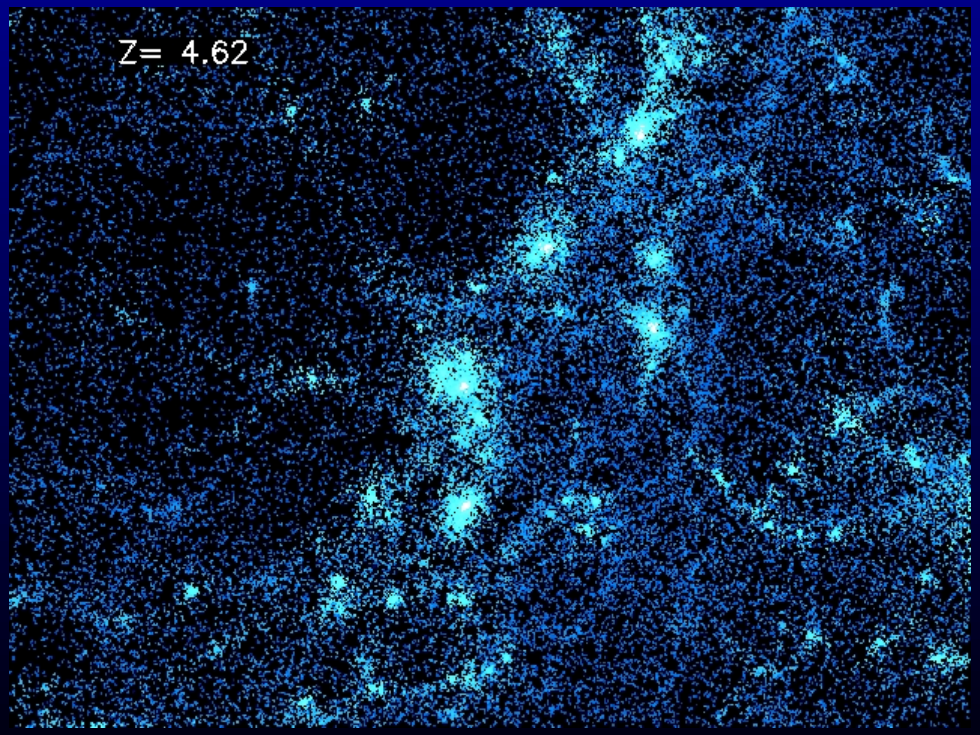
Writing a PM code

Andrey Kravtsov
andrey@oddjob.uchicago.edu
astro.uchicago.edu/~andrey/Talks/PM/pm.pdf

February 20, 2002



Particle-Mesh (PM) technique: a brief history

Particle-Mesh (PM) technique: a brief history

- PM was invented in the 1950's at LANL for simulations of compressible fluid flows

Particle-Mesh (PM) technique: a brief history

- PM was invented in the 1950's at LANL for simulations of compressible fluid flows
- However, the first wide-spread application was for collisionless plasma simulations (for which PM was reinvented in the 1960's and popularized by Hockney and collaborators)

Particle-Mesh (PM) technique: a brief history

- PM was invented in the 1950's at LANL for simulations of compressible fluid flows
- However, the first wide-spread application was for collisionless plasma simulations (for which PM was reinvented in the 1960's and popularized by Hockney and collaborators)
- In the late 70's PM was first applied for 3D cosmological simulations. The method and its descendants (such as P^3M, AP^3M, and ART) have been used in most cosmological simulations ever since [1].

Particle-Mesh (PM) technique: a brief history

- PM was invented in the 1950's at LANL for simulations of compressible fluid flows
- However, the first wide-spread application was for collisionless plasma simulations (for which PM was reinvented in the 1960's and popularized by Hockney and collaborators)
- In the late 70's PM was first applied for 3D cosmological simulations. The method and its descendants (such as P^{3}M, AP^{3}M, and ART) have been used in most cosmological simulations ever since [1].
- The popularity of PM in cosmology is due to
  - ⋆ its relative algorithmic simplicity and speed (the running time scales as $\propto O(N_p) + O(N_c \ln N_c)$, where N_p is the number of particles and N_c is the number of grid cells;
  - ⋆ natural incorporation of periodic boundary conditions;

[1] In the 1990's Tree codes have also enjoyed increasing popularity

Model equations

PM code solves the Poisson equation

$$\nabla^2\Phi = 4\pi G\rho_{\rm tot} - \Lambda,$$

and equations of motion of particles

$$\frac{d\mathbf{r}}{dt} = \mathbf{u}; \quad \frac{d\mathbf{u}}{dt} = -\nabla\Phi,$$

note that all variables are defined in proper coordinates and all spatial derivatives are also taken with respect to these coordinates.

However, it is convenient to re-write the equations in *comoving* variables and make them dimensionless by choosing suitable units (I will denote variables in code units with a tilde ˜). In the following, I will adopt the variables and units used in Anatoly Klypin's PM code. This is just an example. Feel free to choose your own - just make sure you are consistent!

$$\tilde{\mathbf{x}} \equiv a^{-1}\frac{\mathbf{r}}{r_0}, \quad \tilde{\mathbf{p}} \equiv a\ \frac{\mathbf{v}}{v_0}, \quad \tilde{\phi} \equiv \frac{\phi}{\phi_0}, \quad \tilde{\rho} = a^3\frac{\rho}{\rho_0},$$

where $\mathbf{x}$ is the comoving coordinates, $\mathbf{v} = \mathbf{u} - H\mathbf{r} = a\dot{\mathbf{x}}$ is the *peculiar velocity*[2] and ϕ is the *peculiar potential* defined as (Peebles 1980, p. 42)

$$\phi \quad = \quad \Phi + 1/2\ a\ddot{a}\,(r/a)^2 = \Phi + \frac{H_0^2}{2}\left(\Omega_{\Lambda,0} - \frac{1}{2}a^{-3}\Omega_{m,0}\right) r^2,$$

where

$$\Omega_{m,0} = \frac{8\pi G\rho_0}{3H_0^2}; \quad \Omega_{\Lambda,0} = \frac{\Lambda}{3H_0^2}.$$

[2] $p \propto av$ is called momentum, this choice of variable allows us to get rid of the annoying $(\dot{a}/a)$ terms in equations.

The quantities with subscript zero are the units in which corresponding physical variables are measured. The unit of length, r_0, is an arbitrary scale. I will chose r_0 to be the size of the PM grid cell:

$$r_0 = \frac{L_{\rm box}}{N_g}; \quad N_g^3 = \text{total number of grid cells}$$

the rest of the units are defined as

$$\begin{aligned} t_0 &\equiv H_0^{-1}, \\ v_0 &\equiv \frac{r_0}{t_0}, \\ \rho_0 &\equiv \frac{3H_0^2}{8\pi G}\Omega_{m,0}, \\ \phi_0 &\equiv \frac{r_0^2}{t_0^2} = v_0^2. \end{aligned}$$

It is also convenient to choose the expansion factor as time variable (using expressions for the Hubble constant: $\dot{a} = aH(a)$). For this choice of variables, the Poisson equation and equations of motion can be re-written as

$$\nabla^2\phi = 4\pi G\Omega_{m,0}\rho_{crit,0}a^{-1}\delta, \quad \delta = \frac{\rho - \bar{\rho}}{\bar{\rho}},$$

$$\frac{d\mathbf{p}}{da} = -\frac{\nabla\phi}{\dot{a}}, \quad \frac{d\mathbf{x}}{da} = \frac{\mathbf{p}}{\dot{a}a^2}.$$

where δ is the overdensity in comoving coordinates and $\dot{a}$ is

$$\dot{a} = H_0 a^{-1/2}\sqrt{\Omega_{m,0} + \Omega_{k,0}a + \Omega_{\Lambda,0}a^3}; \quad \Omega_{m,0} + \Omega_{\Lambda,0} + \Omega_{k,0} = 1$$

In dimensionless variables the equations are:

$$\tilde{\nabla}^2\tilde{\phi} = \frac{3}{2}\frac{\Omega_0}{a}\tilde{\delta},$$

$$\frac{d\tilde{\mathbf{p}}}{da} = -f(a)\tilde{\nabla}\tilde{\phi}, \quad \frac{d\tilde{\mathbf{x}}}{da} = f(a)\frac{\tilde{\mathbf{p}}}{a^2}.$$

where $\tilde{\delta} = \tilde{\rho} - 1$ and

$$f(a) \equiv H_0/\dot{a} = \left[a^{-1}\left(\Omega_{m,0} + \Omega_{k,0}a + \Omega_{\Lambda,0}a^3\right)\right]^{-1/2}.$$

These equations are used in the three main steps of a PM code:

In dimensionless variables the equations are:

$$\tilde{\nabla}^2\tilde{\phi} = \frac{3}{2}\frac{\Omega_0}{a}\tilde{\delta},$$

$$\frac{d\tilde{\mathbf{p}}}{da} = -f(a)\tilde{\nabla}\tilde{\phi}, \quad \frac{d\tilde{\mathbf{x}}}{da} = f(a)\frac{\tilde{\mathbf{p}}}{a^2}.$$

where $\tilde{\delta} = \tilde{\rho} - 1$ and

$$f(a) \equiv H_0/\dot{a} = \left[a^{-1}\left(\Omega_{m,0} + \Omega_{k,0}a + \Omega_{\Lambda,0}a^3\right)\right]^{-1/2}.$$

These equations are used in the three main steps of a PM code:

- Solve the Poisson equation using the density field estimated with current particle positions.

In dimensionless variables the equations are:

$$\tilde{\nabla}^2\tilde{\phi} = \frac{3}{2}\frac{\Omega_0}{a}\tilde{\delta},$$

$$\frac{d\tilde{\mathbf{p}}}{da} = -f(a)\tilde{\nabla}\tilde{\phi}, \quad \frac{d\tilde{\mathbf{x}}}{da} = f(a)\frac{\tilde{\mathbf{p}}}{a^2}.$$

where $\tilde{\delta} = \tilde{\rho} - 1$ and

$$f(a) \equiv H_0/\dot{a} = \left[a^{-1}\left(\Omega_{m,0} + \Omega_{k,0}a + \Omega_{\Lambda,0}a^3\right)\right]^{-1/2}.$$

These equations are used in the three main steps of a PM code:

- Solve the Poisson equation using the density field estimated with current particle positions.
- Advance momenta using the new potential.

In dimensionless variables the equations are:

$$\tilde{\nabla}^2\tilde{\phi} = \frac{3}{2}\frac{\Omega_0}{a}\tilde{\delta},$$

$$\frac{d\tilde{\mathbf{p}}}{da} = -f(a)\tilde{\nabla}\tilde{\phi}, \quad \frac{d\tilde{\mathbf{x}}}{da} = f(a)\frac{\tilde{\mathbf{p}}}{a^2}.$$

where $\tilde{\delta} = \tilde{\rho} - 1$ and

$$f(a) \equiv H_0/\dot{a} = \left[a^{-1}\left(\Omega_{m,0} + \Omega_{k,0}a + \Omega_{\Lambda,0}a^3\right)\right]^{-1/2}.$$

These equations are used in the three main steps of a PM code:

- Solve the Poisson equation using the density field estimated with current particle positions.
- Advance momenta using the new potential.
- Update particle positions using new momenta.

Data structures

For a PM code with the second-order accurate time integration we need the minimum of *six real numbers for positions and momenta for each particle* (assuming particles have the same mass) and *one real number for the potential of each grid cell*.

The array for the potential can be shared between density and potential: first use it for density, then replace it with potential when the Poisson equation is solved. However, for simplicity you can start with two grid arrays (for density and potential). Also, you will probably need auxiliary arrays for FFT, depending on which FFT solver you choose to use.

The convenient data structures are 1D or 3D arrays for particles (e.g., six 1D arrays for `x(i), y(i), z(i), vx(i), vy(i), vz(i)`) and 3D arrays for the grid variables (e.g., `rho(i,j,k), phi(i,j,k)`).

Run your tests for (32^3, 64^3) or (64^3, 128^3) particles and cells in which case memory requirements should not be an issue.

Density Assignment

The Particle Mesh algorithms assume that particles have certain size, mass, shape, and internal density. This determines the interpolation scheme used to assign densities to grid cells. Let's define the 1D particle shape, $S(x)$, to be mass density at the distance x from the particle for cell size Δx (Hockney & Eastwood 1981). The common choices are

- Nearest Grid Point (NGP): particles are point-like and all of particle's mass is assigned to the single grid cell that contains it:

$$S(x) = \frac{1}{\Delta x}\delta\left(\frac{x}{\Delta x}\right)$$

- Cloud In Cell (CIC): particles are cubes (in 3D) of uniform density and of one grid cell size.

$$S(x) = \frac{1}{\Delta x}\begin{cases} 1, & |x| < \frac{1}{2}\Delta x \\ 0, & \text{otherwise} \end{cases}$$

- Triangular Shaped Cloud (TSC):

$$S(x) = \frac{1}{\Delta x} \begin{cases} 1 - |x|/\Delta x, & |x| < \Delta x \\ 0, & \text{otherwise} \end{cases}$$

The fraction of particle's mass assigned to a cell ijk is the shape function averaged over this cell:

$$W(x_p - x_{ijk}) = \int_{x_{ijk}-\Delta x/2}^{x_{ijk}+\Delta x/2} dx' S(x_p - x');$$

$$W(\mathbf{r}_p - \mathbf{r}_{ijk}) = W(x_p - x_{ijk}) W(y_p - y_{ijk}) W(z_p - z_{ijk});$$

The density in a cell ijk is then

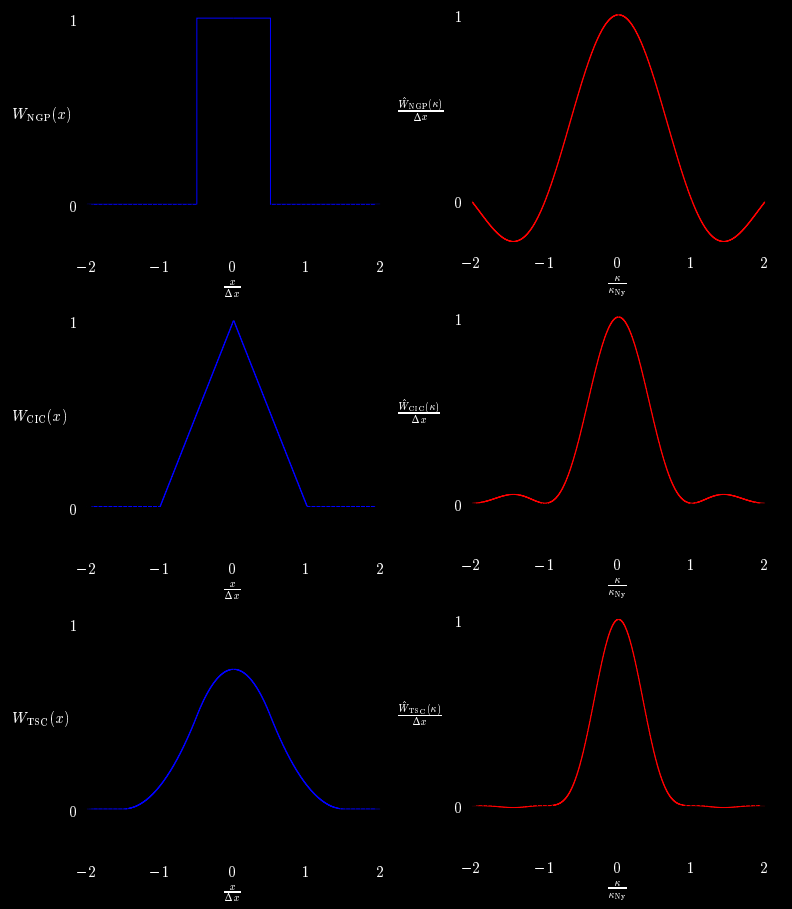
$$\rho_{ijk} = \sum_{p=1}^{N_p} m_p W(\mathbf{r}_p - \mathbf{r}_{ijk})$$

In practice, of course, we loop over particles and assign the density to neighboring cells as opposed to summing over all particles for each cell as the straightforward reading of the above equation would suggest.

PM interpolation kernels in real and Fourier space. Adopted from PM lectures by M.Gross

3-9

the density assignment wider in configuration space.



Implementing CIC interpolation

The choice of interpolation scheme is a tradeoff between accuracy and computational expense. The CIC scheme is both relatively cheap and accurate and is most commonly used in PM codes. I will now describe how to implement it (you can start coding with the simpler NGP scheme and upgrade it to CIC later when your code is tested).

Implementing CIC interpolation

The choice of interpolation scheme is a tradeoff between accuracy and computational expense. The CIC scheme is both relatively cheap and accurate and is most commonly used in PM codes. I will now describe how to implement it (you can start coding with the simpler NGP scheme and upgrade it to CIC later when your code is tested).

Consider the density assignment for a particle with coordinates $\{x_p, y_p, z_p\}$ The cell containing the particle will have indices of

$$i = [x_p]; \quad j = [y_p]; \quad k = [z_p],$$

where $[x]$ is the integer floor function (equivalent to Fortran's `int`). Let's assume that cell centers are at $\{x_c, y_c, z_c\} = \{i + \Delta x/2, j + \Delta x/2, k + \Delta x/2\}$, where Δx is the cell's size (in the internal code units chosen above $\Delta x = 1$, this will be implicitly assumed below). This is a matter of convention, but once chosen the convention should be consistent throughout the code.

In the CIC scheme, particle $\{x_p, y_p, z_p\}$ may contribute to densities in the parent cell (i, j, k) and seven neighboring cells[3]. Let's define

$$d_x = x_p - x_c; \;\; d_y = y_p - y_c; \;\; d_z = z_p - z_c;$$

$$t_x = 1 - d_x; \;\; t_y = 1 - d_y; \;\; t_z = 1 - d_z.$$

Contributions to the eight cells are then linear interpolations in 3D:

$$\begin{aligned}
\rho_{i,j,k} &= \rho_{i,j,k} + m_p t_x t_y t_z; \;\; \rho_{i+1,j,k} = \rho_{i+1,j,k} + m_p d_x t_y t_z; \\
\rho_{i,j+1,k} &= \rho_{i,j+1,k} + m_p t_x d_y t_z; \;\; \rho_{i+1,j+1,k} = \rho_{i+1,j+1,k} + m_p d_x d_y t_z; \\
\rho_{i,j,k+1} &= \rho_{i,j,k+1} + m_p t_x t_y d_z; \;\; \rho_{i+1,j,k+1} = \rho_{i+1,j,k+1} + m_p d_x t_y d_z; \\
\rho_{i,j+1,k+1} &= \rho_{i,j+1,k+1} + m_p t_x d_y d_z; \;\; \rho_{i+1,j+1,k+1} = \rho_{i+1,j+1,k+1} + m_p d_x d_y d_z;,
\end{aligned}$$

where m_p is particle mass. Doing this for all particles will result in the grid of densities $\rho_{i,j,k}$.

[3] Make sure you enforce periodic boundary conditions: $i = \mathrm{mod}(i, N_{g,1})$, etc. for j and k.

Solving the Poisson equation

With the grid of densities, $\rho_{i,j,k}$, in hand, the code can proceed to solve the discretized[4] Poisson equation

$$\begin{aligned}\tilde{\nabla}^2\tilde{\phi} &\approx \tilde{\phi}_{i-1,j,k} + \tilde{\phi}_{i+1,j,k} + \tilde{\phi}_{i,j-1,k} + \tilde{\phi}_{i,j+1,k} + \tilde{\phi}_{i,j,k-1} + \tilde{\phi}_{i,j,k+1} - 6\tilde{\phi}_{i,j,k} = \\ &= \frac{3}{2}\frac{\Omega_0}{a}(\tilde{\rho}_{i,j,k} - 1).\end{aligned}$$

Solving the Poisson equation

With the grid of densities, $\rho_{i,j,k}$, in hand, the code can proceed to solve the discretized[4] Poisson equation

$$\begin{aligned}\tilde{\nabla}^2\tilde{\phi} &\approx \tilde{\phi}_{i-1,j,k}+\tilde{\phi}_{i+1,j,k}+\tilde{\phi}_{i,j-1,k}+\tilde{\phi}_{i,j+1,k}+\tilde{\phi}_{i,j,k-1}+\tilde{\phi}_{i,j,k+1}-6\tilde{\phi}_{i,j,k}= \\ &= \frac{3}{2}\frac{\Omega_0}{a}(\tilde{\rho}_{i,j,k}-1).\end{aligned}$$

The discretization thus results in a large system of linear equations relating unknowns, $\tilde{\phi}_{i,j,k}$, to the known right hand side values. This system can be solved using FFT.

[4] It is customary in PM codes to discretize the Laplacian operator using the 7-point template.

In the Fourier space the Poisson equation is $\tilde{\phi}(\mathbf{k}) = G(\mathbf{k})\tilde{\delta}(\mathbf{k})$, where $G(\mathbf{k})$ is the Green function which for the adopted discretization is given by

$$G(\mathbf{k}) = -\frac{3\Omega_0}{8a}\left[\sin^2\left(\frac{k_x}{2}\right) + \sin^2\left(\frac{k_y}{2}\right) + \sin^2\left(\frac{k_z}{2}\right)\right]^{-1},$$

where $L = N_g$ is the box size in code units and

$$k_x = 2\pi l/L, \quad k_y = 2\pi m/L, \quad k_z = 2\pi n/L, \quad \text{for component } (l, m, n).$$

The singularity at $l = m = n = 0$ should be avoided by setting this component of potential, $\hat{\phi}_{000}$, by hand to zero.

$\Rightarrow$ Given a density field $\tilde{\delta}(\mathbf{r})$ in real space, we can solve for the gravitational potential by

$\Rightarrow$ Given a density field $\tilde{\delta}(\mathbf{r})$ in real space, we can solve for the gravitational potential by

- performing the FFT to get $\tilde{\delta}(\mathbf{k})$,

$\Rightarrow$ Given a density field $\tilde{\delta}(\mathbf{r})$ in real space, we can solve for the gravitational potential by

- performing the FFT to get $\tilde{\delta}(\mathbf{k})$,
- multiplying every element in the field by the corresponding value of $G(\mathbf{k})$ to get $\tilde{\phi}(\mathbf{k})$,

$$\hat{\phi}_{lmn} = G(\mathbf{k}_{lmn})\hat{\rho}_{lmn}, \text{where}$$

$$\hat{f}_{lmn} = (\Delta x)^3 \sum_{i,j,k=0}^{N_g-1} f_{ijk} e^{-i2\pi(il+jm+kn)/N_g},$$

$$f_{ijk} = \frac{1}{L^3} \sum_{l,m,n=0}^{N_g-1} \hat{f}_{lmn} e^{i2\pi(il+jm+kn)/N_g}.$$

$\Rightarrow$ Given a density field $\tilde{\delta}(\mathbf{r})$ in real space, we can solve for the gravitational potential by

- performing the FFT to get $\tilde{\delta}(\mathbf{k})$,
- multiplying every element in the field by the corresponding value of $G(\mathbf{k})$ to get $\tilde{\phi}(\mathbf{k})$,

$$\hat{\phi}_{lmn} = G(\mathbf{k}_{lmn})\hat{\rho}_{lmn}, \text{where}$$

$$\hat{f}_{lmn} = (\Delta x)^3 \sum_{i,j,k=0}^{N_g-1} f_{ijk} e^{-i2\pi(il+jm+kn)/N_g},$$

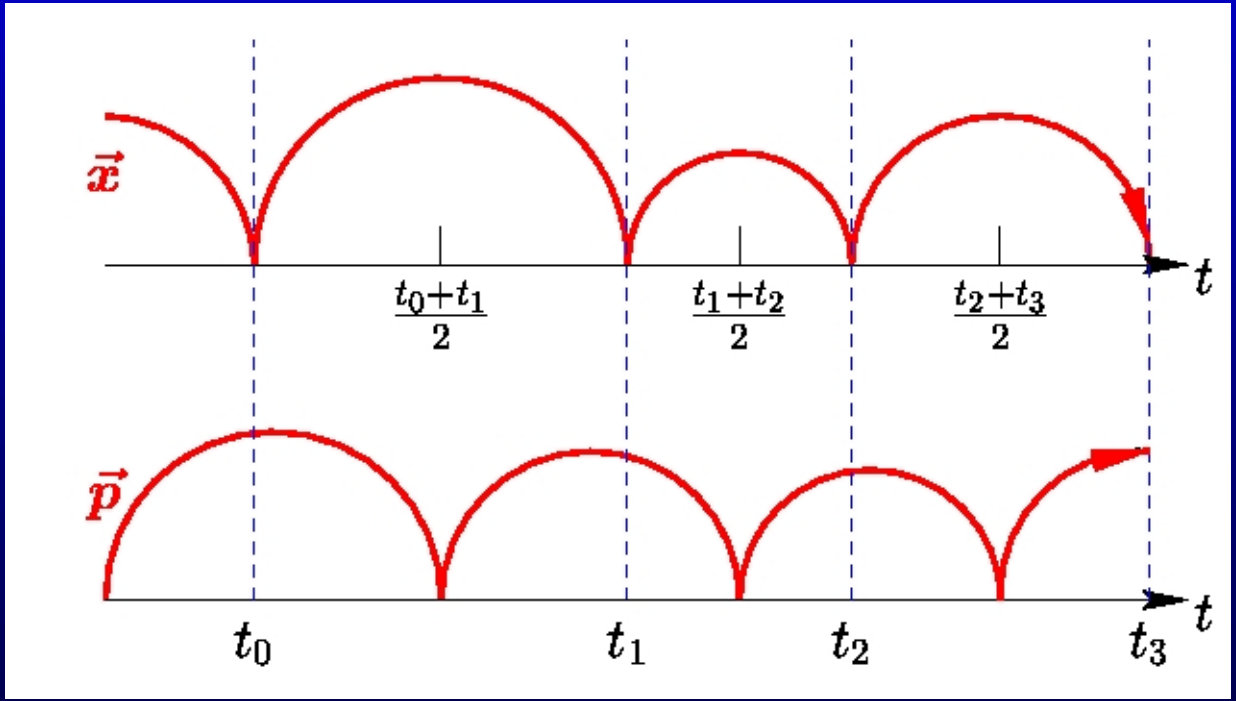
$$f_{ijk} = \frac{1}{L^3} \sum_{l,m,n=0}^{N_g-1} \hat{f}_{lmn} e^{i2\pi(il+jm+kn)/N_g}.$$

- transforming[5] the result back to real space to get $\tilde{\phi}(\mathbf{r})$ discretized at cell centers.

[5] Be careful about normalization of the FFT. The transforms $\tilde{\delta}(\mathbf{r}) \rightarrow \tilde{\delta}(\mathbf{k})$ and $\tilde{\delta}(\mathbf{k}) \rightarrow \tilde{\delta}(\mathbf{r})$ should recover the original field $\tilde{\delta}(\mathbf{r})$.

Updating particle positions and velocities

Let us assume we are using constant integration step in Δa and the second-order accurate *leapfrog* integration.



Schematic of the leapfrog integration for a variable time step. Note how velocities and particles are staggered in time. (adopted from M.Gross's PM notes).

After n time steps, $a_n = a_i + n\Delta a$, during the step $n+1$, we should have coordinates $\tilde{\mathbf{x}}_n$ at a_n and momenta $\tilde{\mathbf{p}}_{n-1/2}$ at $a_{n-1/2} = a_n - \Delta a/2$ from the previous step. Assigning density and solving the Poisson equation gives potential $\tilde{\phi}_n$ at a_n. For the assumed variables and units, positions and momenta are updated as follows:

$$\tilde{\mathbf{p}}_{n+1/2} = \tilde{\mathbf{p}}_{n-1/2} + f(a_n)\tilde{\mathbf{g}}_n\Delta a; \quad \tilde{\mathbf{x}}_{n+1} = \tilde{\mathbf{x}}_n + a_{n+1/2}^{-2} f(a_{n+1/2})\tilde{\mathbf{p}}_{n+1/2}\Delta a.$$

Here, $\tilde{\mathbf{g}}_n = -\tilde{\nabla}\tilde{\phi}_n$ is acceleration *at the particle's position*. This acceleration can be obtained by interpolating accelerations from the neighboring cell centers. The latter are given by

$$\tilde{g}^x_{i,j,k} = -(\tilde{\phi}_{i+1,j,k} - \tilde{\phi}_{i-1,j,k})/2, \quad \tilde{g}^y_{i,j,k} = -(\tilde{\phi}_{i,j+1,k} - \tilde{\phi}_{i,j-1,k})/2,$$

$$\tilde{g}^z_{i,j,k} = -(\tilde{\phi}_{i,j,k+1} - \tilde{\phi}_{i,j,k-1})/2.$$

For each component of the acceleration, you should use the same interpolation scheme as during density assignment. For a given particle the acceleration should be interpolated from the cells to which particle contributed density during density assignment step. *This is important!* Consistency in interpolation schemes ensures absence of artificial self-forces and the third Newton's law (Hockney & Eastwood 1981).

For the NGP scheme, acceleration for a particle is just the acceleration of its parent cell $\tilde{\mathbf{g}}_{i,j,k}$. For the CIC interpolation, we do the following (indices (i,j,k) here correspond to the parent cell of the particle; see section on density assignment).

$$\begin{aligned} g_p^x &= g^x_{i,j,k} t_x t_y t_z + g^x_{i+1,j,k} d_x t_y t_z + g^x_{i,j+1,k} t_x d_y t_z + g^x_{i+1,j+1,k} d_x d_y t_z + \\ & \quad g^x_{i,j,k+1} t_x t_y d_z + g^x_{i+1,j,k+1} d_x t_y d_z + g^x_{i,j+1,k+1} t_x d_y d_z + g^x_{i+1,j+1,k+1} d_x d_y d_z. \end{aligned}$$

And the same for g_y^p and g_z^p. $t_{x,y,z}$ and $d_{x,y,z}$ have the same definition as in the CIC density assignment described above.

After updating particle positions and velocities, the code should do some useful I/O and then proceed to step $n+2$.

Zeldovich approximation

- A first-order (linear) Lagrangian collapse model

  [Zeldovich 1970; see also Peacock's (p. 485) and Padmanabhan's (p. 294) textbooks for pedagogical introduction]

- Zeldovich approximation can be expressed by one equation:

$$\mathbf{x}(t) = \mathbf{q} + D_+(t)\mathbf{S}(\mathbf{q}),$$

it may look more familiar if you think about it as

$$\mathbf{x}(t) = \mathbf{x_0} + \mathbf{v}t; \quad \mathbf{v} = const$$

where

$\mathbf{q}$	is the initial position of a matter parcel (a particle in N-body simulations) and
$\mathbf{x}(t)$	is its position at time t (both $\mathbf{q}$ and $\mathbf{x}$ are *comoving*).
$D_+(t)$	is the linear growth function [see eq. (29) in Carrol et al. (1993) for a useful fitting formula] and
$\mathbf{S}(\mathbf{q})$	is a *time-independent* displacement vector.

- The evolution of density follows from conservation of mass $\rho(\mathbf{x})d^3\mathbf{x} = \bar{\rho}d^3\mathbf{q}$:

$$\rho(\mathbf{x},t) = \frac{\bar{\rho}}{\det(\partial x_j/\partial q_i)} = \frac{\bar{\rho}}{\det[\delta_{ij} + D_+(t) \times (\partial S_j/\partial q_i)]}$$

- The evolution of density follows from conservation of mass $\rho(\mathbf{x})d^3\mathbf{x} = \bar{\rho}d^3\mathbf{q}$:

$$\rho(\mathbf{x},t) = \frac{\bar{\rho}}{\det(\partial x_j/\partial q_i)} = \frac{\bar{\rho}}{\det[\delta_{ij} + D_+(t) \times (\partial S_j/\partial q_i)]}$$

- ZA is exact in 1D until the first crossing of particle trajectories occurs. This is because in 1D ZA describes collapse of parallel sheets of matter and acceleration of each sheet is independent of $\mathbf{x}$ until sheets cross.

- The evolution of density follows from conservation of mass $\rho(\mathbf{x})d^3\mathbf{x} = \bar{\rho}d^3\mathbf{q}$:

$$\rho(\mathbf{x},t) = \frac{\bar{\rho}}{\det(\partial x_j/\partial q_i)} = \frac{\bar{\rho}}{\det[\delta_{ij} + D_+(t) \times (\partial S_j/\partial q_i)]}$$

- ZA is exact in 1D until the first crossing of particle trajectories occurs. This is because in 1D ZA describes collapse of parallel sheets of matter and acceleration of each sheet is independent of $\mathbf{x}$ until sheets cross.

- Although ZA is only an approximation in 3D, it still works very well at the initial stages of evolution of cosmological density fields because flattened *pancake* structures (whose evolution is well described by ZA) are typical in such fields[6] [see, e.g., Bond et al. (1996)]. This makes ZA the method of choice for setting up initial conditions in cosmological simulations.

[6] Another explanation of the success of ZA is its use of displacement field, $S(q)$, as the basis for evolution model. Density depends on the derivatives of $S(q)$ so that small (linear) displacements can correspond to large density contrasts.

A test problem: 1D collapse of a plane wave

- Let's consider the evolution of a 1D sine-wave $S(q) = A\sin(kq)$:

$$x(a) = q + D_+(a)A\sin(kq); \quad k = 2\pi/\lambda;$$

- Let's assume we want to simulate a wave with wavelength equal to the simulation box size, $\lambda = L_{\rm box} = N_g$ using $N_{p,1}^3$ particles and N_g^3 grid cells in a cubic grid. In 3D, the 1D equations can be used to set up initial conditions for $N_{p,1}$ parallel sheets of particles. For some initial epoch, $a_{\rm ini}$, we have

$$x_i(a_{\rm ini}) = q_i + D_+(a_{\rm ini})A\sin\left(\frac{2\pi q_i}{L_{\rm box}}\right); \quad q_i = (i-1)\frac{L_{\rm box}}{N_{p,1}} \text{ for } i = 1, ..., N_{p,1};$$

- Take the time derivative of x to get equation for *peculiar* velocity $v = a\dot{x}$ (or momentum $p = av$, if needed):

$$v(x) = a\dot{D}_+(a_{\rm ini} - \Delta a/2)A\sin(kq),$$

where initialization for the leapfrog scheme is assumed.

- We can choose the first crossing epoch, say $a_{\rm cross} = 10a_{\rm ini}$. The epoch of the first crossing can then be identified as the epoch of the caustic formation (i.e., $\rho(x_{\rm cross}, a_{\rm cross}) = \infty$, where $x_{\rm cross} = q_{\rm cross} = L_{\rm box}/2$), which gives us the wave amplitude A:

$$\rho(x,a) = \frac{\bar{\rho}}{[1 + D_+(a) \times Ak\cos(kq)]}, \quad \Rightarrow A = -\left(D_+(a_{\rm cross})k\right)^{-1}.$$

- Positions $x(q,a)$ and velocities $v(q,a)$ is all we need to set up initial conditions for particles.

- The subsequent 1D evolution using the PM code can be tested by comparing x and v given by the above equations to the coordinates and velocities of particles at different epochs using outputs of your code (for example, you can plot phase diagrams, i.e. particles in $v - x$ or $v - q$ plane).

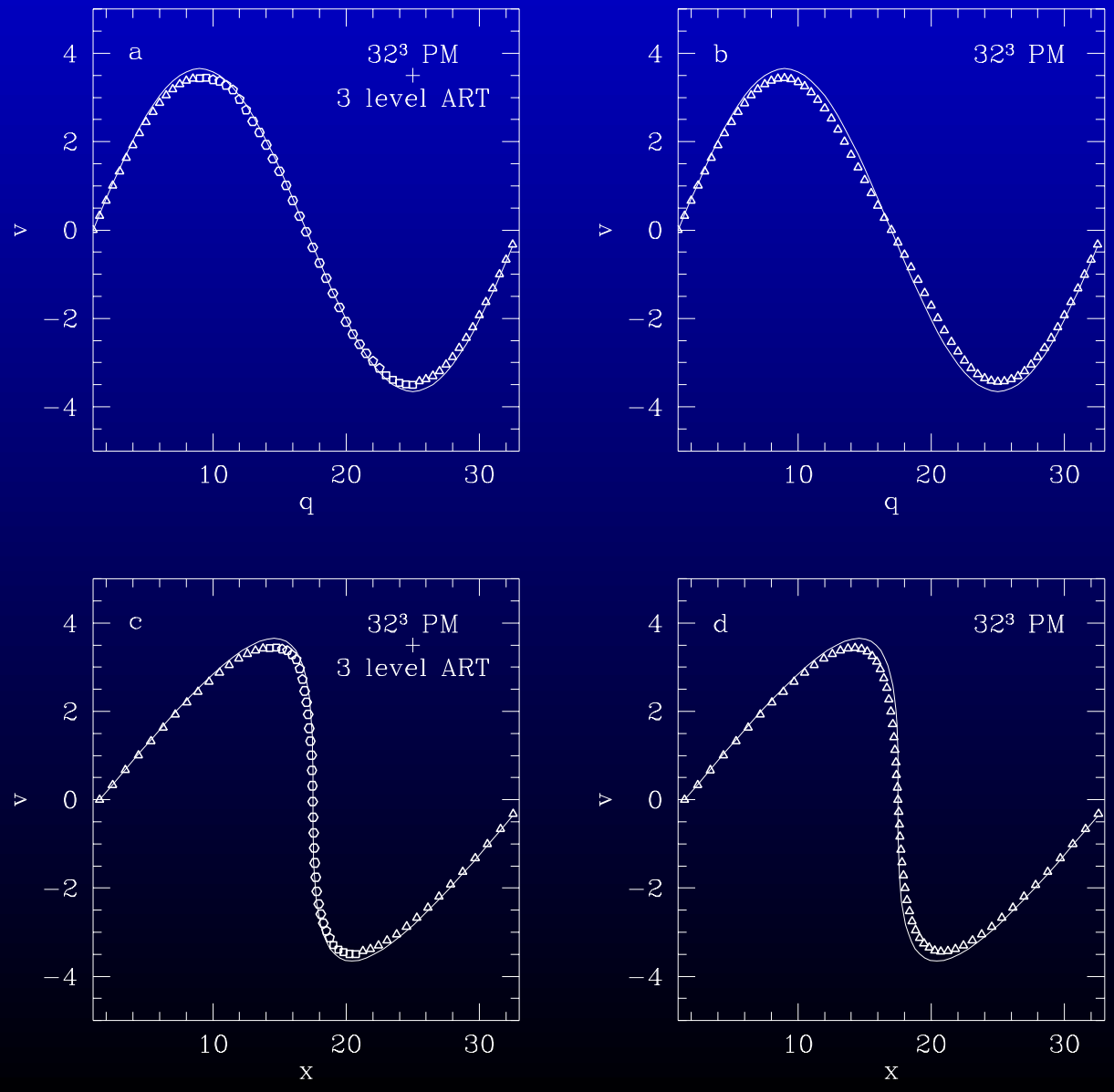
- In addition, you can use solutions for the gravitational potential and particle accelerations to test your gravity solver. For each grid cell:

$$\phi(x,a) = \frac{3}{2}\frac{\Omega_0}{a}\left\{\frac{q^2-x^2}{2} + D_+ \times Ak\left[kq\sin(kq) + \cos(kq) - 1\right]\right\};$$

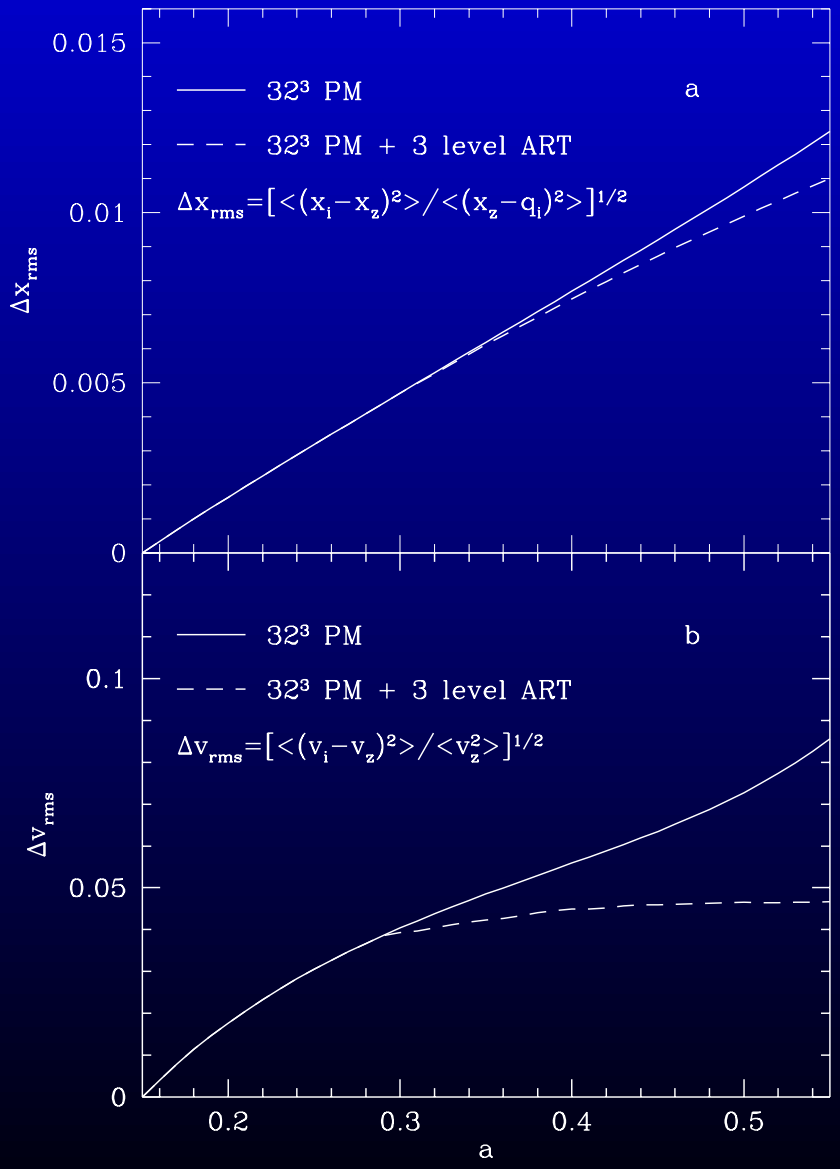
$$g(x,a) = -\frac{\partial\phi}{\partial x} = \frac{3}{2}\frac{\Omega_0}{a}(x-q) = \frac{3}{2}\frac{\Omega_0}{a}D_+(a)A\sin(kq)$$

The only difference from particles is that you do not know cell's Lagrangian coordinate q (q is known for particles if you maintain particles in the same order throughout evolution; particle's index gives q as we saw above). For a given expansion factor a, q of a cell can be calculated by solving equation $q = x - D_+(a)A\sin(kq)$ numerically, with x is coordinate of the cell.

Plane wave collapse test: phase diagram in Lagrangian coordinates q (a) the ART simulation with 32^3 base grid and 3 refinement levels and (b) the PM simulation with a 32^3-cell grid at the crossing time. *Solid line*, analytic solution; *polygons*, numerical results.(c,d) Corresponding phase diagram for physical coordinates x.



Plane wave collapse test: rms deviations of (a) coordinates and (b) velocities from analytic solution vs. the expansion parameter for the PM code (*solid line*) and for the ART code with three levels of refinement (*dashed line*) See, Efstathiou et al. (1985) for details on this and other tests.



Beyond the Zeldovich test: setting up cosmological initial conditions

If your code passes the Zeldovich test, you may try to run a realistic cosmological simulation. To do this, you will have to code a routine to set up initial conditions for the particles using a statistical realization of the power spectrum, $P(k)$, of your favorite cosmological model[7]. Fortunately, the basis for the algorithm is the now familiar ZA.

The displacement of a particle is now determined not by a single wave, but by the entire set of waves that can be represented numerically in the simulation box. Thus, particle's comoving coordinates and momenta, $p = a^2\dot{x}$, are given by

$$\mathbf{x} = \mathbf{q} - D_+(a)\mathbf{S}(\mathbf{q}); \quad \mathbf{p} = -(a - \Delta a/2)^2 \dot{D}_+(a - \Delta a/2)\mathbf{S}(\mathbf{q}),$$

where a is the initial expansion factor and Δa is its step[8], $\mathbf{q}$ is particle's unperturbed position.

[7] An alternative is to set up initial conditions using a public code.

[8] Note that the growth factor $D_+(a)$ is usually scale-independent (e.g., for all models with CDM only) and this is assumed here. This is not true, however, for some models, such as the Cold+Hot Dark Matter (CHDM).

The *displacement vector* $\mathbf{S}$ is given by the discrete Fourier transform (e.g., Padmanabhan 1993, p.294):

$$\mathbf{S}(\mathbf{q}) = \alpha \sum_{k_{x,y,z}=-k_{max}}^{k_{max}} i\mathbf{k}\; c_k \exp\left(i\mathbf{k}\cdot\mathbf{q}\right);$$

$$k_{x,y,z} = \frac{2\pi}{N_g} l,m,n; \quad l,m,n = 0,\pm 1,...,\pm N_{p,1}/2; \quad k^2 = k_x^2 + k_y^2 + k_z^2 \neq 0.$$

Here, α is the power spectrum normalization. The summation is over all possible wavenumbers from the fundamental mode with wavelentgh equal to the box size to the smallest “Nyquist” wavelength with the wavenumber of $N_{p,1}/2$. The real and imaginary components of the Fourier coefficients, $c_k = (a_k - ib_k)/2$ are independent gaussian random numbers with the mean zero and dispersion $\sigma^2 = P(k)/k^4$:

$$a_k = \sqrt{P(k)}\frac{Gauss(0,1)}{k^2}, \quad b_k = \sqrt{P(k)}\frac{Gauss(0,1)}{k^2}.$$

Note that c_k should satisfy condition $c_k = c^*_{-k} = (a_k - ib_k)/2$.

Thus, if we construct a realization of $\{k_{x,y,z}c_k\}$ using a cosmological power spectrum on a grid in the Fourier space, its discrete FFT gives us components of the real space displacement vector $\mathbf{S}(\mathbf{q})$ for all particles.

Applying these equations in practice for N_p particles on a cubic grid with N_g^3 cells amounts to

1) distributing particles uniformly in the simulation volume (e.g., placing particles in the centers of appropriately spaced grid cells).

2) Computing the displacement vector for each particle position. Construct a realization of $k_{x,y,z}c_k$ on three (each for one component of the wavevector) cubic grids. For example, for the x-component, the grid is initialized to $k_x c_k$ where k_x are running from $-k_{Ny}$ to k_{Ny} (assume $k_{Ny} = N_{p,1}/2$, k here is in units of the fundamental mode $2\pi/L_{box} = 2\pi/N_g^{1/3}$) and a_k and b_k are gaussian random numbers defined above.

3) FFT each of the three grids to get $S_x(q)$, $S_y(q)$, and $S_z(q)$ in real space. Apply ZA to displace particles from their Lagrangian positions ($\mathbf{q} \rightarrow \mathbf{x}$).

You will need

- Your favorite FFT solver, if you don't have one see *Numerical Recipes* (Ch. 12).
- A decent random number generator. Be careful here as you will need to generate fairly long sequences of random numbers (I can supply you with a RNG, if needed). The gaussian random pairs can be generated from the uniformly distributed numbers using the Box-Muller method (*Numerical Recipes, Ch. 7*).
- A routine returning $P(k)$ for a specific cosmological model. See Hu & Sugiyama 1996 and Eisenstein & Hu 1999 for useful analytical approximations to $P(k)$.

You can start with a simulation of $\Omega_0 = 1$ CDM universe. Evolution in this model is simple: $D_+(a) = (a/a_0)$. You can check D_+ by computing the two-point correlation function of DM particles at different epochs. If you get to this point, you can then simulate open CDM and flat ΛCDM models to see how the evolution of structures changes.

PM in cosmology: some historical references

- Efstathiou G. & Eastwood J. 1981, MNRAS **194**, 503
- Klypin A.A. & Shandarin S.F. 1983, MNRAS **204**, 891
- Centrella J. & Melott A.L. 1983, Nature **305**, 196-198
- Miller R.H., 1983, ApJ **270**, 390-409
- Efstathiou G., Davis M., White S.D.M., & Frenk C.S. 1985, ApJS **57**, 241-260

Papers with useful info on PM

- Hockney, R. W., and Eastwood, J. W. 1981, *"Computer Simulation Using Particles"*, McGraw-Hill, New York
- Klypin A.A. & Shandarin S.F. 1983, MNRAS **204**, 891
- Efstathiou G., Davis M., White S.D.M., & Frenk C.S. 1985, ApJS **57**, 241-260
  [Review and comparison of cosmological N-body methods]
- Sellwood J.A. 1987, ARA&A **25**, 151
  [Review of particle simulation methods]
- Description of Hugh Couchman's AP^{3}M code
- Bertschinger E. 1998, ARA&A **36**, 599
  [The most recent review of numerical techniques used in cosmological simulations and algorithms for setting up the initial conditions.]

- Michael Gross's, PhD Thesis
  [useful descriptions of PM and an algorithm for setting up ICs in Ch. 2 and Appendix A]

- Klypin, A. & Holtzman, J. 1997, `astro-ph/9712217`
  "Particle-Mesh code for cosmological simulations"

Web links

- Amara's Recap of Particle Simulation Methods: collection of info and links on various N-body algorithms
- Anatoly Klypin's public PM package for cosmological simulations.
- A set of PM lecture slides by M.A.K. Gross (UCSC). The page includes a sample 1D PM code.
- Useful fortran and C (requires OpenGL) codes for visualization of particle distributions in 3D.