Elliptical systems & N-body simulations

Lecture 14, Computational Astrophysics (ASTR660)

Hsiang-Yi Karen Yang, NTHU, 12/15/2022

Announcements

- ***HW6 will be posted on eLearn TODAY . Due at 14:20 next week (12/22/2022).*** Late submission within one week will receive ***75%*** of the credit
- The final ***oral presentations*** are scheduled on ***12/29*** and ***1/5***. The schedule has been posted on eLearn. Please let me know ASAP if you have any conflicts (you could switch time with someone else if they agree)
- The final ***product*** of the project is due on ***1/12***. Please start early and try to finish the final project on time!

Date	Time	Name	Title
12/29/22	14:20-14:40	吳耕緯	3D N-body with fast multipole method
	14:40-15:00	李佳倫	Effects of AGN Quasar Feedback in Galaxy Clusters
	15:00-15:20	簡嘉成	Improving my magnetohydrodynamic simulation for non-ideal plasma fluid
	15:30-15:50	凌志騰	A gravitational SPH simulation of galaxy collision from scratch
	15:50-16:10	鄭文淇	Simulate impact-driven atmospheric loss using N-body simulation
	16:20-16:40	龔一桓	Ram pressure striping of galaxy cluster: Can a subcluster be assumed as gasless
	16:40-17:00	劉一璠	Simulate the Kelvin-Helmholtz instability by using FLASH code
1/5/23	14:20-14:40	胡英祈	Modeling Dust Polarization in Protoplanetary Disk at (Sub)millimeter Wavelengths
	14:40-15:00	周育如	Multi-gaussian fit, EW, and line ratio for [O I] 5577 and 6300 from DG Tau
	15:00-15:20	考司圖巴	Compare MCMC based fitting with phenomenological models for X-ray Spectra
	15:30-15:50	郭弈翔	Calculate the entropy of a subregion using Python computation
	15:50-16:10	謝明學	simulate the tidal locking phenomenon
	16:20-16:40	張修瑜	Three-body problem in the Moon, Earth, and the Sun.
	16:40-17:00	石郡翰	Using the Mechanism of AIRES to Simulate the Propagation of Cosmic Rays in the universe

Previous lecture...

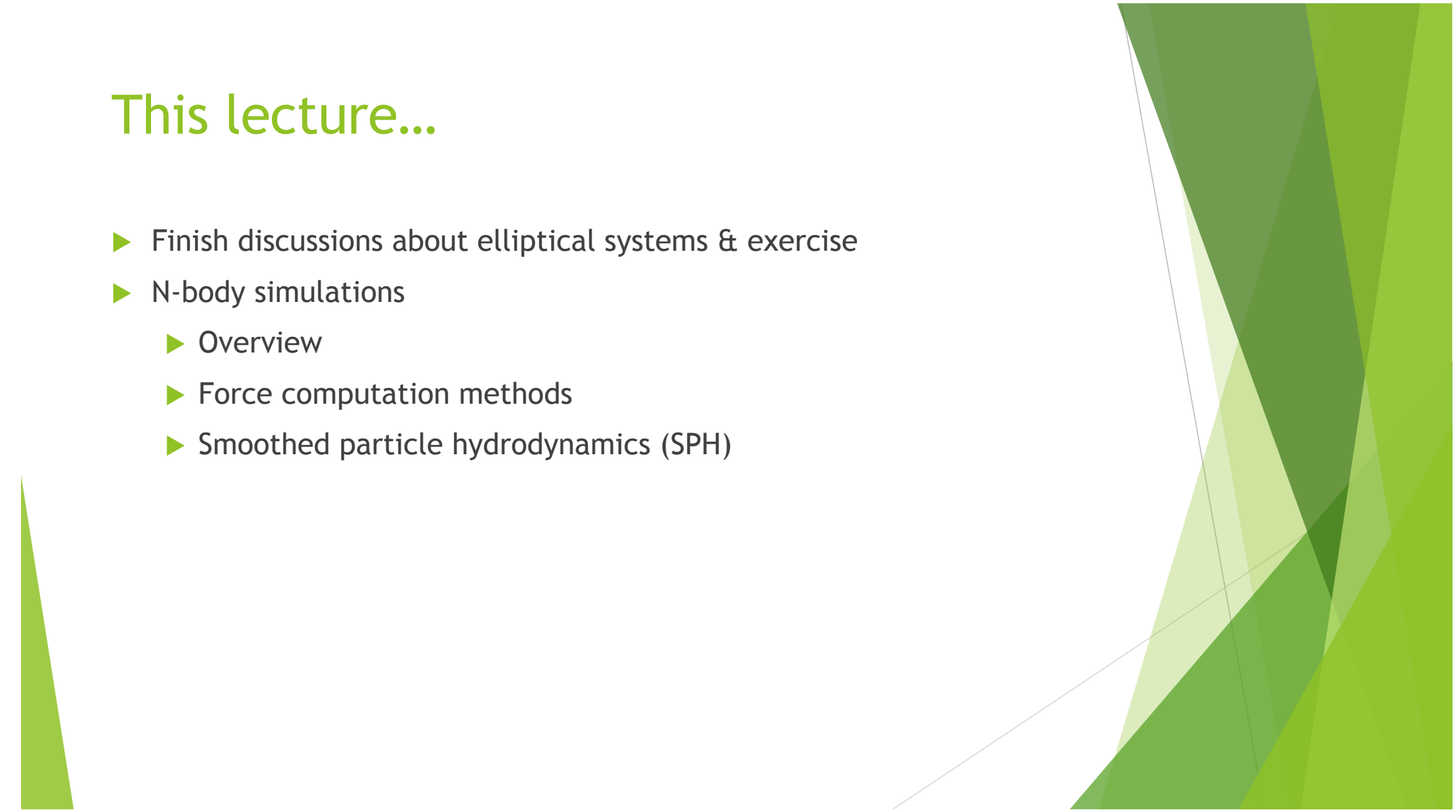
- ***MHD equations*** are used to describe non-relativistic, charge neutral, magnetized fluids
  - MHD equations = HD equations + ***induction equation*** for B + ***divergence-free*** constraint for B
  - B field exerts Lorentz force on the fluid, including magnetic tension and magnetic pressure forces
  - ***Plasma beta*** ($\beta \equiv \frac{P}{P_B} = \frac{8\pi P}{B^2}$) describes whether B field is dynamically important
- ***Ideal*** MHD equations apply to non-resistive, infinitely conducting magnetized fluids
  - Typically true for astrophysical systems (magnetic Reynolds number $Re_M \equiv \frac{Lu}{\eta} \gg 1$)
  - ***Flux-freezing***: magnetic flux passing through a surface moving with the fluid is conserved
  - Three characteristics for linear perturbations: Alfven waves, fast waves, and slow waves

Previous lecture...

- MHD equations can be solved using finite-volume methods similar to pure hydro
- Numerical methods to ensure div(B) = 0 include
  1) using vector potential **A**
  2) applying artificial diffusion
  3) divergence cleaning
  4) constrained transport (CT) method
- Advanced grid refinement techniques for grid-based codes are developed to save computational costs. Commonly used techniques include:
  1) non-uniform meshes
  2) nested grids
  3) adaptive mesh refinement (AMR)
  4) Lagrangian/moving meshes

This lecture…

- Finish discussions about elliptical systems & exercise
- N-body simulations
  - Overview
  - Force computation methods
  - Smoothed particle hydrodynamics (SPH)

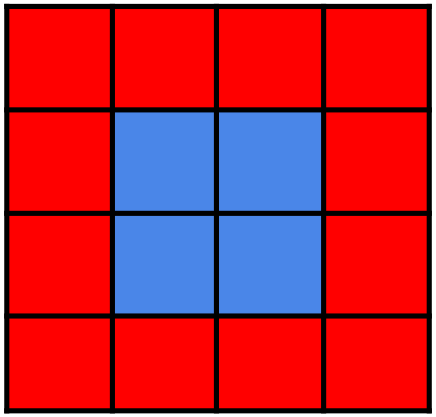


PDE – elliptical systems

Solving the Poisson equation

$$\nabla^2 \phi(\boldsymbol{r}) = \rho(\boldsymbol{r})$$

- ρ: mass density, ϕ: gravitational potential, assuming $4\pi G = 1$
- This is a BVP: given ρ in domain V and ϕ at the boundary, solve for ϕ in domain V
- This is typically solved ***iteratively***: start from an initial guess for solution, improve it until ***residual*** is as small as desired



Given ϕ

Given ρ, solve ϕ

Relaxation method

- Rewrite the Poisson equation: $L\phi = \rho \rightarrow \boxed{\frac{\partial \phi}{\partial t} = L\phi - \rho}$ *L*: elliptic/Laplacian operator
- **Step 1**: choose initial guess of ϕ
- **Step 2**: let the system ***relax until equilibrium is reached***. That is, when

$$\frac{\partial \phi}{\partial t} = 0 \rightarrow L\phi = \rho$$ *(solution to the original elliptic equation is found)*

- What we care about is only the final (relaxed) solution, so intermediate (unrelaxed) solution is allowed to have large errors
- Relaxation can be very time-consuming ($\propto N^2$) -> key to design an algorithm is to speed up the relaxation process

Relaxation method for Poisson equation

$$\nabla^2\phi = \rho \rightarrow \frac{\partial\phi}{\partial t} = \nabla^2\phi - \rho$$

- We can write down a 2D discrete form with FTCS scheme (see Lecture 10) assuming $\Delta x = \Delta y = \Delta$:

$$\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^{n}}{\Delta t} = \frac{1}{\Delta^2}\left(\phi_{i+1,j}^{n} + \phi_{i-1,j}^{n} + \phi_{i,j+1}^{n} + \phi_{i,j-1}^{n} - 4\phi_{i,j}^{n}\right) - \rho_{i,j}$$

- CFL stability criterion requires: $\Delta t \leq \Delta^2/4 \rightarrow$ let $\Delta t = \Delta^2/4$
- Then we have the ***Jacobi method***:

$$\phi_{i,j}^{n+1} = \frac{1}{4}\left(\phi_{i+1,j}^{n} + \phi_{i-1,j}^{n} + \phi_{i,j+1}^{n} + \phi_{i,j-1}^{n} - \Delta^2\rho_{i,j}\right)$$

iterate until relaxation/equilibrium/convergence is reached

Relaxation method #2 – Gauss-Seidel (GS) method

- The Jacobi method converges very slowly (# iterations $\propto N^2$)
- Alternatively, Gauss-Seidel method uses "in-place" updates:

$$\phi_{i,j}^{n+1} = \frac{1}{4}\left(\phi_{i+1,j}^{n/n+1} + \phi_{i-1,j}^{n/n+1} + \phi_{i,j+1}^{n/n+1} + \phi_{i,j-1}^{n/n+1} - \Delta^2 \rho_{i,j}\right)$$

use the updated values (n+1) instead of the original values (n) whenever available

- This method converges faster than the Jacobi method
- But # iterations still $\propto N^2$

Relaxation method #3 – successive over-relaxation (SOR) method

$$\phi_{i,j}^{n+1} = \phi_{i,j}^{n} + \omega\left(\phi_{i,j,GS}^{n+1} - \phi_{i,j}^{n}\right) = (1-\omega)\phi_{i,j}^{n} + \omega\phi_{i,j,GS}^{n+1}$$

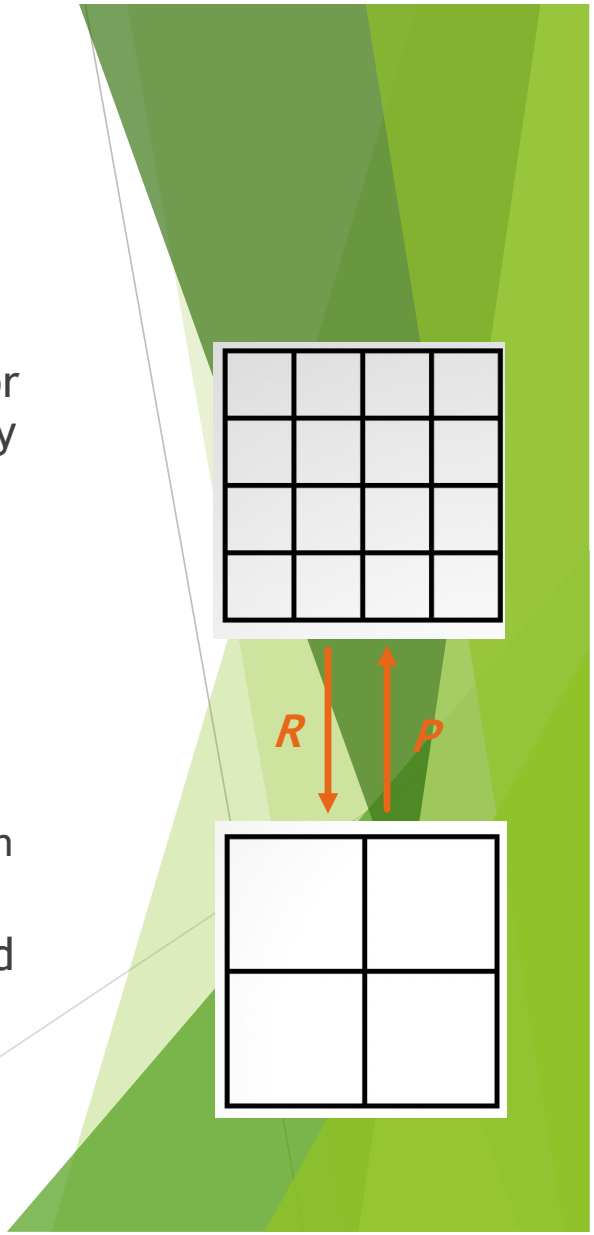
- ω is a fixed relaxation parameter chosen to accelerate convergence
  - **$\omega > 1$: over-relaxation** (as if CFL number > 1)
  - $\omega < 1$: under-relaxation
  - $\omega = 1$: Gauss-Seidel method
  - $\omega > 2$ or $\omega < 0$: method diverges
- SOR method converges much faster than Jacobi or GS methods (***# iterations*** $\propto$ ***N*** instead of N^2)
- However, choosing optimal ω is difficult in general

Alternative – multigrid method

- Even the SOR method is not efficient enough for large problems (for large number of grid points)
- Say if the true solution contains both low-*k* modes and high-*k* modes
  - ***Low-k modes*** (i.e., smoother components): takes long time to relax
  - ***High-k modes*** (i.e., more oscillatory components): relax quickly
- These relaxation methods are called "***smoothers***" because they could reduce the errors for high-k modes quickly
- We could speed up the convergence by ***solving the low-k modes on coarser grids*** – idea behind the "***multigrid method***"

Multigrid (MG) method

- The MG method aims to use multiple levels of grids, so that the error components of various modes in the solutions can be reduced rapidly
- It uses *coarse* grids to solve *low-k* modes
  - Low-*k* modes can relax faster (because of fewer cells)
  - But it cannot capture the high-*k* modes
- It uses *fine* grids to solve *high-k* modes
  - Obtain the high-*k* solution not captured by the coarse grids
  - # iterations can be substantially reduced with reduced low-*k* errors from the coarse grid
- Operations between coarse and fine grids: *restriction/averaging* and *prolongation/interpolation*



Finding the correction on coarse grids

- Target equation to solve: $L\phi = \rho$
  - ϕ is the exact solution to this discretized equation
- After obtaining an approximate solution on the fine grid ($\tilde{\phi}$), on the ***coarse*** grid, instead of directly solving the above equation, we could solve for the ***correction*** for this approximate solution (ϕ^{corr})
- How to solve for ϕ^{corr}?

Residual: $\xi = \rho - L\tilde{\phi}$

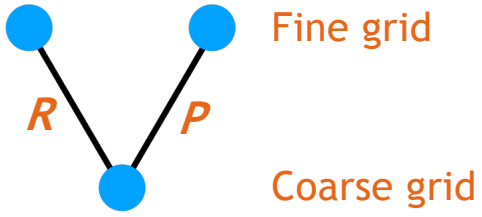
Correction: $\phi^{corr} = \phi - \tilde{\phi}$

$$L\phi^{corr} = \xi$$ Residual equation

A simple two-grid algorithm

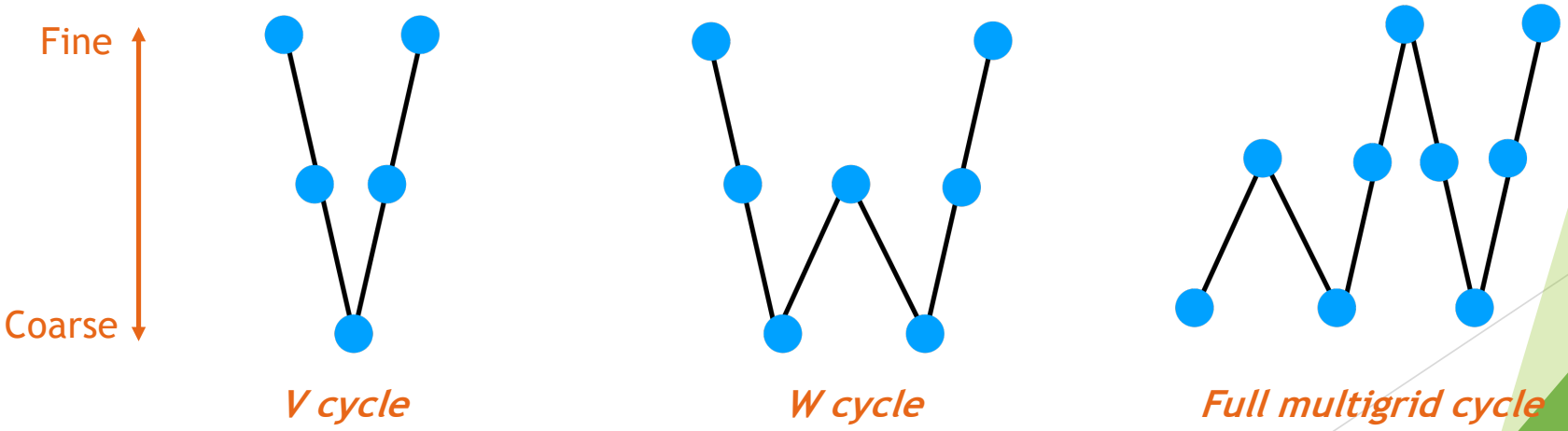
1) On fine grid, compute the solution using the smoother (e.g., GS method)
2) Compute the residual: $\xi = \rho - L\tilde{\phi}$
3) Restrict the residual to coarse grid
4) On coarse grid, compute the correction by solving the residual equation using the smoother
5) Prolong the correction to fine grid and improve the approximate solution
6) On fine grid, apply the smoother to the improved solution

This is a simple "*V-cycle*":



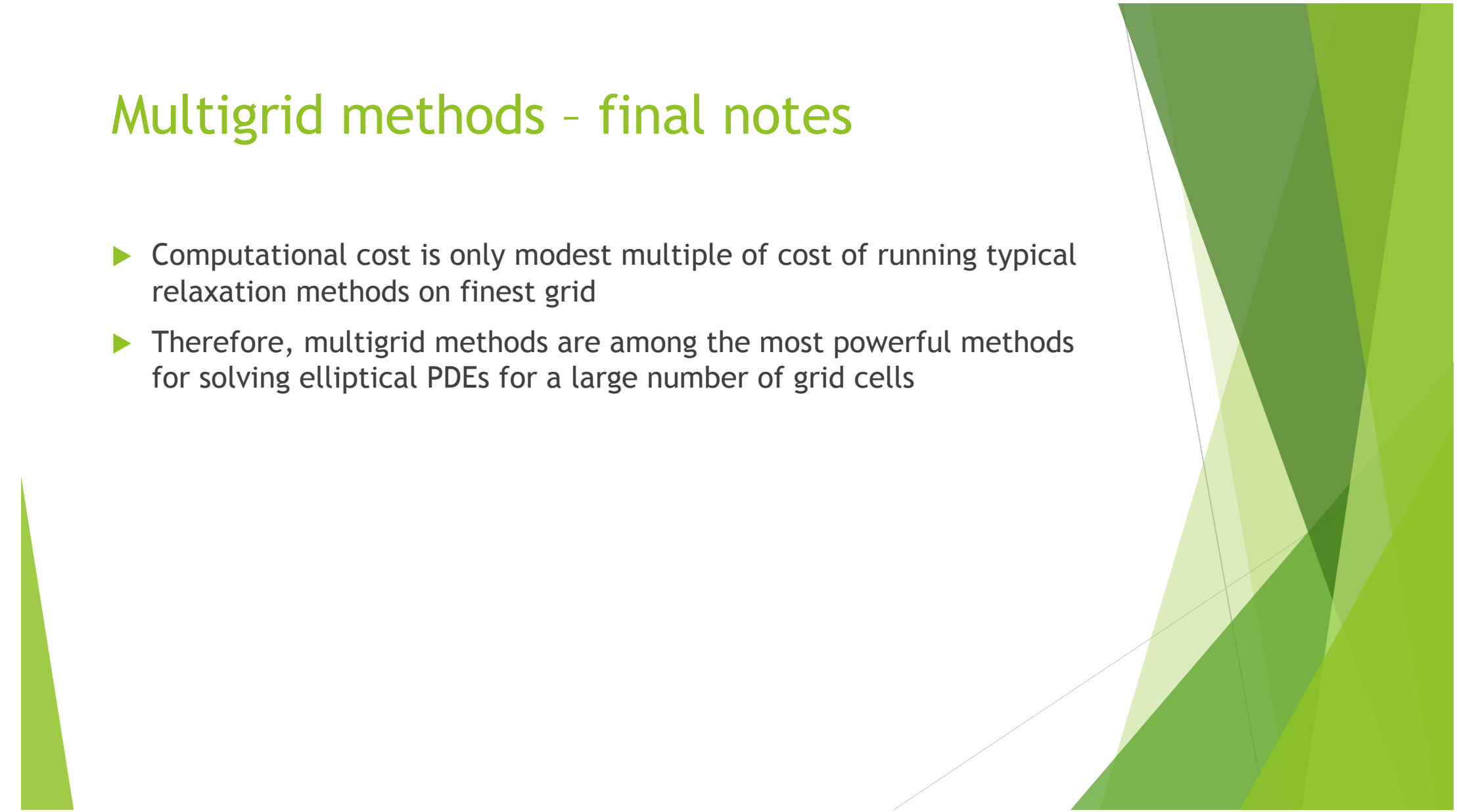
From two-grid to multigrid

- For step 4 in the last slide, we could introduce another even coarser grid, apply n numbers of two-grid iterations to obtain the correction ϕ^{corr}
- Multigrid method is to apply this procedure recursively
- Examples of cycling strategies:



Multigrid methods – final notes

- Computational cost is only modest multiple of cost of running typical relaxation methods on finest grid
- Therefore, multigrid methods are among the most powerful methods for solving elliptical PDEs for a large number of grid cells



Solving elliptical PDEs -- summary

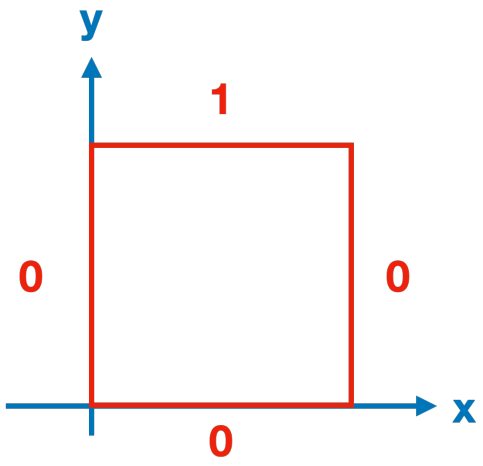
- PDE – elliptical equations (e.g., Laplace equation, Poisson equation) $\nabla^2\phi(\boldsymbol{r}) = \rho(\boldsymbol{r})$
- They describe systems that already reached steady-states – BVPs
- ***Relaxation method*** -- rewrite the equation into time-dependent form, iterate until equilibrium is reached
  - Jacobi method
  - Gauss-Seidel (GS) method
  - Successive over-relaxation (SOR) method
- ***Multigrid (MG) method*** – introduce coarser grids recursively to ***reduce errors for low-k modes*** more quickly
  - Use V-cycles or W-cycles to alternate between coarse/fine grids
  - Two operations: ***restriction*** (fine to coarse) and ***prolongation*** (coarse to fine)
  - On coarse grid, solve the ***residual equation*** to find ***corrections***

$$L\phi^{corr} = \xi$$

In-class exercises

Exercise -- solving the Laplace equation using the Jacobi relaxation methods

- For the Laplace equation, $\nabla^2\phi = \rho = 0 \rightarrow \frac{\partial \phi}{\partial t} = \nabla^2\phi$
- Jacobi method: $\phi_{i,j}^{n+1} = \frac{1}{4}\left(\phi_{i+1,j}^{n} + \phi_{i-1,j}^{n} + \phi_{i,j+1}^{n} + \phi_{i,j-1}^{n} - \Delta^2 \rho_{i,j}\right)$
- Let's use this B.C. for ϕ:



$n_x = n_y = 10$
$(x_{min}, x_{max}) = (0, 1)$
$(y_{min}, y_{max}) = (0, 1)$

Exercise -- solving the Laplace equation using the Jacobi relaxation methods

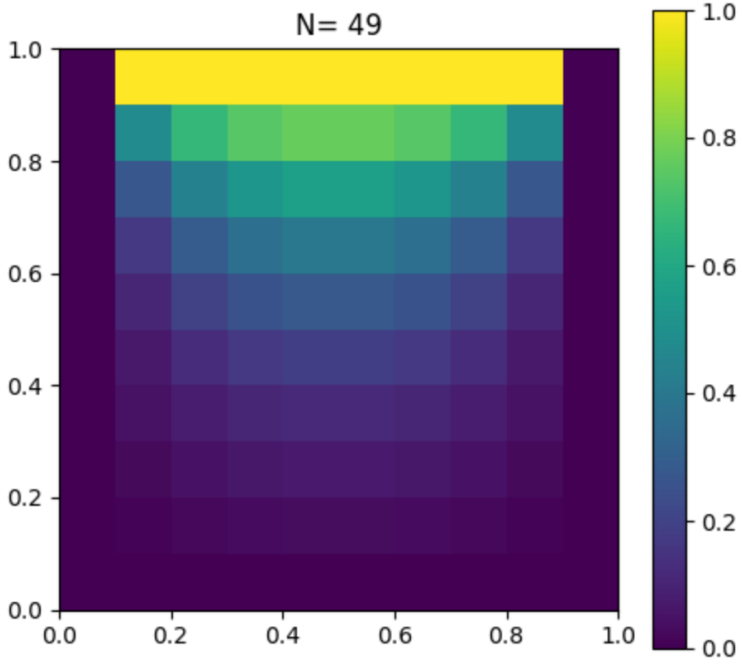
- Download the sample Jupyter Notebook from the following link: https://www.dropbox.com/s/r0m7zyy0kcxj89r/laplace_template_L14.ipynb?dl=0
- If you know how to run Jupyter Notebooks from CICA, feel free to do so. Otherwise you could go to the web-based Jupyter Lab: https://jupyter.org/try-jupyter/lab/
- Upload the template to the Jupyter Lab environment. Take a look at the template and understand how it works
- Implement the B.C. in the `set_boundary_conditions` function and the Jacobi method in the `evolve_jacobi` function
- Run the code and visualize the result

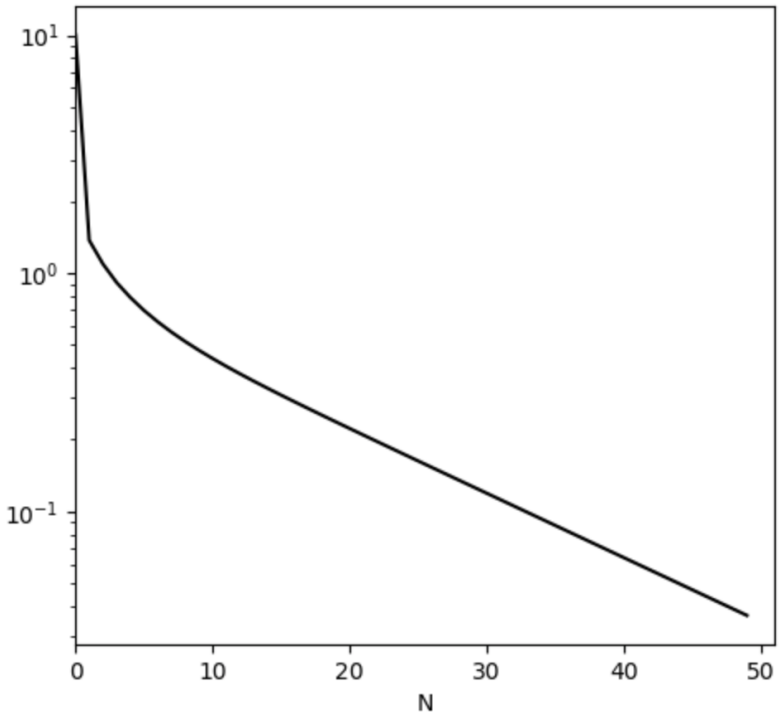
Jacobi method

$$\phi_{i,j}^{n+1} = \frac{1}{4}\left(\phi_{i+1,j}^{n} + \phi_{i-1,j}^{n} + \phi_{i,j+1}^{n} + \phi_{i,j-1}^{n} - \Delta^2 \rho_{i,j}\right)$$

Exercise -- solving the Laplace equation using the Jacobi relaxation methods

- Results for the Jacobi method:





Exercise #1b – solving the Laplace equation using the Gauss-Seidel method

- Implement the Gauss-Seidel (GS) method in the evolve_gauss_seidel function
- Run the code and visualize the result. Does it converge faster than the Jacobi method?

GS method

$$\phi_{i,j}^{n+1} = \frac{1}{4}\left(\phi_{i+1,j}^{n/n+1} + \phi_{i-1,j}^{n/n+1} + \phi_{i,j+1}^{n/n+1} + \phi_{i,j-1}^{n/n+1} - \Delta^2 \rho_{i,j}\right)$$

use the updated values (n+1) instead of the original values (n) whenever available

Exercise #1c – solving the Laplace equation using the SOR method

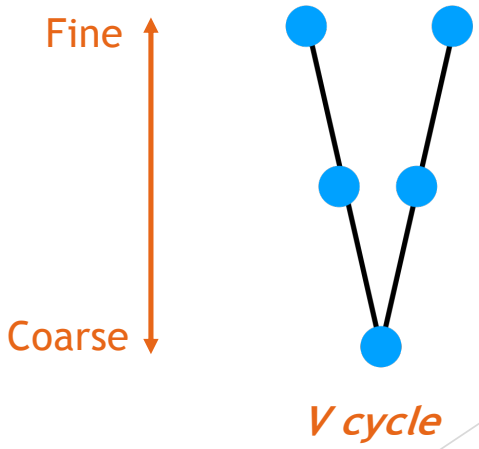
- Implement the ***successive over-relaxation (SOR)*** method, which applies a weighting to the original solution and the GS solution
- Run the code (using ***w = 1.2***) and visualize the result. Does it converge faster than the Jacobi/GS methods?
- To get the bonus credit, please submit your code and the screenshots for the ***SOR*** method to the TAs by end of today (12/15/2022)

SOR method

$$\phi_{i,j}^{n+1} = (1-\omega)\phi_{i,j}^{n} + \omega\phi_{i,j,GS}^{n+1}$$

Exercise #2 – solving the Laplace equation using the multigrid method

- Download the sample Jupyter Notebook from the following link: https://www.dropbox.com/s/r0m7zyy0kcxj89r/laplace_template_L14.ipynb?dl=0
- Implement the ***multigrid*** method. Try using a ***V-cycle***. For simplicity, use the GS method for NITER iterations to obtain the solutions on each level (we don't solve the residual equation for now)
- Run the code and visualize the result

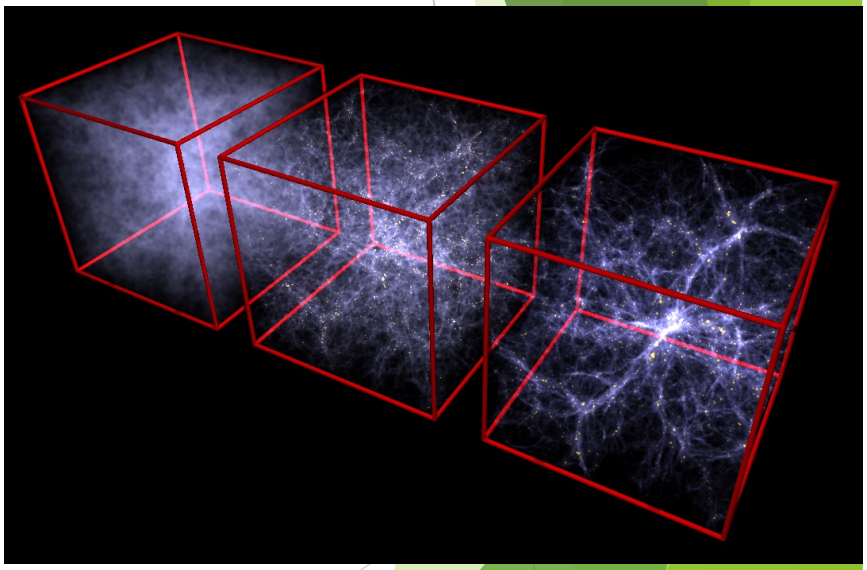


V cycle

N-body simulations

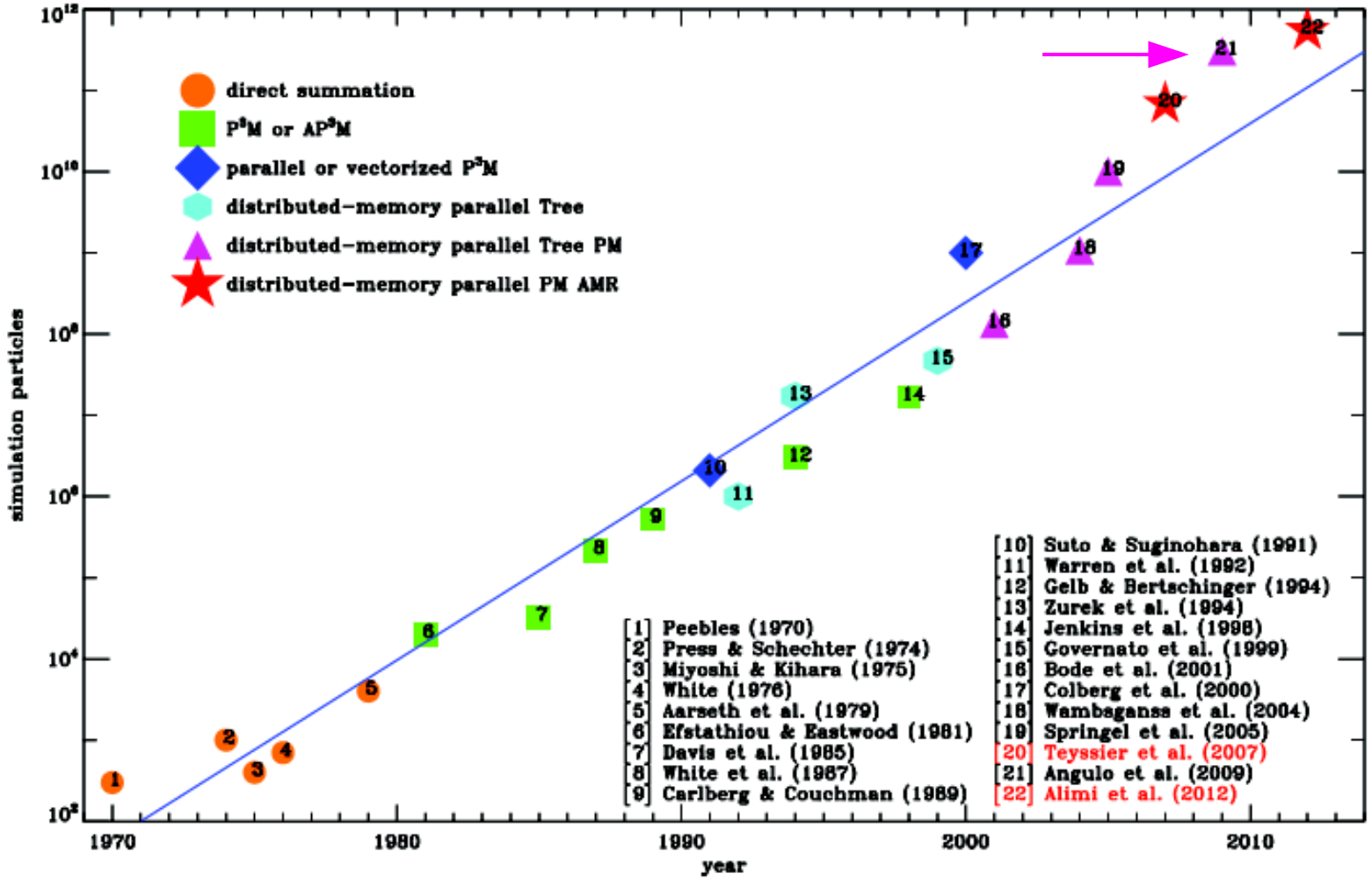
N-body simulations

- An N-body simulation is a simulation of a dynamical system of ***particles*** under the influence of physical forces (e.g., gravity, EM force)
- Widely used in astrophysics
  - Plasmas (N >> 1 for ions/electrons)
  - Solar system dynamics (N~10 planets)
  - Globular clusters (N ~ 10^6 stars)
  - Galaxy mergers (N ~ 10^{12} stars)
  - Clusters of galaxies (N ~ 1000 galaxies)
  - Large scale structure (N >> 1 dark matter or gas particles)
- Widely used outside astrophysics
  - Network (internet, power grids)
  - Molecular and statistical mechanics
  - High-energy physics (accelerator simulation)
  - ...



Springel et al. (2003)

Sizes of cosmological N-body simulations



AbacusSummit:
60 trillion particles
(Maksimova et al. 2021)

Angulo et al. (2012)

What do particles represent?

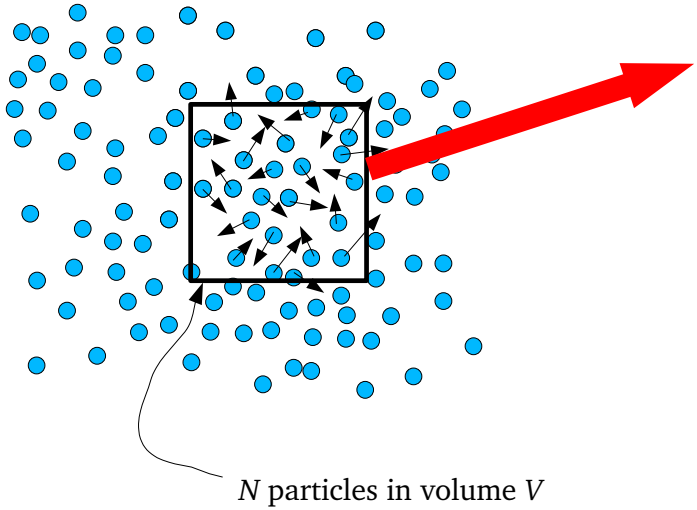
- Planets, stars, supernovae, black holes
  - Each particle represents a single point mass
- Star clusters
  - Each particle represents a bunch of stars (e.g., star particles in cosmological simulations)
- Dark matter
  - Monte-Carlo sampling of the phase space distribution function of DM
  - Can be either collisionless (CDM) or collisional (self-interacting DM)
- Gas
  - Monte-Carlo sampling of the phase space distribution function of gas (e.g., smooth particle hydrodynamics/SPH simulations)
- Tracers
  - Trace the trajectory of gas elements
- Photons
  - Monte-Carlo simulations of radiative transfer equations

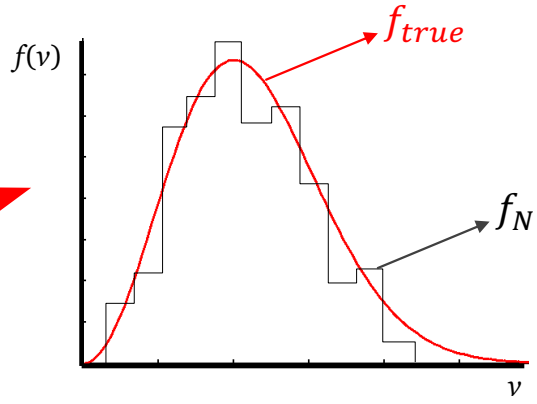
Considerations for simulating particles

- Point-mass objects
  - Essential to capture ***two-body relaxation (interactions/collisions/scattering)***
  - Need to deal with diverged gravity at the center
  - Need to capture binary formation
- Finite-sized objects (e.g., star clusters, dark matter, gas particles)
  - Need to avoid two-body relaxation and binary formation
- Particles can be created, destroyed, or scattered on-the-fly during the simulations
- Particle properties (e.g., mass, age, metallicity, spin, stellar composition, etc) could change on-the-fly
- Feedback (e.g., stellar wind, SN explosion, AGN jets) could be implemented using particles

When particles represent finite-sized objects

- In many applications, we use particles as a collection of particles (e.g., dark matter, gas, dust)
- N particles in volume V are used to randomly sample the particle distribution function





$$n(\boldsymbol{x}) = \int f(\boldsymbol{x}, v)\, dv \approx \frac{N}{V}$$

$$\frac{|f_N - f_{true}|}{|f_{true}|} \propto N^{-1/2}$$

- As $N \to \infty$, error ("shot noise") in approximate distribution function f_N goes to 0

The problem of two-body relaxation

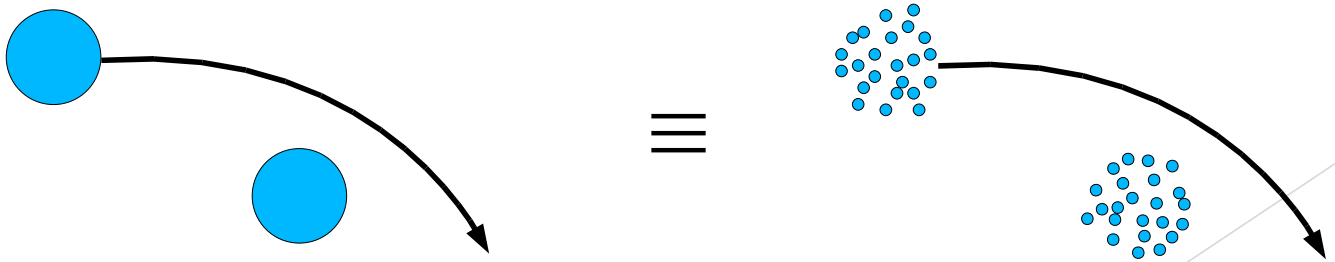
- Consider particle interactions due to gravity, then the evolution of particle positions and velocities can be treated as an IVP:

$$\frac{d\boldsymbol{x}_i}{dt} = \boldsymbol{v}_i$$

$$\frac{d\boldsymbol{v}_i}{dt} = -\nabla\phi_{ext} - G\sum_{\substack{j=1 \\ j\neq i}}^{N} m_j \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{|\boldsymbol{x}_i - \boldsymbol{x}_j|^3}$$

ϕ_{ext}: external potential

- During close encounters, there could be large-angle scattering between two particles. However, in the real world such coherent scattering for a collection of particles would not happen



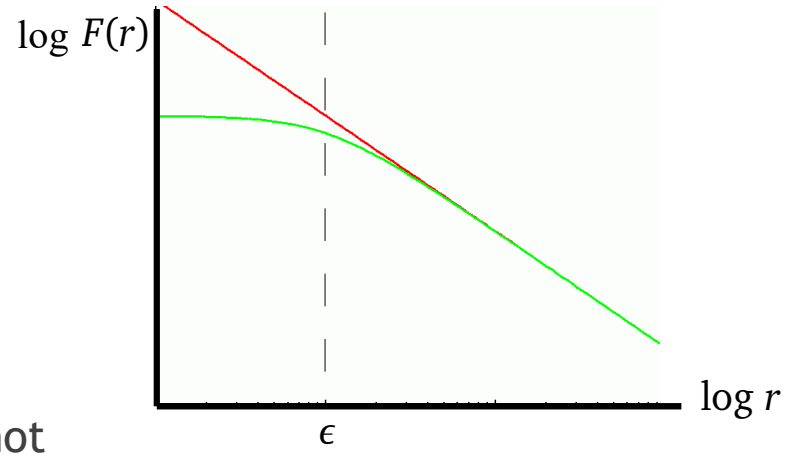
unphysical two-body relaxation

How do we avoid unphysical two-body relaxation?

- ***Use large N***
  - Using too few particles would lead to rapid isotropization of particle velocities due to two-body relaxation. Thus, convergence tests are essential
- ***Use a softened force law***
  - For gravity this typically takes the form:

$$F(r) = -\frac{Gm^2}{r^2 + \epsilon^2}$$

  - The ***softening length*** ϵ causes the force at small separations to be smaller, so close encounters would not produce large deflections



Choosing proper values of N and ϵ

- We would like to ***increase N (to reduce shot noise)*** and ***decrease ϵ (to maximize force resolution)***
- However, their effects are connected so these parameters need to be chosen carefully
- If we fix N and reduce ϵ
  - we have better force resolution
  - but increase unphysical two-body relaxation
- If we fix ϵ and increase N
  - we minimize unphysical two-body relaxation and shot noise
  - but do not improve force resolution (features smaller than ϵ cannot be resolved)
- Therefore, typically we need to vary their values simultaneously: $N\epsilon^3 \sim constant$

Evolving the particles

- Simulating the evolution of particles due to some forces is an ***IVP***
- Basic algorithm (recall lecture 7):
  - Compute the force/acceleration of each particle at t_n: $\boldsymbol{a}_i^n$
  - Use $\boldsymbol{a}_i^n$ to advance $\boldsymbol{v}_i^n$: $\boldsymbol{v}_i^{n+1} = \boldsymbol{v}_i^n + \Delta t \boldsymbol{a}_i^n$
  - Use $\boldsymbol{v}_i^n$ to advance $\boldsymbol{x}_i^n$: $\boldsymbol{x}_i^{n+1} = \boldsymbol{x}_i^n + \Delta t \boldsymbol{v}_i^n$
- Need to choose
  - A ***time integration method***
  - A ***force computation method*** (more details later)
- Considerations involved:
  - How many timesteps are needed
  - What level of accuracy is needed
  - What type of computing hardware is available

Time-integration methods – some notes

- Available solvers: Runge-Kutta, leapfrog, symplectic integrators, etc
- But there involve some special requirements that would affect our choice of a particular solver:
  - We require at least $\mathcal{O}(\Delta t^2)$ accuracy
  - If N is large, we would like to avoid implicit methods
  - We would like to avoid methods that involve multiple evaluations of the acceleration at each step (since they are expensive)
  - If a large number of timesteps are required, the method should explicitly conserve energy, otherwise the accumulation of roundoff errors would make the solution meaningless

Force computation methods

Algorithm #1: particle-particle (PP) or direct N-body

- The simplest approach for force calculation is to compute the acceleration for each particle ***directly***:

$$\boldsymbol{a}_i = -G \sum_{\substack{j=1 \\ j \neq i}}^{N} m_j \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{|\boldsymbol{x}_i - \boldsymbol{x}_j|^3}$$

- **Pros:** Force law is exact, so for applications in which two-body interactions are important (e.g., planetary systems, globular cluster dynamics), this may be the best method
- **Cons:** Force computation scales as N^2 -- very expensive for even modest N

Algorithm #2: particle-mesh (PM)

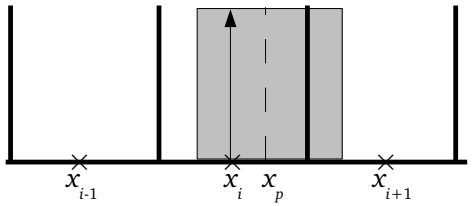
- If particle interactions are not important (i.e., collisionless), we can speed up dramatically by using fast mesh-based Poisson solvers, which scale as $N_g ln N_g$
- Procedure:
  - **Step 1**: assign each particle's mass to a grid using a "***mass assignment operator***". Sum contributions from all particles to obtain a density field ρ
  - **Step 2**: using a ***mesh-based Poisson solver***, solve $\nabla^2 \phi = 4\pi G \rho$
  - **Step 3**: compute the gravitational force on the grid by finite-differencing the potential
  - **Step 4**: interpolate force onto the particles using a "***force interpolation operator***"
  - **Step 5**: advance particle positions and velocities in time
  - **Step 6**: repeat

PM – mass assignment operator

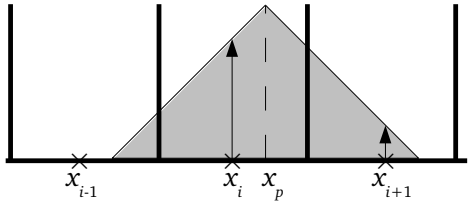
- A mass assignment operator (step 1 in PM) takes a particle's position and assign its mass to one or more nearby mesh points
- Example in 1D: $$\rho_i = \frac{m}{\Delta x} \sum_{p=1}^{N_p} \underbrace{W(x_i - x_p)}_{\text{assignment function}}$$
- Three commonly used operators:
  - ***Nearest grid point (NGP)*** $$W_{NGP}(x) = \begin{cases} 1 & -\Delta x/2 < x < \Delta x/2 \\ 0 & otherwise \end{cases}$$
  - ***Cloud-in-cell (CIC)*** $$W_{CIC}(x) = \begin{cases} 1 + x/\Delta x & -\Delta x < x \\ 1 - x/\Delta x & x < \Delta x \\ 0 & otherwise \end{cases}$$
  - ***Triangle-shaped cloud (TSC)*** $$W_{TSC}(x) = \underbrace{W_{CIC}(x) * W_{NGP}(x)}_{\text{convolution}}$$

Assignment functions W

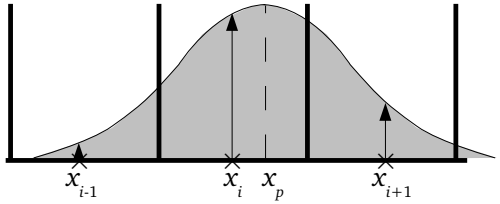
NGP



CIC



TSC



PM – mass assignment operator

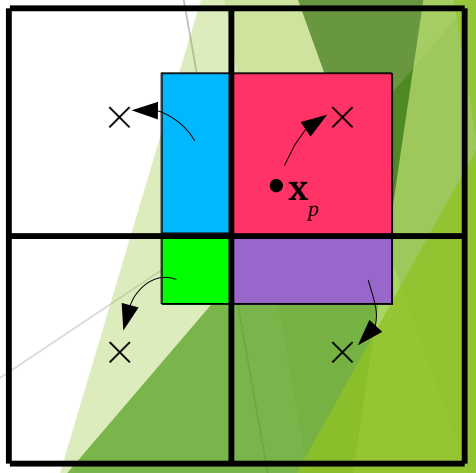
- The same operators can be used in reverse as the "force interpolation operators" (step 4 in PM) to interpolate forces from the grid onto particle positions:

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

- For multi-D problems we typically use products of 1D weighting functions:

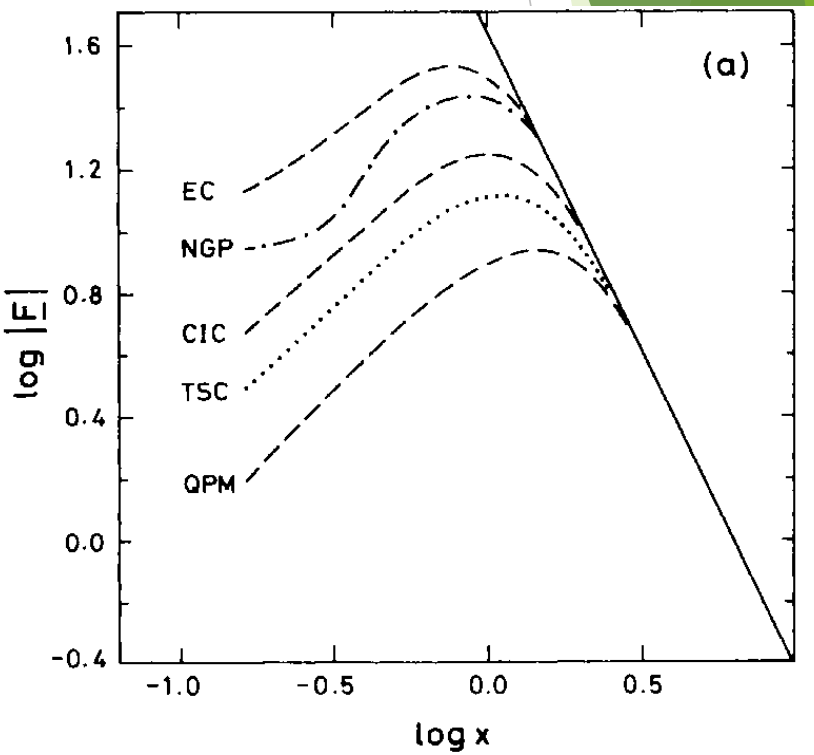
$$\rho_{ijk} = \frac{m}{\Delta x \Delta y \Delta z} \sum_{p=1}^{N_p} W(x_i - x_p) W(y_j - y_p) W(z_k - z_p)$$

Example: 2D CIC



PM – effects of force smoothing

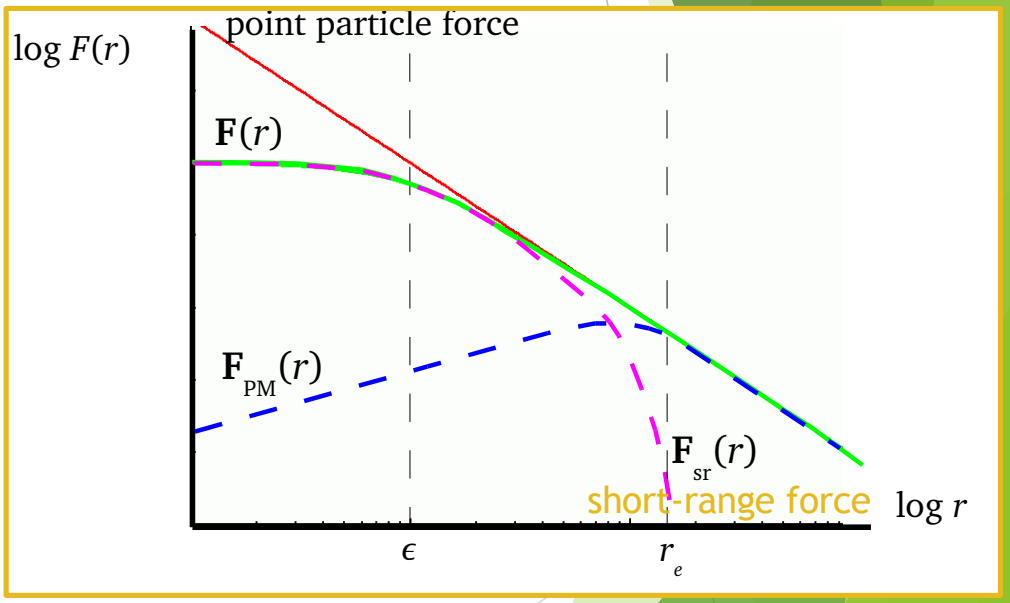
- Because particles are distributed onto mesh, the force felt by particles deviate from Newtonian law at separations smaller than 2-3 zone widths
- ***Force resolution = 2-3 zone widths*** in PM method
- PM methods automatically include ***force smoothing/softening*** needed to prevent unphysical two-body relaxation



Efstathiou et al. (1985)

Algorithm #3: particle-particle-particle-mesh (P^3M)

- Invented in the early 1970s by R. W. Hockney and J. W. Eastwood
- Designed to improve force accuracy of PM at small separations without cost of PP
- Idea is to use PP for nearby particles and PM for distant particles
- Improved force resolution ~ ϵ = softening length for the short-range force



Algorithm #4: tree codes

- Developed in mid 1980s by Appel, Jernigan & Porter, and Barnes & Hut
- Idea:
  - for nearby particles, sum their forces directly
  - *for distant particles (especially clustered particles), use approximated forces (e.g., multipole expansion) at their center of mass (C.O.M.)*
- This could reduce the cost to $\mathcal{O}(N_p ln N_p)$
- How do we distinguish nearby/distant particles? use the "*tree*"!

Example: merging of two galaxies

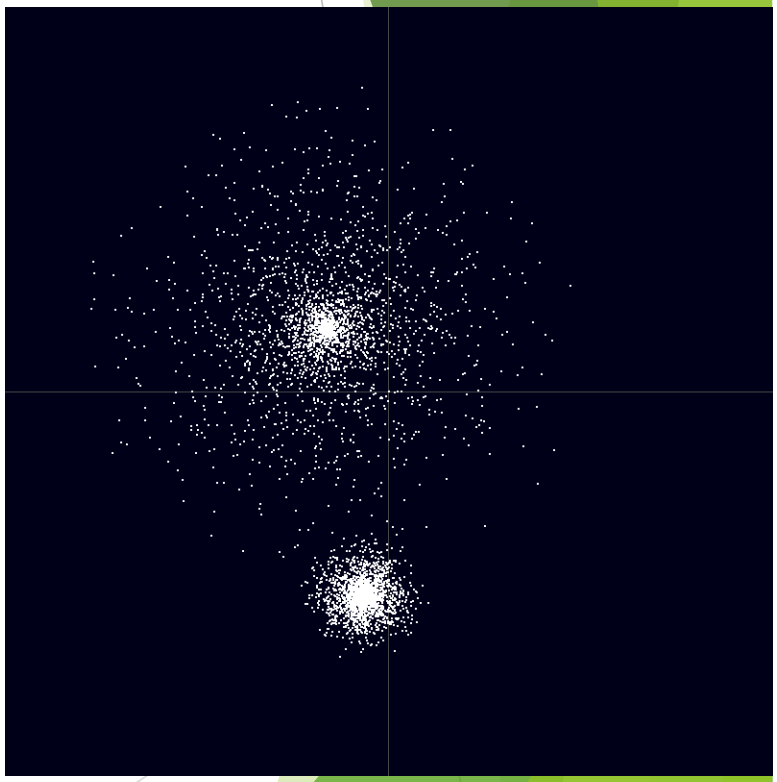
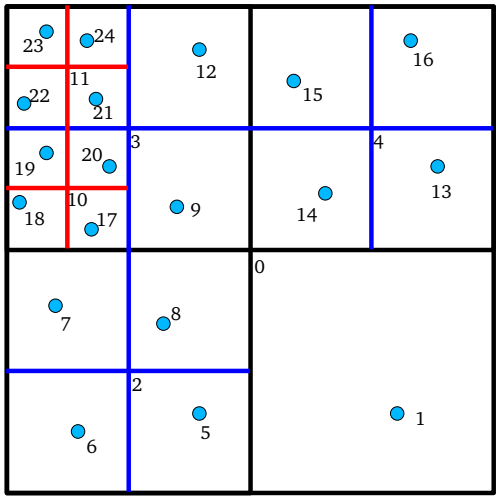
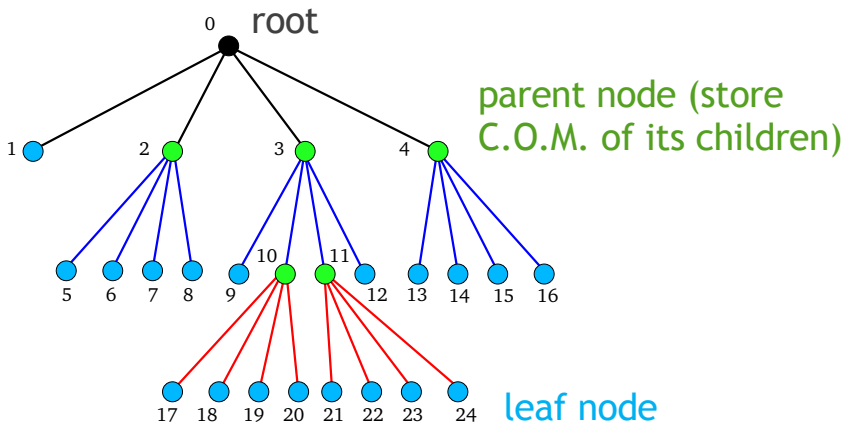


Image credit: Wikipedia

Tree algorithm

- **Step 1**: store the particles into a tree structure (simulation volume is divided into cubic cells until each cell has only one particle)



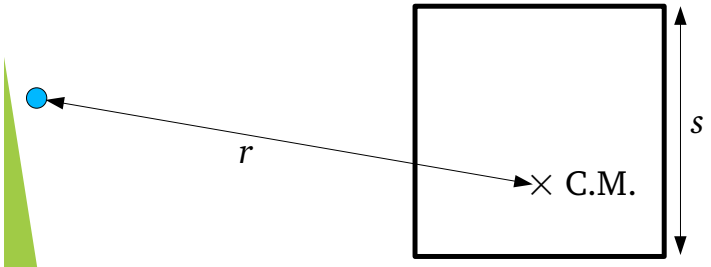


- **Step 2**: for each particle, go from the root
  - If a parent node is distant enough, compute the force using the C.O.M; otherwise go down to its children
  - If a leaf node is reached, sum the forces directly

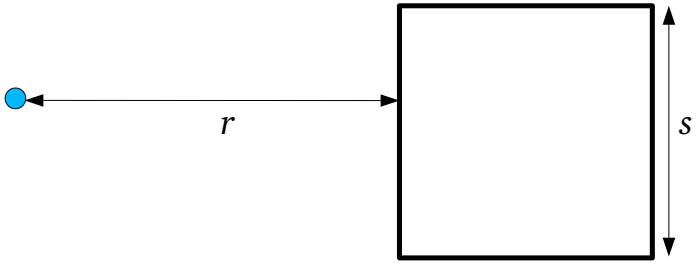
Tree codes – distance criteria

How to tell if a node is distant enough? Three commonly used criteria:

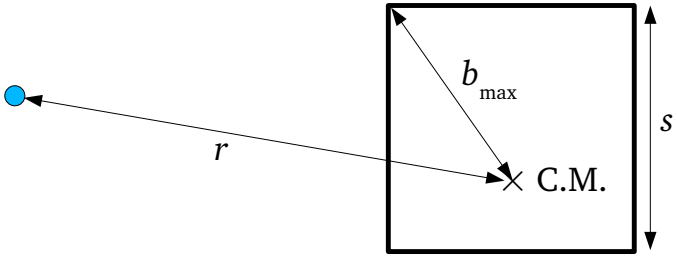
Barnes-Hut method: if $\frac{s}{r} < \theta$



Min-distance method: if $\frac{s}{r} < \theta$



Bmax method: if $\frac{b_{max}}{r} < \theta$



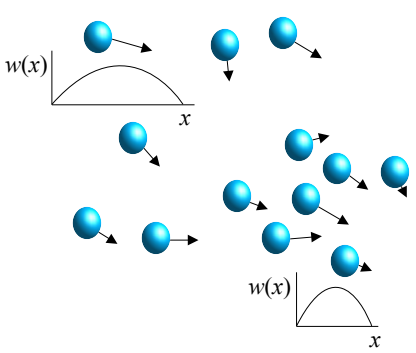
Examples of astrophysical N-body codes

- Particle-particle (PP)
  - GRAPESPH
- Particle-mesh (PM)
  - FLASH
  - Enzo
  - ART
  - TPM
- P^3M
  - Hydra
- Tree
  - GADGET (TreeSPH)
  - Gasoline (TreeSPH)

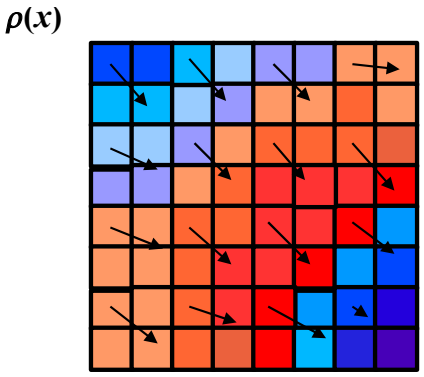
Smoothed particle hydrodynamics (SPH)

Smoothed particle hydrodynamics (SPH)

- Invented by Lucy (1977) and Gingold & Monaghan (1977)
- It is a ***particle-based method for hydrodynamics***
- Particles are ***moving interpolation centers*** for fluid quantities



SPH



Eulerian grid-based hydro

- Popular in astrophysics because
  - It is a ***Lagrangian*** scheme (resolution automatically adapts to density)
  - It is easy to implement on top of an existing N-body code

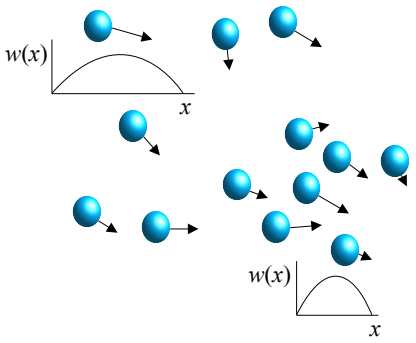
SPH particles

- Each SPH particle represents a discrete moving elements of the fluid
- Each particle is tagged with its position **x** and fluid quantities (mass, density, pressure, velocity, smoothing length *h*, ...)
- Physical quantity of the particle can be obtained using a "***smoothing kernel***" with a characteristic radius (i.e., the "***smoothing length h***"):

$$A(\boldsymbol{r}) = \sum_j V_j A_j W(|\boldsymbol{r} - \boldsymbol{r}_j|, h)$$

- Example of particle density:

$$\rho_i = \rho(\boldsymbol{r}_i) = \sum_j m_j W_{ij}$$



SPH

SPH smoothing kernels

Commonly used kernels include

- ***Gaussian kernel***:

$$W(x,h)=\frac{1}{h\sqrt{\pi}}e^{-x^2/h^2}$$

- ***Spline kernel*** (*d* = dimensions):

$$W(x,h)=\frac{\sigma}{h^d}\begin{cases}1-\frac{3}{2}\left(\frac{x}{h}\right)^2+\frac{3}{4}\left(\frac{x}{h}\right)^3 & if\ 0\le\frac{x}{h}\le 1\\ \frac{1}{4}\left[2-\left(\frac{x}{h}\right)\right]^3 & if\ 1\le\frac{x}{h}\le 2\\ 0 & otherwise\end{cases}$$

- Advantage of spline kernel: no interaction for x > 2h

SPH equations

- With $\rho_p = \sum_q m_q W_{pq}$, the continuity equation can be written as:

$$\frac{d\rho_p}{dt} = \sum_q m_q (\boldsymbol{v}_p - \boldsymbol{v}_q) \nabla_p W_{pq}$$

- For the momentum equation, note that momentum would not be conserved if asymmetric gradient formulations are used, e.g.,

$$\rho_p \nabla P_p = \sum_q m_q (P_q - P_p) \nabla_p W_{pq}, \qquad \nabla_p W_{pq} \equiv \nabla_{\boldsymbol{x}_p} W(\boldsymbol{x}_q - \boldsymbol{x}_p, h_p)$$

- Therefore, we need to "***symmetrize***" the gradient using $\frac{\nabla P}{\rho} = \nabla\left(\frac{P}{\rho}\right) + \frac{P}{\rho^2}\nabla\rho$

- The momentum equation then becomes

$$\frac{d\boldsymbol{v}_p}{dt} = -\sum_q m_q \left(\frac{P_q}{\rho_q^2} + \frac{P_p}{\rho_p^2}\right) \nabla_p W_{pq}$$

SPH equations

- Similarly, the symmetrized specific internal energy equation is:

$$\frac{d\varepsilon}{dt} = -\left(\frac{P}{\rho}\right)\nabla\cdot\boldsymbol{v} = -\nabla\cdot\left(\frac{P\boldsymbol{v}}{\rho}\right) + \boldsymbol{v}\cdot\nabla\left(\frac{P}{\rho}\right)$$

$$\rightarrow \quad \frac{d\varepsilon_p}{dt} = \frac{1}{2}\sum_q m_q\left(\frac{P_q}{\rho_q^2} + \frac{P_p}{\rho_p^2}\right)(\boldsymbol{v}_p - \boldsymbol{v}_q)\cdot\nabla_p W_{pq}$$

- Finally, the particles are moved via $\frac{d\boldsymbol{x}_p}{dt} = \boldsymbol{v}_p$

Artificial viscosity in SPH

- In SPH codes, it is shown that ***artificial viscosity*** is needed to improve numerical stability of supersonic flows (i.e., shocks)
- The artificial viscosity corresponds to an additional term in the momentum equation:

$$\frac{d\,\boldsymbol{v}_p}{dt} = -\sum_q m_q \left(\frac{P_q}{\rho_q^2} + \frac{P_p}{\rho_p^2} + \underline{\Pi_{pq}} \right) \nabla_p W_{pq}$$

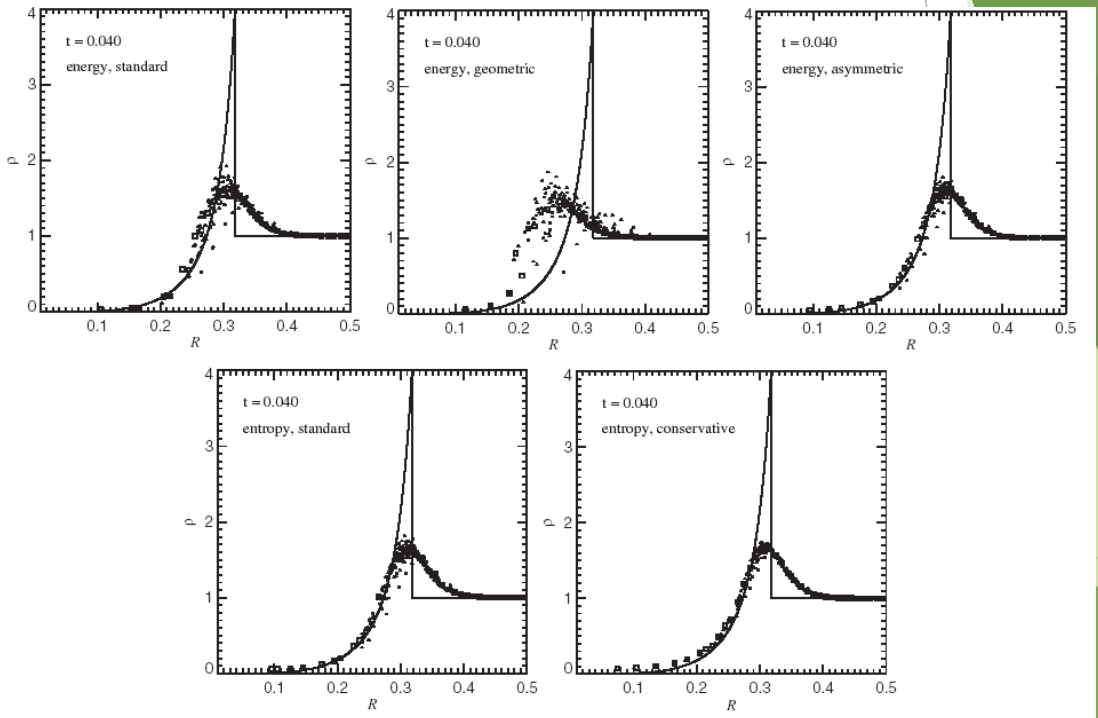
artificial viscosity

- However, using the artificial viscosity to capture the shocks means that the shocks cannot be resolved with high resolutions

Performance of SPH

- Example of the Sedov test:

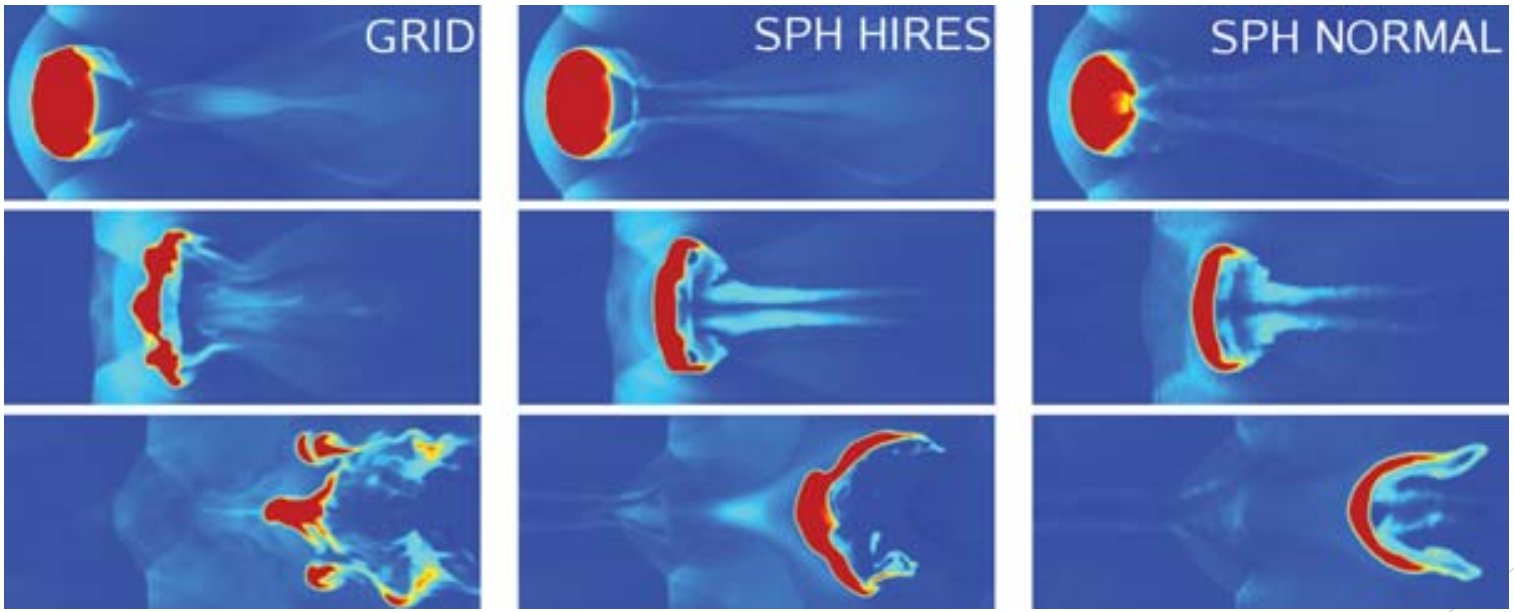
Springel & Hernquist (2002)



- 3D SPH requires at least ~10^4 particles to get reasonable results for the shock problems

Compare grid-based codes & SPH codes

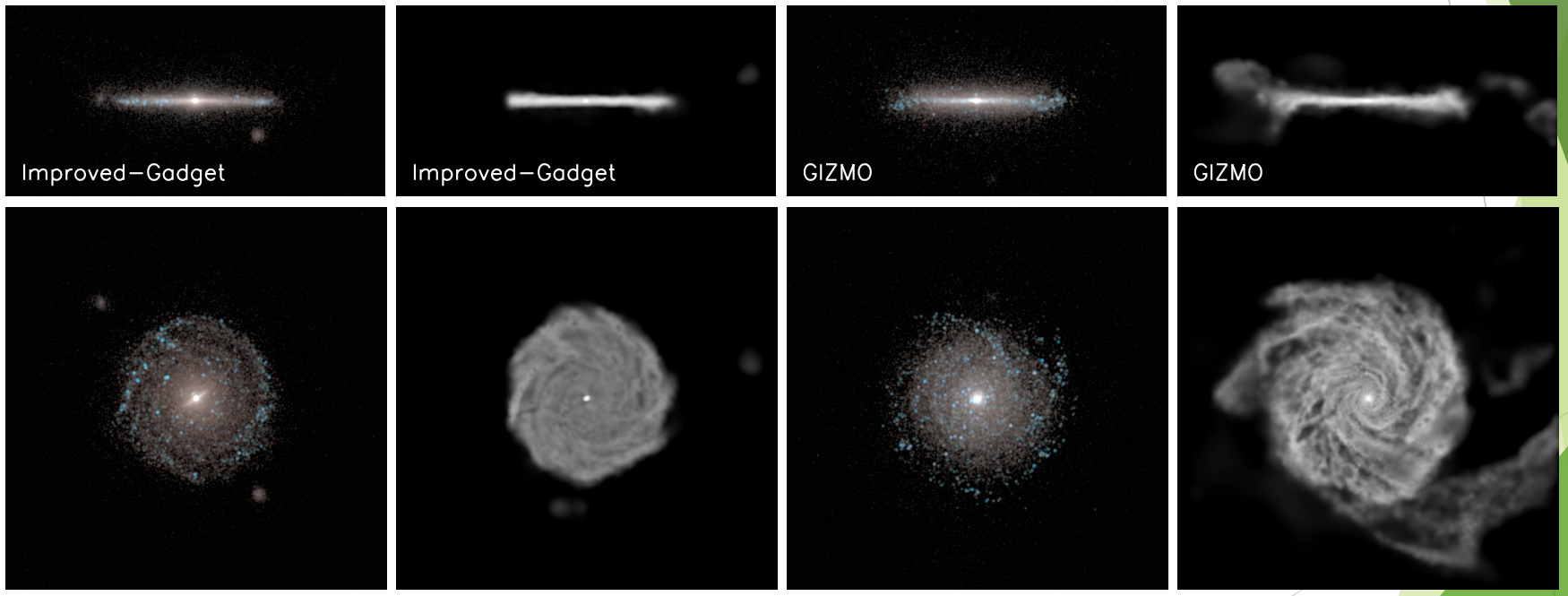
SPH simulations typically show much ***suppressed hydro instabilities*** compared to grid-based codes



Agertz et al. (2007)

Compare grid-based codes & SPH codes

SPH codes typically show ***suppressed turbulence*** than grid-based codes



Zhu & Li (2017)

Comparisons between grid-based codes & SPH codes

- This is exercise #3 in your HW6!
- Please search in the literature (in NASA ADS or arXiv) using keywords (e.g., ”code comparison”, “SPH”)
- Choose one of the articles you’re interested in. Read the article and summarize what you learned about the differences among the codes being compared (see detailed descriptions in HW6)
- Next week we will have an in-class discussion about the ***pros*** and ***cons*** of grid-based vs. SPH codes. Be prepared to discuss with your classmates and contribute your opinions!

N-body simulations -- summary

- N-body or particle-based simulations are widely used in astrophysics
- Particles could represent a single object (e.g., planets, stars) or a Monte-Carlo sampling of a collection of particles
  - For the latter, need to avoid ***unphysical two-body relaxation*** by using a large N or using a softened force law
- To evolve particles, we need
  - ***Time-integration methods***: leapfrog, Runge-Kutta, symplectic integrators
  - ***Force-integration methods***: PP, PM, P^3M, tree
- ***Smoothed particle hydrodynamics (SPH):*** particle-based, Lagrangian method for hydrodynamics
  - Uses ***SPH kernels*** to compute fluid quantities
  - Uses ***symmetrized SPH equations*** to evolve the fluid quantities
  - Uses ***artificial viscosity*** for shock capturing

References & acknowledgements

- Course materials of Computational Astrophysics from Prof. Kuo-Chuan Pan (NTHU)
- Course materials of Computational Astrophysics from Prof. Hsi-Yu Schive (NTU)
- Course materials of Computational Astrophysics and Cosmology from Prof. Paul Ricker (UIUC)
- “Computational Physics” by Rubin H. Landau, Manuel Jose Paez and Cristian C. Bordeianu
- “Scientific Computing – An Introductory Survey” by Michael T. Heath