Time integration methods

Basically an ODE integration problem, but some special requirements lead us to prefer particular solvers:

- We require at least $O(\Delta t^2)$ accuracy.
- If N is large, we would like to avoid implicit methods.
- We would also like to avoid methods that involve multiple evaluations of the acceleration at each step (since these are expensive).
- If we must integrate for a large number of timesteps, our method should explicitly conserve energy, or else the accumulation of roundoff error will make the solution meaningless.

Leapfrog (Størmer-Verlet) method

The *leapfrog integration method* is commonly used in particle simulation. In this method, positions and velocities are staggered in time (hence the name).

$$\mathbf{v}_i^{n+1/2} = \mathbf{v}_i^{n-1/2} + \frac{1}{m_i}\mathbf{F}\left(\mathbf{x}_i^n\right)\Delta t$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{v}_i^{n+1/2}\,\Delta t$$

This method has $O(\Delta t^2)$ truncation error: if we eliminate **v** from the above equations, we have (in 1D, dropping the particle index i)

$$\frac{x^{n+1} - 2\,x^n + x^{n-1}}{\Delta t^2} = \frac{F\left(x^n\right)}{m}$$

The true solution X satisfies

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

The truncation error δ is obtained when we substitute X into the update method and use Taylor expansions about X^n to substitute for X^{n+1} and X^{n-1}:

$$\frac{X^{n+1} - 2\,X^n + X^{n-1}}{\Delta t^2} = \frac{F\left(X^n\right)}{m} - \delta^n$$

$$\frac{d^2 X^n}{dt^2} + \frac{\Delta t^2}{12}\frac{d^4 X^n}{dt^4} + \ldots = \frac{F\left(X^n\right)}{m} - \delta^n \quad \Rightarrow \quad \delta^n \propto \Delta t^2$$

Predictor-corrector methods

Leapfrog is very easy to code and works well, but requires for stability

$$\Delta t \leq 2\left[\frac{1}{m}\left|\frac{\partial F}{\partial x}\right|_{\max}\right]^{-1/2}$$

Predictor-corrector methods give us some of the stability benefit of an implicit method without requiring us to invert a matrix, at the cost of having to store previous-timestep data. Also, they let us go to higher order without requiring more and more intermediate evaluations of the acceleration **a**.

Procedure:

1. Using previous timestep values $\mathbf{a}^{n}$, $\mathbf{a}^{n-1}$, ..., use polynomial extrapolation in time to estimate $\mathbf{x}^{n+1}$ (*predictor step*).

2. Using the predicted new value of **x**, estimate $\mathbf{a}^{n+1}$ (*evaluation step*).

3. Substitute this acceleration into the extrapolation formula again to obtain an improved estimate of $\mathbf{x}^{n+1}$ (*corrector step*).

4. Using the corrected new value of **x**, recompute $\mathbf{a}^{n+1}$ for use in the next predictor step (*evaluation step*).

Predictor-corrector methods – 2

The stability properties of predictor-corrector methods are better than explicit methods, but not as good as implicit methods.

For a *k*th-order method we need to store $\mathbf{a}^n$, $\mathbf{a}^{n-1}$, ..., $\mathbf{a}^{n-k+2}$ for each particle.

Second-order method:

$$\left.\begin{aligned}\mathbf{x}^* &= \mathbf{x}^n + \mathbf{v}^n \Delta t + \frac{1}{2}\Delta t^2 \mathbf{a}^n \\ \mathbf{v}^* &= \mathbf{v}^n + \mathbf{a}^n \Delta t\end{aligned}\right\} \text{ predictor step}$$

$$\left.\begin{aligned}\mathbf{x}^{n+1} &= \mathbf{x}^n + \mathbf{v}^n \Delta t + \frac{1}{2}\Delta t^2 \mathbf{a}^* \\ \mathbf{v}^{n+1} &= \mathbf{v}^n + \mathbf{a}^* \Delta t\end{aligned}\right\} \text{ corrector step}$$

Third-order method:

$$\left.\begin{aligned}\mathbf{x}^* &= \mathbf{x}^n + \mathbf{v}^n \Delta t + \left[\frac{2}{3}\mathbf{a}^n - \frac{1}{6}\mathbf{a}^{n-1}\right]\Delta t^2 \\ \mathbf{v}^* &= \mathbf{v}^n + \left[\frac{3}{2}\mathbf{a}^n - \frac{1}{2}\mathbf{a}^{n-1}\right]\Delta t\end{aligned}\right\} \text{ predictor step}$$

$$\left.\begin{aligned}\mathbf{x}^{n+1} &= \mathbf{x}^n + \mathbf{v}^n \Delta t + \left[\frac{1}{6}\mathbf{a}^* + \frac{1}{3}\mathbf{a}^n\right]\Delta t^2 \\ \mathbf{v}^{n+1} &= \mathbf{v}^n + \left[\frac{1}{2}\mathbf{a}^* + \frac{1}{2}\mathbf{a}^n\right]\Delta t\end{aligned}\right\} \text{ corrector step}$$

Symplectic methods

The equations of motion are *Hamiltonian*, ie.,

$$\frac{d\,\mathbf{q}}{dt} = \nabla_{\mathbf{p}}\,\mathcal{H}\,(\mathbf{q}\,,\,\mathbf{p}) \qquad\qquad \frac{d\,\mathbf{p}}{dt} = -\nabla_{\mathbf{q}}\,\mathcal{H}\,(\mathbf{q}\,,\,\mathbf{p})$$

Thus there are certain integrals of the motion that solutions should preserve (e.g., energy). We would like to build this into an integration method.

Construct the vector $\mathbf{z} \equiv \begin{pmatrix} \mathbf{q} & \mathbf{p} \end{pmatrix}^T$. Then we have

$$\frac{d\,\mathbf{z}}{dt} = \boldsymbol{J} \cdot \nabla_{\mathbf{z}}\,\mathcal{H}\,(\mathbf{z})$$

where $\boldsymbol{J}$ is the *symplectic matrix*

$$\boldsymbol{J} \equiv \begin{pmatrix} \mathbf{0} & \boldsymbol{I} \\ -\boldsymbol{I} & \mathbf{0} \end{pmatrix}$$

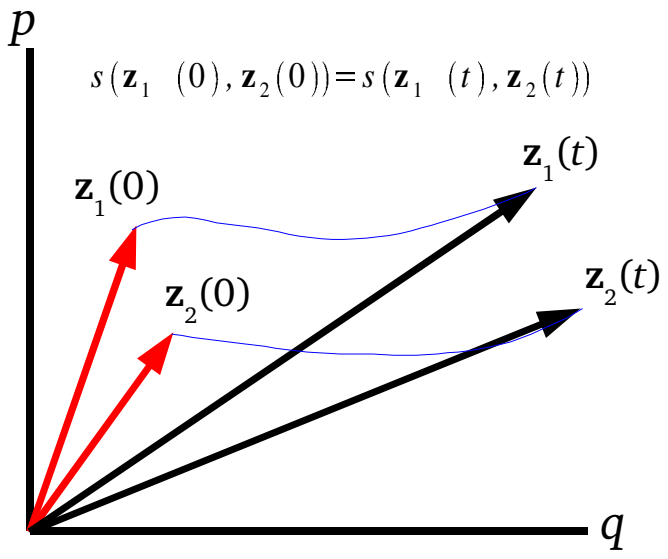
Now, evolution of $\mathbf{z}(t)$ from some initial conditions $\mathbf{z}_0(t)$ (a *canonical transformation*) preserves the energy *and* the *symplectic form*

$$s\,(\mathbf{z}_1\ \ ,\mathbf{z}_2) \equiv \mathbf{z}_1^T\ \ \cdot \boldsymbol{J} \cdot \mathbf{z}_2$$

If the Hamiltonian is non-integrable, either the energy or the symplectic form may be conserved, but not both.

Symplectic methods – 2

It turns out that leapfrog conserves the symplectic form, while methods such as Runge-Kutta and predictor-corrector do not.



A fourth-order symplectic method is *Candy's method*: let the Hamiltonian be separable, so that

$$\mathcal{H}(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + T(\mathbf{p})$$

Define $\mathbf{F}(\mathbf{q}) \equiv -\nabla_{\mathbf{q}} U(\mathbf{q}) \qquad \mathbf{P}(\mathbf{q}) \equiv \nabla_{\mathbf{p}} T(\mathbf{p})$

and start with the state $(\mathbf{q}_0, \mathbf{p}_0)$ at time t_n. Then

$$\mathbf{p}_i = \mathbf{p}_{i-1} + b_i \mathbf{F}(\mathbf{q}_{i-1}) \Delta t$$
$$\mathbf{q}_i = \mathbf{q}_{i-1} + a_i \mathbf{P}(\mathbf{p}_i) \Delta t \qquad i = 1 \ldots 4$$

where

$$a_1 = a_4 = (2 + 2^{1/3} + 2^{-1/3})/6$$
$$a_2 = a_3 = (1 - 2^{1/3} - 2^{-1/3})/6$$

$$b_1 = 0$$
$$b_2 = b_4 = 1/(2 - 2^{1/3})$$
$$b_3 = 1/(1 - 2^{2/3})$$

The state $(\mathbf{q}_4, \mathbf{p}_4)$ is the updated state (time t_{n+1}).

Particle-particle (PP) or direct *N*-body

Simplest possible *N*-body technique.

1. For each particle i, compute $\mathbf{a}_i = -G \sum_{\substack{j=1 \\ j \neq i}}^{N} m_j \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|^3}$

2. For each particle, integrate $\frac{d(\mathbf{x}_i, \mathbf{v}_i)}{dt} = (\mathbf{v}_i, \mathbf{a}_i)$ from t_n to $t_n + \Delta t$.

3. Repeat.

Advantages:

- Force law is exact, so for applications in which collisions are important (e.g., globular clusters), this may be the best method.
- No artificial restrictions on geometry or boundary conditions.

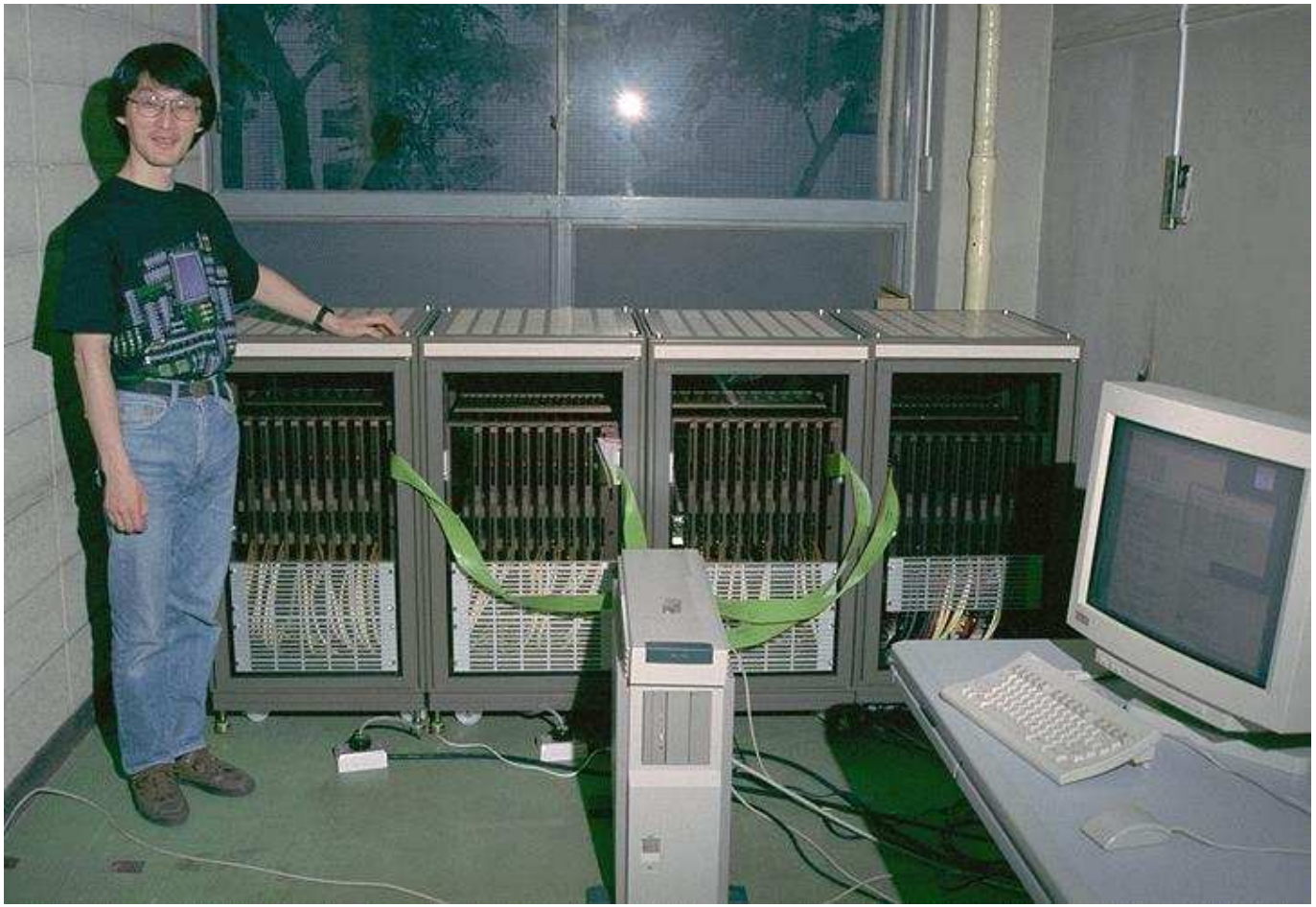
Disadvantage:

- Force computation scales as N^2 – very expensive for even modest N.

PP is usually combined with individual particle timesteps and special treatment of close encounters and binary stars.

Particle-particle – 2

GRAPE (GRAvity PipE) special-purpose force-summation computer
J. Makino et al. – University of Tokyo



Particle-particle – 3

Gravothermal oscillations first observed in *N*-body simulations of post-core-collapse globular clusters – J. Makino (1996), using GRAPE-4

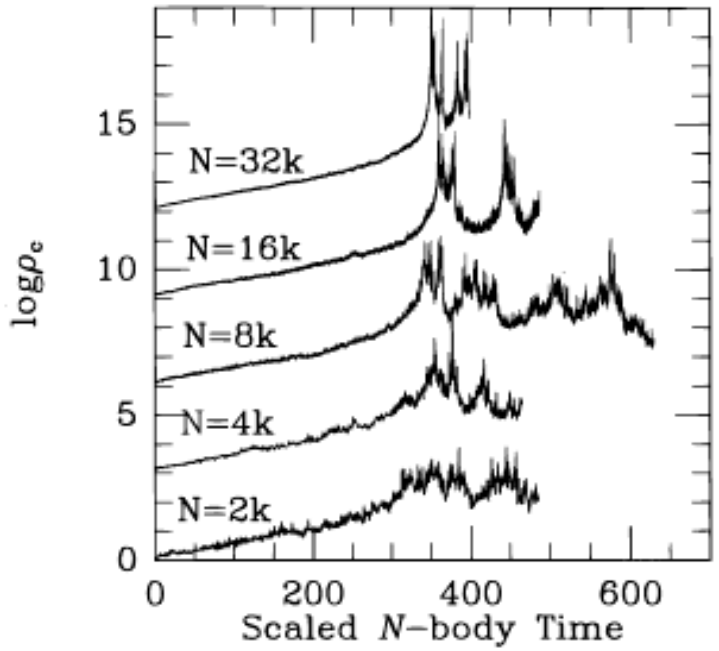


FIG. 1.—Logarithm of the central density plotted as a function of the scaled *N*-body time. Curves for different values of *N* are vertically shifted by 3 units.

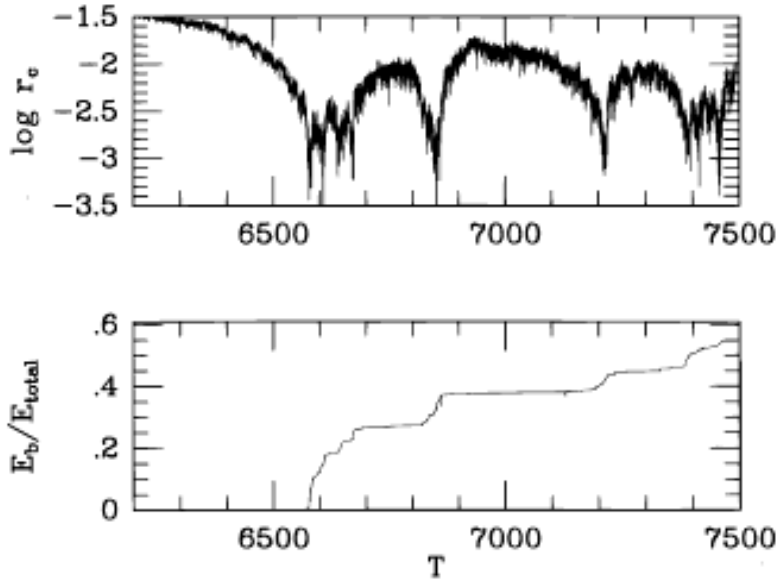


FIG. 2.—Core radius (*top*) and binding energy of binaries (*bottom*) as a function of time for the 32K particle run.

Particle-mesh (PM)

If our problem is collisionless, we can achieve a dramatic speedup over direct *N*-body by exploiting fast mesh-based Poisson solvers, which scale as $N_g \ln N_g$.

First introduced for plasma simulation in the early 1960s.

Procedure:

1. Assign each particle's mass to a grid using a *mass assignment operator*. The sum of the contributions of all of the particles will be a gridded density field ρ_{ij}.

2. Using a mesh-based Poisson solver, solve $\nabla^2\phi = 4\pi G\rho$.

3. Compute the gravitational force on the mesh by finite-differencing the mesh potential.

4. Interpolate forces to the particle positions using a *force interpolation operator*.

5. Advance the positions and velocities of the particles in time.

6. Repeat.

Particle-mesh – 2

A mass assignment operator takes a particle's position and uses it to assign masses to one or more nearby mesh points. In 1D we have for N_p particles

$$\rho_i = \frac{m}{\Delta x} \sum_{p=1}^{N_p} W(x_i - x_p)$$

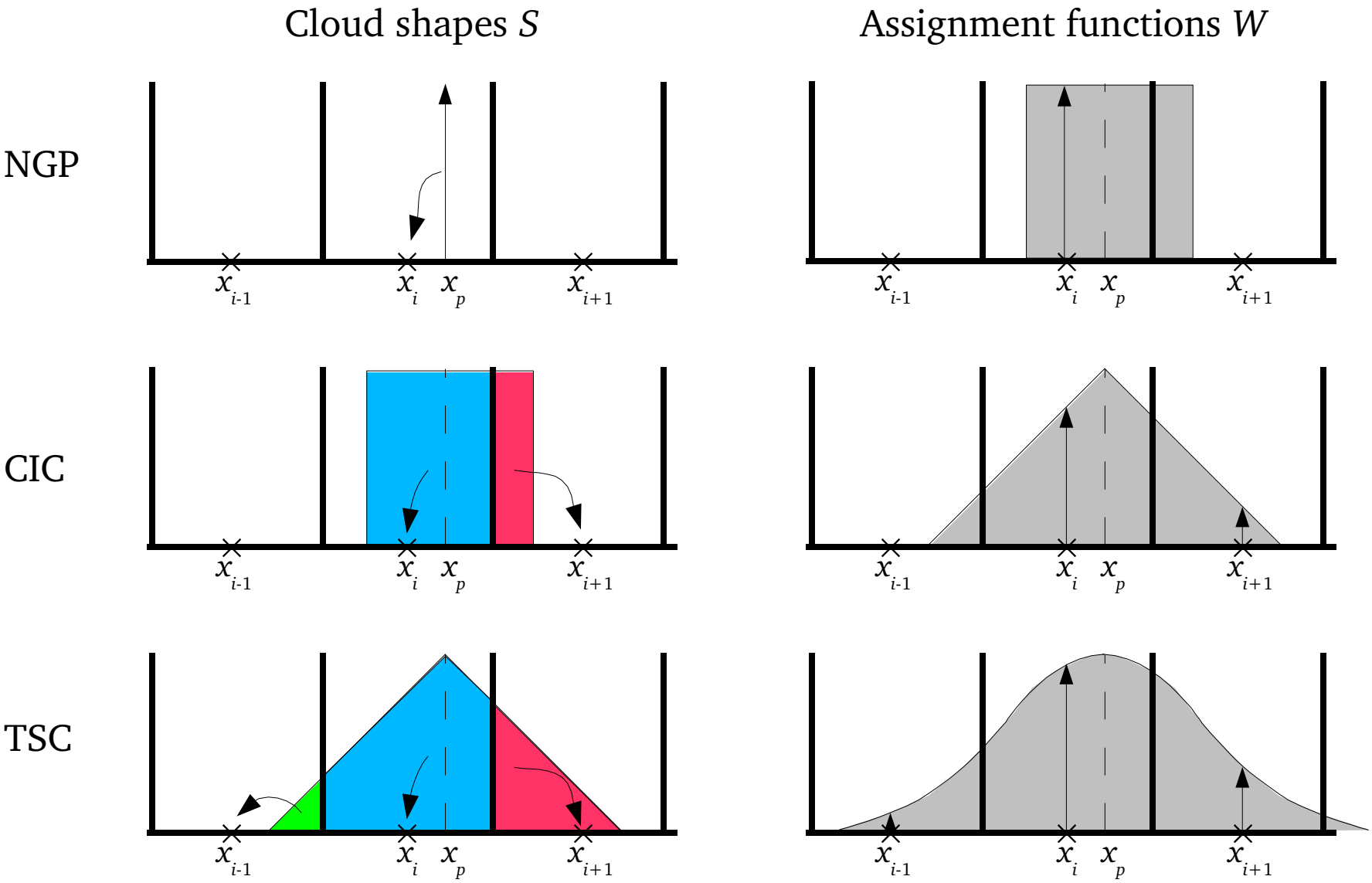
Three commonly used operators:

- Nearest grid point (NGP) $$W_{\mathrm{NGP}}(x) = \begin{cases} 1 & -\Delta x/2 < x < \Delta x/2 \\ 0 & \text{otherwise} \end{cases}$$
- Cloud-in-cell (CIC) $$W_{\mathrm{CIC}}(x) = \begin{cases} 1 + x/\Delta x & -\Delta x < x \\ 1 - x/\Delta x & x < \Delta x \\ 0 & \text{otherwise} \end{cases}$$
- Triangle-shaped cloud (TSC) $$W_{\mathrm{TSC}}(x) = W_{\mathrm{CIC}}(x) * W_{\mathrm{NGP}}(x)$$

The same operators can be used in reverse to interpolate forces from the grid onto particle positions:

$$F(x_p) = -m \sum_{i=1}^{N_g} \left(\frac{d\phi}{dx} \right)_i W(x_p - x_i)$$

Particle-mesh – 3



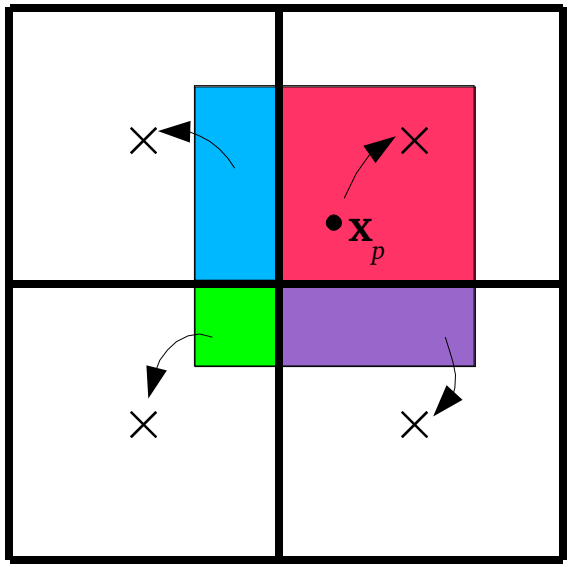
The assignment function is the convolution of the cloud shape with a top hat.

Particle-mesh – 4

In more than one dimension we typically use products of 1D weighting functions:

$$\rho_{ijk} = \frac{m}{\Delta x \, \Delta y \, \Delta z} \sum_{p=1}^{N_p} W(x_i - x_p) W(y_j - y_p) W(z_k - z_p)$$

These correspond to “particle clouds” that are effectively square/cubic: e.g., for CIC,



It is possible to use other cloud shapes (non-product weighting functions), but the product forms are generally used because

- they are cheap to compute
- their error properties are easy to determine analytically

Particle-mesh – 5

The force felt by particles deviates from the Newtonian law at separations smaller than about 2-3 zone widths.

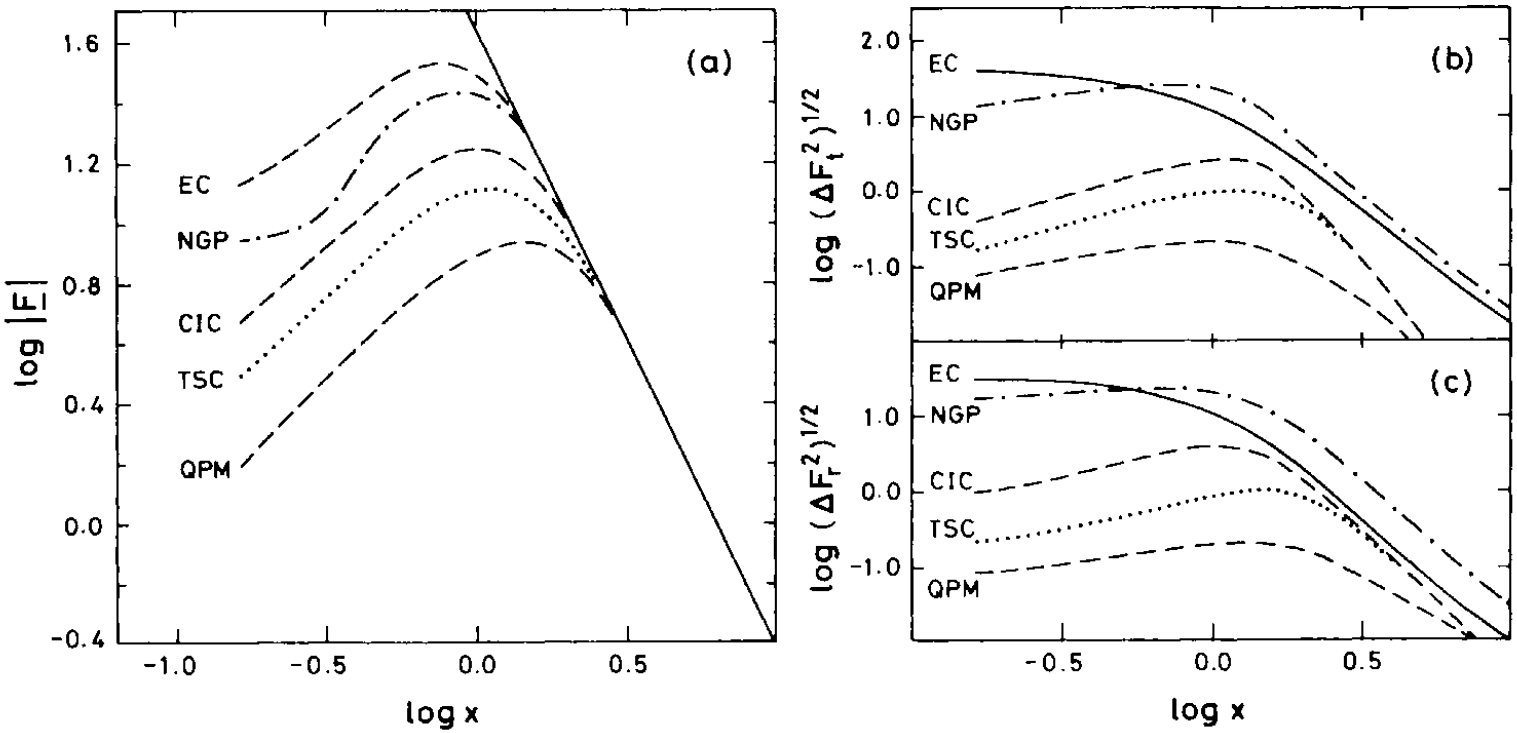


Figure 2 (*a*) The mean interparticle force as a function of separation in mesh spaces for several particle-mesh schemes. The solid line in (*a*) shows the unsoftened force. The other panels show the rms fluctuations in the tangential (*b*) and radial (*c*) directions [reproduced from Efstathiou et al. (1985), with permission].

However, this is just the sort of force smoothing we need to prevent artificial two-body relaxation!