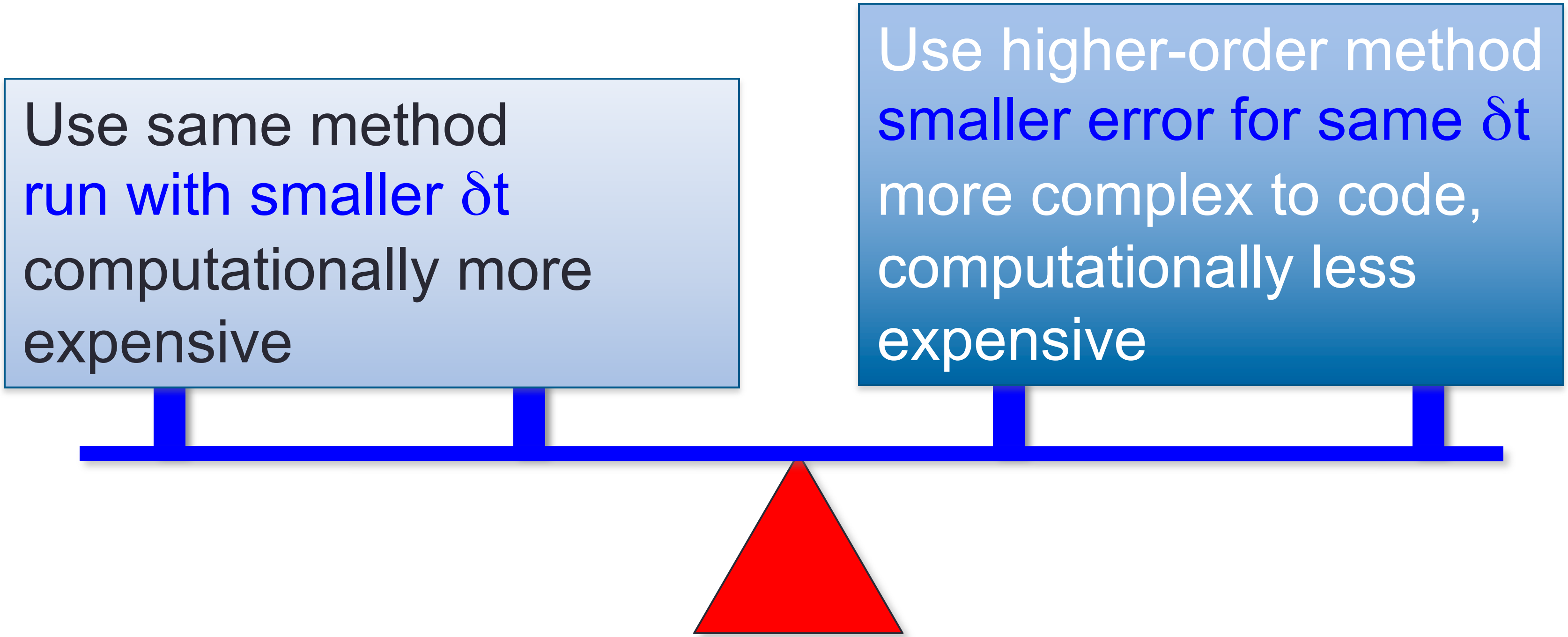
- Leapfrog scheme


Step 0
Step 1
Step n
δ
$\mathbf{x}(\tau)$
$\mathbf{v}(\tau+\delta/2)$
$\mathbf{x}(\tau+\delta)$
$\mathbf{v}(\tau+3\delta/2)$
$\mathbf{x}(\tau+n\delta)$
$\mathbf{v}(\tau+m\delta/2)$
δ

$$\vec{x}(t+\delta t) = \vec{x}(t) + \vec{v}(t + {}^{1}\!/_{2}\delta t) \times \delta t$$

$$\vec{v}(t + {}^{1}\!/_{2}\delta t) = \vec{v}(t - {}^{1}\!/_{2}\delta t) + \vec{a}(t) \times \delta t$$

Reducing the error



Use same method
run with smaller δt
computationally more expensive

Use higher-order method
smaller error for same δt
more complex to code, computationally less expensive

- Huge variety of schemes exist
  - some are adaptive and vary Δt automatically
  - beyond the scope of this course
  - Earth's 2nd moon calculations used high order integrator scheme

Checking correctness of solution

- This is a real issue!
- May be able to monitor special quantities
  - (average) energy & momentum should be conserved
    - might fluctuate over time for some types of simulations, but should never consistently increase or decrease
- Qualitative answer should not depend on step size
  - Multiple simulations to confirm answer
- Stability:
  - Physical quantities (distances, forces, energies) should not diverge unrealistically (“blow up”) – a clear sign that something is going wrong

Checks & sums

- Kinetic energy: $$E_K(t) = \frac{1}{2}\sum_i m_i \vec{v}_i(t)^2$$
- Potential energy: $$E_P(t) = \sum_{\langle i,j\rangle} U(|\vec{r}_j(t) - \vec{r}_i(t)|)$$
  - ($\sum_{\langle i,j\rangle}$ means sum over all pairs)
- Total energy: $E_T = E_P(t) + E_K(t) = C$
- Momentum: $$\vec{P} = \sum_i m_i \vec{v}_i(t)$$

(average) energy & momentum should be conserved

Force calculation schemes

Avoiding O(N^2)

Forces & potentials

- Potential = mathematical function describing potential energy associated with (a) position(s) or configuration(s)
  - Pairwise (two-body) potential: consider energy due to relative positions / configurations of pairs of particles

- Force = derivative of potential: $F = -\frac{dV}{dr}$
  - e.g. for gravitational pairwise force:

$$V(r_{ij}) = G\frac{m_i m_j}{r_{ij}}$$

$$F(r_{ij}) = -G\frac{m_i m_j}{r_{ij}^2}$$

Schemes for calculating N-body forces

The direct (particle-particle):

- Brute force approach to solving the N-Body problem
  - calculate at each time step the force on each particle as the sum of the forces from all other particles
- Minor simplification: forces are all equal and opposite, and do not need to be calculated twice:

```
For i = 1 to N
  Force[i]=0
  For j = i+1 to N
    f = force(i,j)
    Force[i] += f
    Force[j] -= f
```

$O(N^2)$

Schemes for calculating N-body forces

- Can we do better than $O(N^2)$?
  - Yes, how much better depends on the force
  - Distinguish between short range forces and long range forces

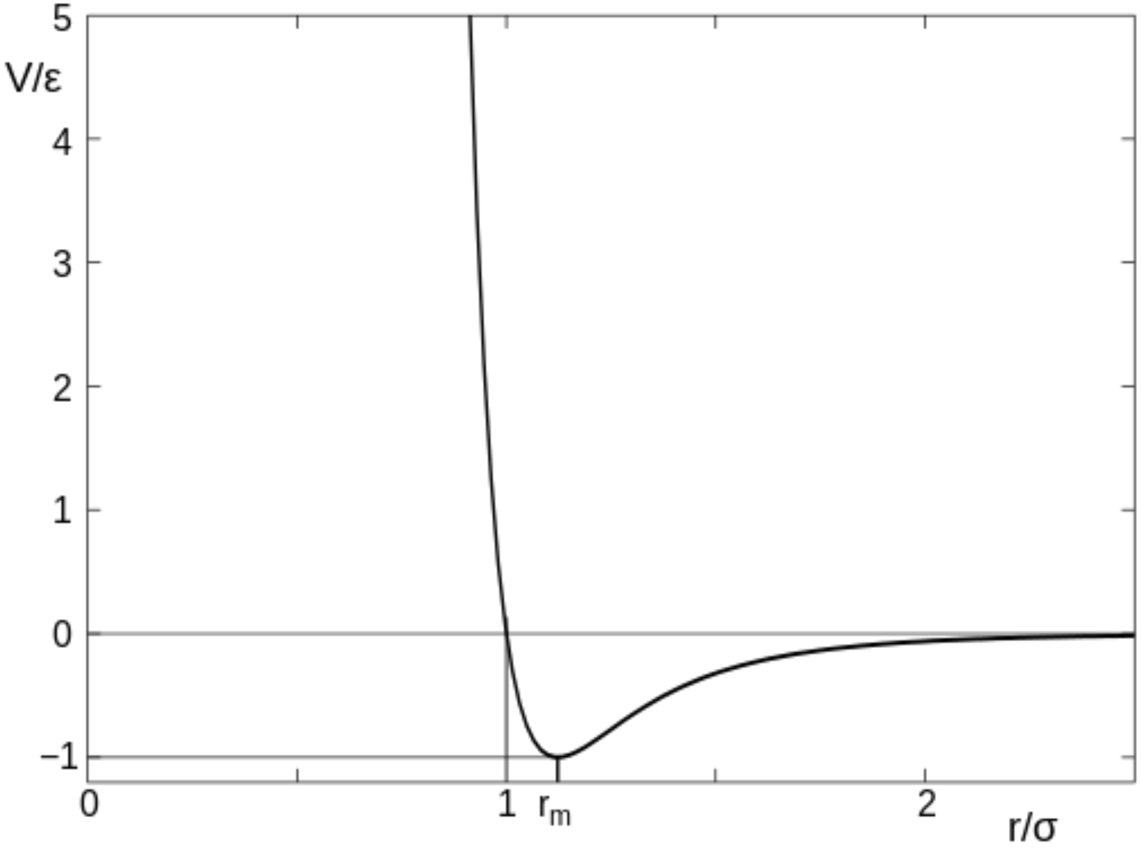
- Short-range forces: potential decreases faster than $1/r^3$
  - (energy contribution per unit volume goes to zero for $r \rightarrow \infty$)

- Long-range forces: potential decreases slower than $1/r^3$
  - (energy contribution per unit volume does **not** go to zero for $r \rightarrow \infty$)

Common forces / potentials

- Short-range interactions: *e.g.* Lennard-Jones (hard-core atomic repulsion):

$$V(r_{ij}) = 4\varepsilon_{ij}\left(\left[\frac{\sigma_{ij}}{r_{ij}}\right]^{12} - \left[\frac{\sigma_{ij}}{r_{ij}}\right]^{6}\right)$$



Common forces / potentials

- Short-range interactions:

  e.g. Lennard-Jones

$$V(r_{ij}) = 4\varepsilon_{ij}\left(\left[\frac{\sigma_{ij}}{r_{ij}}\right]^{12} - \left[\frac{\sigma_{ij}}{r_{ij}}\right]^{6}\right)$$

- Long-range interactions:

  e.g. Coulombic

  (electrostatic attraction similar to gravity)

$$V(r_{ij}) = \frac{q_i q_j}{r_{ij}}$$

- Intramolecular: bonds, angles, torsions, e.g. harmonic potential

  (vibrations)

$$V(r_{ij}) = \frac{1}{2}k\left(r_{ij} - r_{ij,0}\right)^2$$

Short-range forces - neighbour lists

- For short-range potentials, can choose a cutoff distance
  - Only consider interactions between particles within cutoff distance
  - Cutoff distance controls balance between accuracy and performance

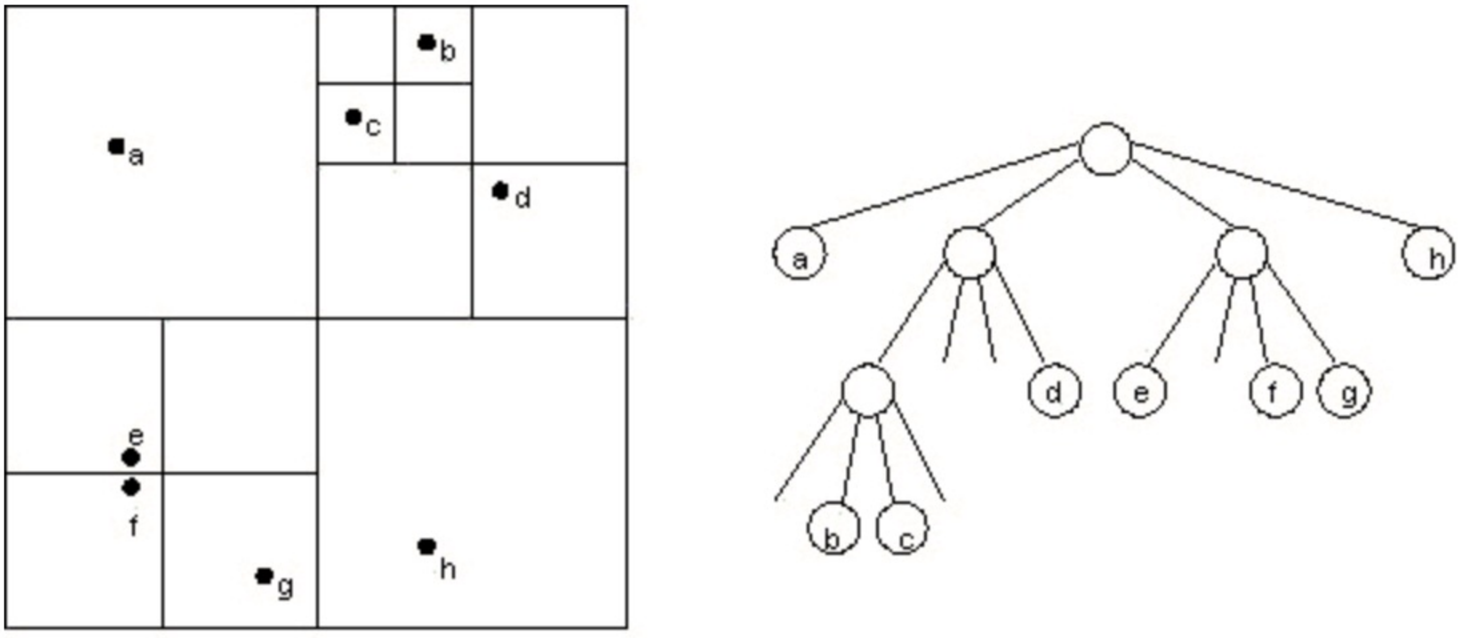
- Need an efficient way to keep track and quickly access positions of particles within each other's cutoff distance
  - Create & maintain neighbour lists of short-range interacting particles
  - List change in simulation, but may not need to update every time step!
  - Can construct neighbour lists in O(N) using a grid search
  - Size of lists chosen carefully to optimise cache usage

Short-range forces - neighbour lists

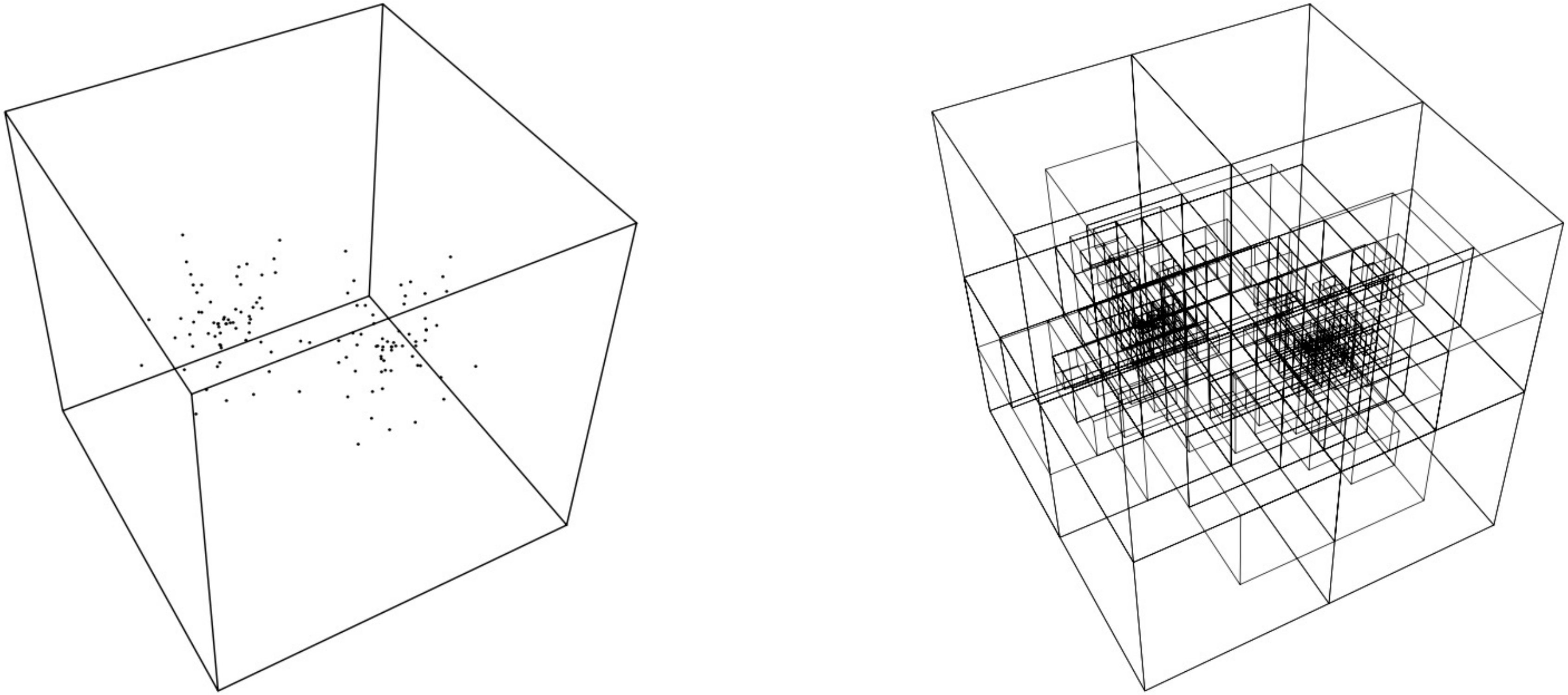
- Neighbour lists work well for problems with spatially balanced / bound particle density, e.g. molecular problems

- For astrophysical problems may have highly spatially variable particle density
  - Using a hierarchical method (see long-range forces) allows us to compute short-range forces with O(N log N) or O(N), depending on the method

Long-range forces – hierarchical methods



Long-range forces – hierarchical methods



Long-range forces – hierarchical methods

“Divide & Conquer”

- Divide the spatial domain and all particles in it into boxes each containing a single particle

- Build a hierarchical clustering representation of all particles in the simulation domain using a tree data structure
    - Each internal node of the tree stores a representation that can be used to compute interaction with all particles in its subtree

- Compute all forces on one particle by traversing tree, computing interactions with all clusters that satisfy an acceptance criterion
    – Acceptance criterion, e.g. (cluster size) / (distance from cluster), controls strictness of approximation ➔ trades off performance & accuracy

Long-range forces – Barnes Hut

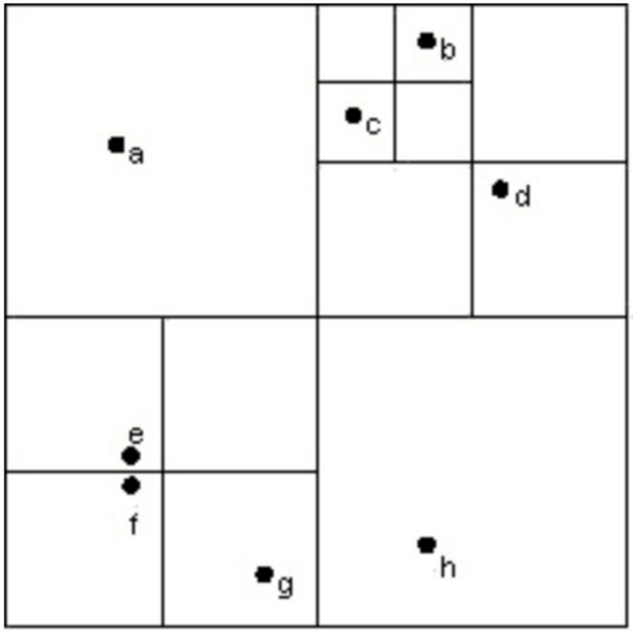
- Example hierarchical method: Barnes Hut algorithm
- Approximate interaction with all particles in a distant cluster as a single interaction with total cluster mass M located at cluster's centre of mass $\vec{R}$:

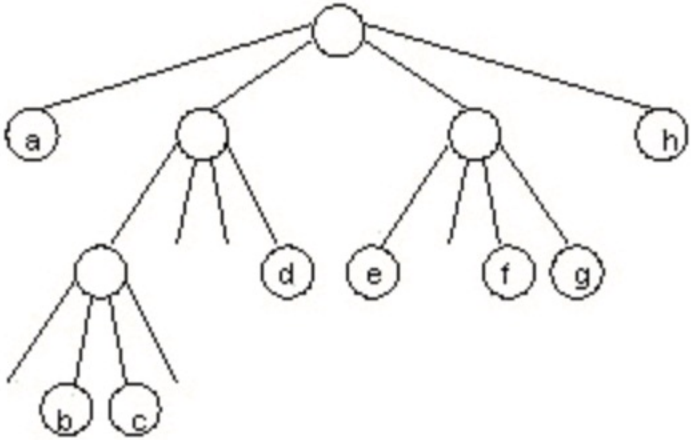
$$\vec{R} = \frac{1}{M}\sum_{i=1}^{N} m_i \vec{r}_i$$

$$M = \sum_{i=1}^{N} m_i$$

For "mass" can read "charge" (and centre of mass → centre of charge) if considering electrostatic attraction rather than gravity

Long-range forces – Barnes Hut (2D)





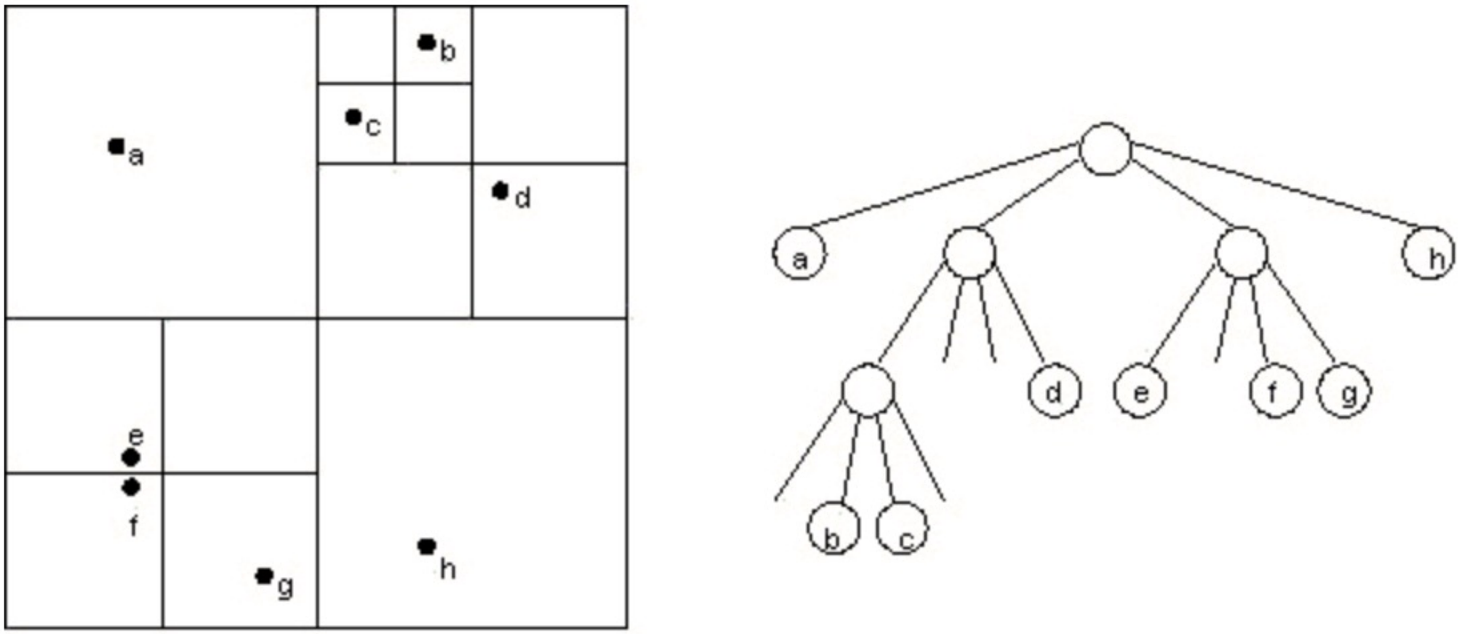
Long-range forces – Barnes Hut

Tree construction phase (2D):

- Starting with the root node of the tree (= all particles and the entire simulation domain), recursively subdivide the domain into 4 equal parts until each subdomain contains just one particle
- At each subdividing iteration *n*, associate each cluster of particles and its subdomain with a node in the tree at that depth (depth *n*)
- Traverse tree back up from leaves to root, computing and storing in each internal node the total mass (charge) and centre of mass (charge) of particles in its subtree

Constructing the tree takes O(N log N)

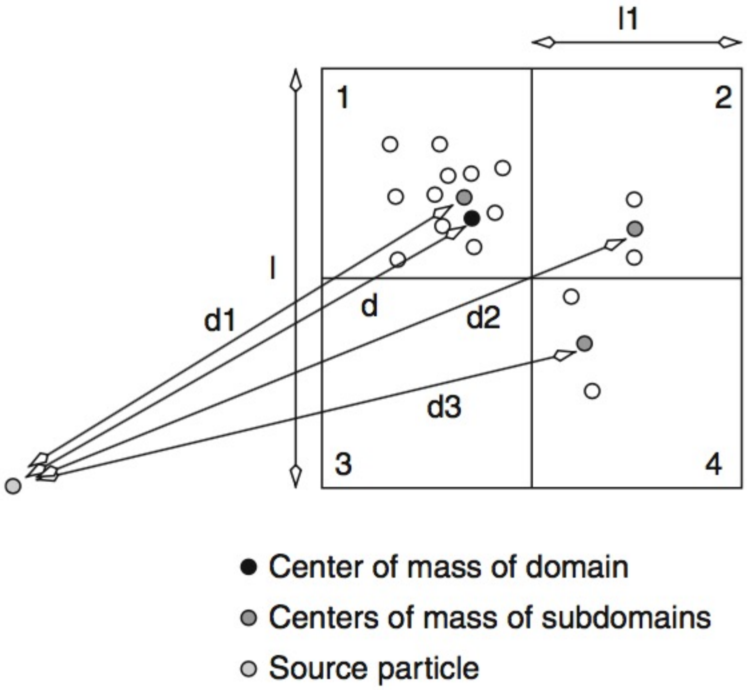
Long-range forces – Barnes Hut (2D)



Long-range forces – Barnes Hut (2D)

Force computation phase:

- To compute the interaction of one particle with all other particles, start at the root of the tree and traverse (depth first)
- At each level, if acceptance criterion met then compute force between particle and corresponding cluster, otherwise descend and repeat for its children



```
if (l/d < a)
          compute direct force interaction
          with the center of mass of domain,
else
          if (l1/d1 < a)
                    compute direct force computation
                    with center of mass of subdomain 1
          else
                    expand subdomain 1 further.

          Apply similar criteria to domains 2, 3, and 4.
```

Long-range forces – Barnes Hut

- For a balanced tree, each particle needs to compute O(log N) interactions, so overall force computation is O(N log N)
- However!: tree size can become arbitrarily large for a pair of particles arbitrarily close – arbitrarily large number of boxes needed to resolve the pair into separate boxes. worst-case complexity of this technique is unbounded
  - Solutions exist…
- Tree codes are fast but require auxiliary storage
- Will need to periodically rebuild tree
  - Optimised schemes exist to dynamically rebuild minimal partial subtrees

Long-range forces: fast multipole method (FMM)

- Other hierarchical methods exist,
  e.g. fast multipole method (FMM)

- FMM compared to Barnes Hut:
  - cluster–cluster interactions in addition to particle–cluster
  - series expansions of potentials instead of direct force calculations using centres of mass

- Computational complexity of FMM can be shown to be O(N) for “well-structured” particle distributions

Long-range forces: particle-mesh (PM)

- Represent particles (discrete, spatially localised) as a spatial density function $\rho(\vec{r})$ (continuous)
  - $\rho(\vec{r})$ actually only stored at discrete points on a spatial grid (mesh)

- Potential $\Phi(\vec{r})$ felt by all particles solved on mesh using FFTs:
- $\nabla^2\Phi(\vec{r}) = 4\pi G\rho(\vec{r})$ ← FFT → $\hat{\Phi}(\vec{k}) = -4\pi G\frac{\hat{\rho}}{k^2}$

- Computational complexity is O(G log G)
  - G = number of grid points
  - computation dominated by FFTs

Long-range forces: particle-mesh (PM)

- Fast, but phenomena at smaller scale than mesh spacing not modelled accurately

- Fine for large-scale / coarse-grained phenomena, but not for detailed (short-range) molecular dynamics

- Higher resolution:
  - Decrease grid spacing L
  - Increases number of grid points cubically: $G \propto 1/L^3$
  - Rapidly increasing cost of FFTs and hence overall complexity

Long-range forces: particle-particle/particle-mesh (P^{3}M)

- Combine:
  - Particle-particle calculations
    - accurately resolve phenomena on small spatial scale
  - Mesh-based calculation
    - for long-range interactions
- Choose crossover range to switch between approaches carefully to avoid O(N^2) direct particle-particle force calculation component from dominating
- Complexity is O(N + G), can approach O(N)
- For periodic systems, known as Particle-Mesh Ewald (PME) – used in many molecular dynamics algorithms

N-body force “fixes”

(computational scientific ‘hacks’)

Particle radii – avoiding instability

- Positions of point particles can in principle coincide

- Forces often include terms of the form ($1/r^q$), where $q>0$: these have a singularity in the force as particles get very close to each other

- Large forces are calculated as if they last an entire timestep, easily resulting in an unrealistically close approach / overlap between particles & overestimation of the force:
  - Can lead to instability – “explosions” of particle energy / velocity

- Overcome with very small time-step, but then need to recalculate all forces much more often

Particle radii – avoiding instability

- Smaller timesteps and multi-step methods can be used, but ultimately steep nature of the potential means there is a limit to how accurately close encounters can be calculated

- Can use “hack” to give particles an effective non-zero radius to maintain stability:
  - Treat particles as point masses to calculate force as usual if separation distance is greater than their radius
  - Set potential to zero if distance is less than this radius
  - Very unrealistic!

Is there a better alternative?

Particle radii – avoiding instability

- Modify potential functions to include a “softened” distance:
  - Treat all distances as if they were some small distance *ε* greater than the real separation *r :*

$$P = -G\frac{Mm}{r} \rightarrow P = -G\frac{Mm}{(r^2+\epsilon^2)^{1/2}}$$

$$\vec{F}_i = G\frac{M_i m_j(\vec{x}_i - \vec{x}_j)}{x_i - x_j}$$

- Avoids instability and introduces a smaller error than the step-change “effective finite radius” hack

Summary

- Time discretisation / integration schemes
  - Understanding accuracy (errors)
  - Checking correctness & stability
- Force calculation schemes
  - Direct (brute force)
  - Neighbour lists
  - Hierarchical / tree-based
  - Particle-mesh, P^{3}M
  - Periodic boundary conditions
- N-body force “fixes”