Chapter 11

Cosmological Simulations Using Particle-Mesh Methods

H. M. P. Couchman

11.1 Introduction

The purpose of cosmological simulations is to model the growth of structure in the universe. The techniques described here model the mass density of the universe with a distribution of softened particles. Evolution is simulated by following the trajectories of these particles under their mutual gravity. Since the region of the universe modelled is typically very much smaller than the horizon volume, Newtonian equations of motion are adequate to describe the particle dynamics and evolution of structure. The resulting numerical schemes are straightforward and robust. An alternative approach is to describe the mass distribution as a fluid and calculate the evolution from the appropriate fluid equations (e.g., Peebles 1987). A number of other analytic and semi-analytic approaches to gravitational evolution have been discussed. The description of the growth of small perturbations in the matter distribution is well established, both to first order and at higher orders (see, for example, Peebles 1980). Many observational constraints, however, are the result of highly non-linear gravitational evolution for which perturbation analysis is of limited use. Numerical N-body techniques offer a simple and effective tool for investigating non-linear cosmological gravitational evolution.

Many of the results arising from potential theory and the theory of gravitating systems of particles, as discussed in this volume and elsewhere, are directly applicable to the cosmological situation and will not be addressed here in further detail. The principal features peculiar to cosmological simulations arise from the construction of an initial particle distribution, the necessity of describing evolution in an expanding cosmic background and the choice of suitable boundary conditions. The remainder of the introduction addresses these issues and describes the popular mesh-based cosmological particle methods. Two computer codes are discussed in this chapter: the first is used to generate an initial distribution of particles; the second to evolve it. The algorithms are described in Sect. 11.2 and the FORTRAN77 implementation in Sect. 11.3. The reader interested in trying the codes immediately may skip to Sect. 11.4 where a test case is presented.

11.1.1 Boundary Conditions

Cosmological simulations almost universally model evolution within a cubic (or sometimes cuboidal) region with triply periodic boundary conditions. The rationale is that this is the most realistic way to simulate evolution within a small comoving region of an effectively infinite universe. An advantage is that edge effects are completely eliminated. Further, the equations of motion are straightforward in comoving coordinates. The main drawback, compared with vacuum boundary conditions, is that the pairwise force reflects the symmetry of the cube for separations approaching half the box size. Although this is the correct force for a periodic system, the universe is unlikely to be periodic on the scales we wish to simulate. If care is not taken with the choice of initial conditions and their evolution, this artificial periodicity leads to undesirable results.

A secondary motivation for embracing periodic boundary conditions is that solving for the potential on a mesh using the Fast Fourier Transform (FFT) leads automatically to a periodic potential as a result of the discrete representation of modes in Fourier space. It should be noted, however, that cosmological simulations are not restricted to using the FFT. Hernquist, Bouchet and Suto (1991) describe a method for achieving periodic boundary conditions when the gravity solver is a tree code.

11.1.2 Initial Conditions

A wide range of cosmological models may be represented by the evolution of a particle distribution. Here we consider models in which structure evolves from a spectrum of low amplitude perturbations in an otherwise homogeneous matter density – the standard cosmological model. Further, we suppose that the density field is a Gaussian random process. The program provided here sets up an approximation to such a field.

Periodic boundary conditions imply discrete sampling in Fourier space. Further, discrete representation of the density field restricts the range of wavenumbers in Fourier space to that allowed by the Nyquist theorem. Fluctuations in the particle distribution are represented by a population of modes with independent random phases in this finite, discretely sampled Fourier space. On scales which are small compared with the box size, the Central Limit Theorem guarantees an approximately Gaussian density distribution. This will not be true on scales approaching the box size where the density of modes is sparse. A Gaussian distribution may be approximated on large scales only by averaging over an ensemble of distributions in which mode amplitudes are drawn from a Gaussian variate. The spectral truncation imposed by the periodic boundary conditions may lead to a particle distribution which is a poor representation of the universe we wish to model.

The unperturbed initial particle distribution may sample the homogeneous mass density of the universe in a range of ways from random to uniform. Random initial sampling introduces Poisson noise on small scales. It has been suggested that a uniform distribution may lead to unwanted coherence in the subsequent evolution. It is likely that small-scale discreteness quickly erases memory of the the initial small-scale particle distribution.

11.1.3 Evolution in the Standard Cosmological Model

Periodic boundary conditions mesh naturally with a Fourier description of cosmic structure evolution. If the initial conditions are described by a low amplitude fluctuation spectrum the correspondence is strengthened. Linear evolution of small perturbations is well understood (e.g., Peebles 1980). Comoving Fourier modes grow independently until their amplitudes approach the mean density. Subsequently, modes couple to nearby wavenumbers in Fourier space. Thus, evolution which drives the longest waves in a periodic box non-linear cannot be correctly modelled. Small-scale objects which have virialised prior to large-scale non-linearity may be correctly modelled, although the degree to which this is true is hard to quantify.

Setting the physical scale of the cube involves a compromise between small-scale resolution and large-scale power. A first order correction for the power missing on scales larger than the box is to modify the expansion rate of the background to mimic the evolution of a larger region embedding the cube. This cannot, however, model large-scale shear fields and still gives incorrect non-linear evolution for the largest waves. Depending upon the initial spectrum and the resolution required, it may be more realistic to use a method which simulates the region in isolation with imposed external fields. Spectral truncation on small scales corresponds to a smoothing of the initial distribution. A corresponding smoothing must be employed in the evolution if particle dynamics are not to be dominated by discreteness effects on small-scales. Particle ‘softening’ is discussed in the next section.

Ultimately it is the form of the power spectrum which determines the significance of the issues presented above and which sets the realistic range of scales over which periodic particle methods are useful.

11.1.4 Particle–Mesh Methods in Cosmology

Particle–Mesh methods have many advantages for cosmological simulations. Rectangular meshes may be trivially repeated for periodicity and, as noted above, if the FFT is used to solve for the potential, periodicity results automatically. A second advantage is that the memory requirement for a scheme with mesh size equal to the mean interparticle spacing is only 30%–60% greater than the essential storage for particle positions and velocities. Further, rapid solution methods – such as the FFT – exist for rectangular meshes.

Calculating interparticle forces with a regular mesh results in an equivalent minimum particle softening of roughly two mesh spacings. (This limit is set by the Nyquist frequency of the mesh.) This is the basic Particle-Mesh (PM) calculation. It is suitable for the study of many *collisionless* phenomena, such as the dynamics of galaxies and, in the context of cosmology, for the study of the topology of large-scale structure. Modelling of *collisional* systems, such as the clustering and dynamics of point-mass galaxies, requires a softening significantly smaller than the mean interparticle spacing. It is generally uneconomical to provide the required force by resorting to increasingly fine meshes. The Particle-Particle–Particle-Mesh (P^3M) method described by Hockney & Eastwood (1981) provides a feasible alternative. This augments the mesh force on a particle with a short range component summed directly from near neighbours.

An important use of P^3M in cosmology is in the simulation of collisionless systems in an expanding background. Typically, the softening of the pairwise force is fixed in physical units and is chosen to be comparable initially to the mean interparticle spacing. As the simulation evolves, overdense regions in the particle distribution break away from the cosmic expansion and collapse. Regions which are virialised and of constant physical size shrink in comoving coordinates. P^3M attempts to match the force softening to the interparticle separation *within* these bound objects. This is necessary to maintain tightly bound objects. In practice this technique is ideal only for groups of particles forming at a particular epoch, unless a softening varying from group to group is employed. Given the relatively limited dynamic range of many simulations it is a useful technique.

Ironically, the particle–particle (PP) sum, which makes P^3M attractive by providing sub-mesh scale resolution, is also the method's main drawback. The PP calculation must search for neighbours out to roughly two mesh spacings to properly augment the PM force. The number of particles within this search distance grows as clustering develops, significantly slowing calculation of the PP sum. The simplest extension to P^3M is to use a finer mesh in regions where the particle density is high, in order to reduce the number of particles per mesh cell back to unity. The Adaptive P^3M technique (AP^3M) is built around the standard P^3M algorithm, which itself consists of the PM cycle followed by the direct particle-particle sum. AP^3M produces fully equivalent forces to P^3M but represents a more efficient implementation of the force splitting idea of P^3M. The AP^3M program may be used in any of the three modes with an appropriate choice of input parameter.

All of these methods are derived from techniques described or developed by Hockney and Eastwood (1981). Although these grid based methods were initially developed for simulations of plasmas, the P^3M program requires only minor modification to be used for simulations of gravitational clustering. The first large-N cosmological studies using P^3M were undertaken by Efstathiou & Eastwood (1979). Subsequently Efstathiou et al. (1985) described modifications to the standard code and discussed various technical matters relating to cosmological simulations including integration of the equations of motion. They also discussed a technique for generating an initial fluctuation spectrum. Some refinements to this technique are described in Carlberg & Couchman (1989). The AP^3M algorithm was described by Couchman (1991).

11.2 Particle–Mesh Methods: Numerical Techniques

The aim of cosmological particle codes is to follow the evolution of a large number of interacting particles under Newtonian gravity. Realistic simulations of the formation of cosmic structure require a very large dynamic range, from subgalactic scales to supercluster scales; a range of at least 10^9 in mass. Current simulations follow the evolution of only $\sim 10^4$–10^7 particles. Many of the numerical methods and programming strategies described here are aimed at maximizing the number of particles that may be used for given computer resources.

11.2.1 Time integration

The equations of motion governing the evolution of the particle distribution in an expanding cosmological model are most conveniently cast in terms of comoving coordinates $\mathbf{x} = \mathbf{r}/a(t)$, where $\mathbf{r}$ is the proper coordinate and a is the expansion parameter. The equation of motion of the i-th particle is then

$$\frac{d\mathbf{v}_i}{dt} + 2\frac{\dot{a}}{a}\mathbf{v}_i = -\frac{G}{a^3}\sum_{j\neq i,J} m_j \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|^3} = \frac{\mathbf{f}_i}{a^3}, \tag{11.1}$$

where $\mathbf{v} = \dot{\mathbf{x}}$ and m_j is the mass of the j-th particle (Peebles 1980). In a periodic box the sum is over all $N-1$ particles, $j \neq i$, and their images, J. This force assumes point mass particles and should strictly speaking be convolved with the particle shape to allow for softening.

Time-centred leapfrog integration is used. This is chosen primarily because it requires no extra storage with new velocities overwriting old. It may be easily replaced by a higher order scheme if desired. Predictor–corrector schemes are particularly useful for the gravitational N-body problem. In order to allow some additional flexibility in the integration scheme the independent variable $p \propto a^\alpha$ is introduced (cf. Efstathiou et al. 1985). The optimum choice of the parameter, α, will depend upon the problem under investigation. With constant step size Δp, leapfrog offers $O(\Delta p^3)$ accuracy. For simplicity, we consider an Einstein–de Sitter universe throughout, with $a \propto t^{2/3}$. Equation (11.1) becomes

$$\frac{d\mathbf{v}_i'}{dp} + \left(1 + \frac{1}{2\alpha}\right)\frac{\mathbf{v}_i'}{p} = \left(\frac{3}{2\alpha p}\right)^2 \mathbf{f}_i, \tag{11.2}$$

where $\mathbf{v}' = d\mathbf{x}/dp$.

In the numerical scheme a convenient scaling is $p = (3/2\alpha)a^\alpha$ and then $\mathbf{v} = (a^\alpha/t)\mathbf{v}'$. If $a = t = (2\alpha/3)p = 1$ at the start of the simulation, then $\mathbf{v}'$ is the proper peculiar velocity, $a\dot{\mathbf{x}}$, at $t = 1$.

The leapfrog integration is set up as follows. Positions are given, and forces calculated at the half step and velocities at each full step. This is convenient for starting the scheme and for the output of data when positions and velocities are given or required at the same time. The discrete equations for updating positions to the $n+1/2$-th step and velocities to the $n+1$-th step are then

$$\begin{aligned} \mathbf{x}_{n+1/2} &= \mathbf{x}_{n-1/2} + \mathbf{v}_n' \Delta p, \\ \mathbf{v}_{n+1}' &= \mathbf{v}_n' \frac{1 - H(p)\Delta p}{1 + H(p)\Delta p} + \mathbf{f}_{n+1/2} \frac{9/(2\alpha p)^2}{1 + H(p)\Delta p} \Delta p, \end{aligned} \tag{11.3}$$

where $H(p) = (1 + 1/2\alpha)/2p$ and p is given at the half step. (Particle subscripts have been omitted for clarity.)

The correct form for energy conservation in comoving coordinates in an expanding world model is the Layzer–Irvine equation (here expressed in terms of a, the expansion factor):

$$\frac{d(a^2 T)}{da} + a\frac{dU}{da} = 0, \tag{11.4}$$

where $T = a^2/2 \sum_i m_i v_i^2$ and $U = -G/2 \sum m_i m_j / |\mathbf{x}_i - \mathbf{x}_j|$. The potential energy sum, U, is over all i, j, $j \neq i$ and the periodic images. The proper potential energy is U/a. The quantity $a\mathbf{v}_i$ is the proper peculiar velocity. (See Peebles (1980) for full details.) Equation (11.4) is strictly correct only for constant comoving particle shape but is adequate for the present discussion.

Equation (11.4) may be integrated by parts to give either $a^2T + aU - \int U da = (T+U)|_{a=1}$ or $aT + U + \int T da = (T+U)|_{a=1}$. Either of these equations may be monitored as an accuracy check during the integration of the discrete equations. Global conservation of energy, however, falls well short of a sufficient condition for numerical accuracy.

In an Einstein–de Sitter universe the density has the critical value $\rho = 3(\dot{a}/a)^2/8\pi G$. In terms of the numerical model in which N equal mass particles are distributed in a periodic cube with volume L^3 this corresponds to setting $m_i = 1$ and $G = L^3/6\pi N$ in the above equations.

11.2.2 Mesh Force calculation

Mesh (or grid) forces are calculated using the following steps:

1. Particles are assigned to the mesh with an appropriate smoothing function or kernel, U, to produce a discrete density distribution.
2. The density distribution is convolved with the Green's function, G, appropriate for the required pairwise particle interaction using FFTs. This produces a potential at mesh points.
3. The potential is differenced to obtain forces at mesh points. The differencing operator is denoted by $\mathbf{D}$.
4. Forces are interpolated back onto the particles using the same kernel, U, as in 1. above. Use of the same kernel results in a momentum conserving force provided the differencing operator used in 3. is anti-symmetric and calculates forces on the same mesh as in 1.

Since each of the steps above involves a convolution it is most convenient to perform the analysis in Fourier space. Following the notation of Hockney and Eastwood (1981), the periodic mesh force on a particle at $\mathbf{x}_i$ due to a particle at $\mathbf{x}_j$ is given by

$$\mathbf{F}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{L^3} \sum_{\mathbf{k}} \tilde{\mathbf{D}}(\mathbf{k}) \tilde{G}(\mathbf{k}) \sum_{\mathbf{n}} \tilde{U}(\mathbf{k_n}) e^{i\mathbf{k_n}\cdot\mathbf{x}_i} \sum_{\mathbf{m}} \tilde{U}(\mathbf{k_m}) e^{-i\mathbf{k_m}\cdot\mathbf{x}_j}. \tag{11.5}$$

The symbol ˜ indicates the continuous Fourier transform of a quantity. The first sum is over all $\mathbf{l}k_0$ within the principal zone of the discretely sampled periodic mesh with fundamental wavenumber $k_0 = 2\pi/L$. L is the number of mesh points in each dimension. The vector $\mathbf{l}$ is an integer triple (l_1, l_2, l_3). In the principal zone $-L/2 < l_i \leq L/2$, which corresponds to wavenumbers satisfying, in each dimension, $-\pi < k_i \leq \pi$. These are the Fourier components that can be represented by the FFT on an $L \times L \times L$ mesh. The functions $\tilde{\mathbf{D}}$ and $\tilde{G}$ are periodic in Fourier space by construction. The two final sums represent the alias sums over the harmonic content of U (and, in general, of the density distribution) which cannot be represented by the discrete mesh. The vector triples in these sums, $\mathbf{n}$ and $\mathbf{m}$, range over all integers and $\mathbf{k_n} = \mathbf{k} - 2\pi\mathbf{n}$. Practically, these sums may be restricted to the first or second alias by a suitably smooth choice of the assignment function U.

The Green's function $\tilde{G}$ is chosen to optimize the match of the mesh force, $\mathbf{F}$, to the desired force by minimizing the effects of the discrete mesh for a given smoothing kernel and differencing operator. This minimizes translational and directional variations of the pairwise interparticle force. For PM these considerations may be unimportant since any inaccuracy will occur on the softening scale and will correspond to non-spherical softened particles, non-central forces at small separations and fluctuations in the force as particles move across the mesh cells. For $\mathrm{P^3M}$, where we wish to model structure on sub-mesh scales by augmenting the mesh force with a short range component, translational invariance of the pairwise force is essential. Arranging for the mesh force to be spherically symmetric at small scales simplifies computation of the required short-range matching force, since only the separation radius need be known between particles and not the directional components.

The required force should be sufficiently smooth that it can be adequately represented by the discrete mesh. The force chosen corresponds to that between two finite spherically symmetric density clouds with shape given by $\rho(r) = (48/\pi a^4)(a/2 - r)$ for $r < a/2$, where a is the smoothing length. Following Hockney and Eastwood (1981) this shape is denoted by S_2. Since the highest frequency that can be represented by a discrete mesh (the Nyquist frequency) corresponds to a wavelength of two mesh spacings only forces corresponding to S_2 clouds with $a \gtrsim 2$ can be adequately reproduced. If we denote the Fourier transform of the S_2 shape by $\tilde{S}_2$, then the transform of the required force, $\mathbf{R}$, is given by $\tilde{\mathbf{R}} = -i\mathbf{k}\tilde{S}_2^2/k^2$.

The Green's function optimization technique is described in detail by Hockney & Eastwood (1981). The Green's function $\tilde{G}(\mathbf{k})$ is chosen to minimize the total square deviation of the mesh force $\mathbf{F}(\mathbf{x}_1, \mathbf{x}_2)$ from the required force $\mathbf{R}(|\mathbf{x}_1 - \mathbf{x}_2|)$. The integrated square difference in real space is expressed in terms of a sum over Fourier components using Parseval's theorem for a periodic system. The resulting optimal Green's function is

$$\tilde{G}(\mathbf{k}) = \frac{\tilde{\mathbf{D}}(\mathbf{k}) \cdot \sum_{\mathbf{n}} \tilde{U}^2(\mathbf{k_n})\tilde{\mathbf{R}}(\mathbf{k_n})}{|\tilde{\mathbf{D}}(\mathbf{k})|^2 \left[\sum_{\mathbf{n}} \tilde{U}^2(\mathbf{k_n})\right]^2}. \tag{11.6}$$

The assignment function used is the product in each dimension of a 'Triangular Shaped Cloud' (TSC): $U(\mathbf{x}) = u(x)u(y)u(z)$ where

$$u(x) = \begin{cases} \frac{3}{4} - x^2 & |x| \le \frac{1}{2}; \\ \frac{1}{2}\left(\frac{3}{2} - |x|\right)^2 & \frac{1}{2} \le |x| \le \frac{3}{2}; \\ 0 & \text{otherwise.} \end{cases} \tag{11.7}$$

The Fourier transform is $\tilde{u}(k) = \left(\sin\frac{k}{2}/\frac{k}{2}\right)^3$. The product construction avoids the necessity of calculating radius vectors for a spherically symmetric interpolation scheme. The triangular shaped cloud requires interpolation to three grid points in each dimension. Smoother, higher order schemes provide greater accuracy but the cost rises rapidly in three dimensions. It is computationally cheaper to use 'interlacing' than go to the next order scheme which requires interpolation to 64 grid points as compared with 27 for the TSC. Interlacing averages the forces from a second PM calculation with particle coordinates displaced by $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$.

A 10 point differencing operator, $\mathbf{D}$, is used. The x component is

$$
\begin{aligned}
D_x(i,j,k) = {} & \tfrac{1}{6}(\phi_{i-1,j,k} - \phi_{i+1,j,k}) \\
& + \tfrac{1}{12}(\phi_{i-1,j+1,k} - \phi_{i+1,j+1,k} \; + \; \phi_{i-1,j-1,k} - \phi_{i+1,j-1,k} \\
& \quad + \phi_{i-1,j,k+1} - \phi_{i+1,j,k+1} \; + \; \phi_{i-1,j,k-1} - \phi_{i+1,j,k-1}),
\end{aligned}
\tag{11.8}
$$

where ϕ_{ijk} is the potential at grid points returned by the FFT convolution. This difference operator is very much more accurate than the most straightforward 2 point differencing scheme, although more computationally expensive. The operator is also more accurate than the 8 point operator calculated from a mesh offset by a half grid unit in each dimension – which does not conserve momentum – whilst being only marginally more expensive. A further improvement in accuracy is possible if the forces are calculated directly using the FFT. Analogous techniques are used to derive an optimal estimate of the mesh potential energy. The optimal Green's function requires modification for the isolated mesh calculation. A complete description of the PM cycle is given in the book by Hockney and Eastwood (1981).

11.2.3 Particle-Particle Sum

The PM algorithm described above produces a spherically symmetric pairwise mesh force with minimum errors for a given softening. The direct particle-particle sum increments this force with a short range component which is non-zero only for small separations, where the required overall force cannot be reproduced by the PM calculation. The $\mathrm{P^3M}$ algorithm is a compromise between accuracy and efficiency. Small softenings in the PM calculation lead to larger errors as the aliasing of high frequency components in the required force increases. The advantage of a small softening is that the particle-particle pairwise interaction is non-zero over smaller separations, thus reducing the number of neighbours to be summed. Although improvements to the mesh force may increase the cycle time of the mesh calculation the reduction in cost of the direct sum rapidly compensates as clustering develops. The fact that gravity tends to clump particles makes virtually any improvement to the mesh force worthwhile. This is in contrast with plasma simulation for which the detailed timing balance between the mesh calculation and the direct sum is more relevant. It should be noted that there is essentially no penalty in improving the Green's function as it is calculated only once and then stored.

The mesh force produces a pairwise force at separation r with modulus which approximates $R(r/a_m)$, where a_m is the mesh softening. If the overall pairwise force required is $F(r)$, then the direct sum must add a force

$$
F_{\mathrm{PP}} = F(r) - R(r/a_m) \tag{11.9}
$$

from each neighbour at separation $r < r_c(a_m)$. For simplicity, we take $F(r) = R(r/a_s)$ with $a_s < a_m$, where a_s is the required overall softening. For the S_2 shaped particles, taking r_c to lie in the range $(0.7–0.8)a_m$ gives an error in the force which is similar to the mesh-force error for $2.5 \lesssim a_m \lesssim 3.5$. The requirement for the PP calculation is then to compute efficiently the force in Eq. (11.9) for all particle pairs with separation $r < r_c$. This is achieved as follows:

1. The particle distribution is partitioned into a mesh of cubic boxes of side r_c. This mesh will be called the 'chaining mesh.' The short-range cutoff, r_c, is chosen such that there is an integral number of cells in each dimension of the simulation cube.
2. For each particle in chaining-mesh cell I, the total short-range force is accumulated by adding force contributions, according to Eq. (11.9), from all other particles in cell I and its 26 neighbouring cells.

The pairwise potential energy from near neighbours – needed to augment the mesh energy – is accumulated concurrently with the force evaluation.

11.2.4 Refined Mesh Calculation

Organizing the particles into cells in the P^3M algorithm allows easy identification of neighbours which leads to the efficiency of the particle-particle sum. The calculation becomes very slow once clusters develop since the number of close pairs is large. Improvements in speed may be achieved by reducing a_m, and hence r_c, provided the accuracy of the mesh scheme remains acceptable. In practice a minimum chaining-cell size of approximately two potential mesh units may be achieved. The force error rises rapidly for smaller softenings. In simulations in which heavy clustering develops it quickly becomes more cost effective to replace the direct sum in heavily clustered regions by a further, local, P^3M calculation with a finer mesh. This is the AP^3M algorithm. The main steps are as follows:

1. Isolate regions for which the gravity would be more cheaply computed by a refined mesh. It is convenient to isolate patches which cover an exact cubic block of chaining-mesh cells.
2. Do the standard P^3M cycle on all particles except omit the direct sum for all particle pairs which lie in one of the regions isolated in 1. above. At the end of this step the full force has been accumulated on all particles with the exception of isolated self forces on particles in individual patches which have only been subject to the mesh force $R(r/a_m)$.
3. Do a further P^3M calculation on those regions found in 1. but with a force that reproduces only the short range part of the standard P^3M force. Allow for further refinement of these new grids.

The PM part of the refined P^3M calculation in Item 3. must reproduce a force $R(r/a_r) - R(r/a_m)$ for correct matching to the original mesh calculation in 1. Here a_r is the refined-mesh softening. The PM calculation here differs from the initial PM calculation in that the particle distribution in each refined patch must be treated as an isolated group of particles. This requires that the density data be extended by a factor of two in each dimension and padded with zeros for the correct isolated potential to be reproduced by the FFT convolution (Eastwood & Brownrigg 1979). The final PP sum is then accumulated for each individual patch of particles. The required PP pairwise force is now $R(r/a_s) - R(r/a_r)$. The overall effect is that the short-range cutoff $r_c(a_r)$ is very much smaller than in the initial PP calculation. The chaining mesh is finer and the number of neighbouring pairs is reduced, significantly improving performance in heavily clustered regions.

11.2.5 Initial conditions

Zel'dovich's (1970) approximation provides a simple way of imposing a spectrum of fluctuations on a particle distribution. The approximation maps the initial, Lagrangian, coordinate $\mathbf{q}$ of a particle to its perturbed, Eulerian, comoving coordinate $\mathbf{x}$ as follows:

$$\mathbf{x} = \mathbf{q} + b(t)\mathbf{s}(\mathbf{q}), \tag{11.10}$$

where $\mathbf{s}$ is the 'displacement field' and b describes the growing mode of linear fluctuations. For the Einstein–de Sitter model described here, linear modes grow proportionally to the expansion, hence $b = a$. Equation (11.10) implies $\mathbf{v} = \dot{\mathbf{x}} = \dot{a}\mathbf{s}(\mathbf{q})$. Substitution in Eq. (11.1) gives $\mathbf{s}(\mathbf{q}) = \lim_{t\to 0}(3/2a)\mathbf{f}$. Thus, displacements may be achieved by using the same techniques as are used to generate the mesh force. Corrections are made to allow for the discretisation used in the calculation of $\mathbf{s}$ from a given power spectrum. These compensate for the differencing used to obtain the displacement field and for the interpolation of displacements at grid points onto the particles. The same operators, U and $\mathbf{D}$, are used as in the mesh calculation described above. The displacement is given by

$$\mathbf{s}(\mathbf{q}) = \frac{1}{L^3}\sum_{\mathbf{k}} \tilde{\mathbf{D}}(\mathbf{k})\tilde{Z}(\mathbf{k})\tilde{\delta}(\mathbf{k}) \sum_{\mathbf{n}} \tilde{U}(\mathbf{k_n}) e^{i\mathbf{k_n}\cdot\mathbf{q}}. \tag{11.11}$$

The sums are as in Eq. (11.5). The modes used to populate the discrete Fourier transform space satisfy $\tilde{\delta}(\mathbf{k}) = \sqrt{P(k)}\, e^{i\phi_{\mathbf{k}}}$, where P is the power spectrum and $\phi_{\mathbf{k}}$ is a random phase. The function Z is the optimal Green's function for the Zel'dovich displacements: $\tilde{Z}(\mathbf{k}) = 1/\big[\mathbf{k}\cdot\tilde{\mathbf{D}}(\mathbf{k})\sum_{\mathbf{n}}\tilde{U}(\mathbf{k_n})\big]$. This expression is correct for small displacements from an initially uniform distribution of particles with the *same spacing as the Fourier grid*. The expression requires modification for random initial conditions or if the particle spacing is less than the grid spacing. It is incorrect to impose frequencies higher than the particle Nyquist frequency, as would occur if this expression were used for particle spacings larger than the grid spacing. Finally, it should be stressed that although Eq. (11.11) can give a discrete power spectrum which closely approximates the desired spectrum, $P(k)$, over the range of frequencies represented by the grid, it may not result in a particle distribution which is representative of the matter distribution in an infinite universe with the same spectrum.

Once a particle has been assigned a displacement, $a\mathbf{s}(\mathbf{q})$, its velocity is simply $\dot{a}\mathbf{s}; = (2/3)\mathbf{s}$ at $a = t = 1$. Assigning velocities in this way picks the growing mode of the perturbations, $\propto a$. It is important that the displacements be small to accurately reproduce the growing mode. For large displacements the velocities assigned to particles will be too large – tending to inhibit collapse on small scales. This problem is considered by Efstathiou et al. (1985). It is usually feasible to start a simulation at an epoch at which particle displacements and velocities are accurately reproduced. The additional computational expense of integrating from earlier epochs is generally small, especially since clustering is light at early epochs. These initial conditions may be used for starting other Friedmann models since, at early times, the expansion is close to the Einstein–de Sitter solution. The error corresponds to including a small component of the decaying mode.

11.3 FORTRAN77 Implementation

The evolution program `prun` and initial conditions program `pstart` are described in this section. Due to the length of the code only the main subroutines are described in detail.

The programs provided are optimized for scalar machines and contain constructions which are unsuitable for specialized architectures such as vector processors. Where applicable, modifications are described which are better optimised for other architectures. The FFT used is `FOURN` from Press et al. (1986). The performance of large (or multi-dimensional) FFTs may vary widely depending upon the architecture of the computer being used due to the wide range of data strides in the FFT algorithm. It may be advantageous to recode the multidimensional FFT as a series of 1-D or 2-D FFTs. For computers with a small data cache the improvement in execution time may be substantial. Large improvements in speed are often obtained by using an FFT from a local optimised subroutine library.

These programs make use of two FORTRAN77 constructions which are currently discouraged: the `common` and `goto` statements. The `common` blocks are generally used as a 'structures' and can be replaced by passing more arguments in subroutine calls. Several arrays which are not used concurrently are mapped to a work array, `work`, using `equivalence` statements to reduce memory requirements. The availability of 'structures' and dynamic memory allocation in Fortran90 will remove the need for the `common` statements in the present code. The `goto` statement is used to construct loops or as an efficient break from a `do` loop.

The code reproduced in the text is schematic only and should be studied in conjunction with a complete listing. Implicit FORTRAN naming rules are assumed. Initializations to zero have been omitted where the meaning is clear. In cases where an operation is repeated over the three dimensions or there are repetitive operations, only one is shown. All code relating to the calculation of potential energies has been omitted. Potential energies are carried by variables starting with `U` and may be ignored when encountered in the text. Inline comments, preceded by an exclamation mark (!), have been added to the code reproduced below.

11.3.1 Program `prun`

The simulation volume is a cubic box with L(=`L`) potential mesh cells per side. The parameter L must be an integer power of 2. Particle positions satisfy $1 \leq \mathbf{r}_i < L + 1$. This avoids referencing arrays `(0:L-1)` or the scaling that would be necessary for particles arranged in the unit cube. Changing the scaling entails only minor program modifications.

Two 'include' files referenced by the programs are described first, followed by the main module of `prun`. The remainder of the code is described following the order established in the previous section. First, the leapfrog timestep routine is outlined followed by the routines necessary for the PM, P^3M and AP^3M calculations. The subroutine `grav` and its sub-programs solves for the gravity of the particle distribution. It is self-contained and could be used in a different context than that presented here or with different integration schemes. The input and output routines and various utilities are described in Sect. 11.3.10.

Include `psize.f`. Sets maximum dimensions for all arrays. See routine `arraycheck` for limits on `mpmax`.

```
      parameter(Nmax=32768)      ! Max. no. of particles
      parameter(Lmax=32)         ! Max. no. of grid cells per side
      parameter(Npmax=32768,Lpmax=32)    ! Max. dimensions for refinements
      parameter(mpmax=1000)              ! Max. no. of refinements
```

Include `pinfo.f`. Passes program parameters between modules. The contents of the array `ibuf` are written as an unformatted record to every data output file.

```
      common/info/nobj,ips,istart,irun,ipout,ipstop,ipdump,
     x           iseed1,iseed2,L,iru,nlmx,
     x           alpha,perr,aexp,p,dpnp1,dpn,Ust,Tst,Uh,Usum,Usm,T,Tsum,
     x           delI,delC,dfCU,rLbox,delta,soft,sftinit,sftmin
      dimension ibuf(100)
      equivalence (ibuf,nobj)    ! Variables in info written into start of ibuf
```

Program `prun`. Main module.

Calls: `arraycheck`, `Nget`, `input`, `infout` and `step`

```
      program prun
      include 'psize.f'
      include 'pinfo.f'

      character startfile*60,greendir*60,datadir*60
      common /datadir/datadir        ! Directory for datafiles
      common /greendir/greendir      ! Directory for stored Green's functions
      real r(3,Nmax),v(3,Nmax)       ! Particle positions and velocities: x, y, z
      call arraycheck     ! Check for array conflicts set in psize.f

      open(55,file='prun.dat',status='old')      ! Input program parameters
      read(55,'(a)')greendir
      read(55,'(a)')datadir
      read(55,'(a)') startfile                   ! Input data file

      nobj=Nget(startfile)          ! No. of particles in data file startfile
      if(nobj.gt.Nmax)stop 'N>Nmax'
      call input(nobj,r,v,startfile)    ! Read ibuf, r, v

      read(55,*) irun             ! Parameters overwrite variables in ibuf
      read(55,*) ipstop,ipout,ipdump
      read(55,*) dpnp1,alphat
      read(55,*) sftinit,sftmin
      read(55,*) perr,nlmx
      close(55)

      call infout(irun,'r')       ! Initializes log files
      call step(nobj,r,v)         ! Evolve
      stop
```

11.3.2 Time-Stepping

Leapfrog integration is used. Routine `step` loops through time-steps. At each step `updaterv` updates particle positions and velocities.

Subroutine `step`. Does the time-stepping. Program parameters are written to the logfile every step via `infout`. Routine `output` handles the binary output of particle positions and velocities.

Calls: `updaterv`, `output` and `infout`

```
      subroutine step(N,r,v)
      include 'pinfo.f'
      real r(3,N),v(3,N)

1     call updaterv(N,r,v,dpnp1)  ! Update r_{n-1/2} -> r_{n+1/2} and v_n -> v_{n+1}
      ips=ips+1                   ! Step counter (n -> n+1)
      dpn=dpnp1
      call output(N,r,v,irun,0)    ! Writes rvdump.dat1 iff mod(ip,ipdump)=0
      if(mod(ips,ipout).eq.0)then    ! Output r and v at same epoch
```

```
   call updaterv(N,r,v,0.)     ! Update r_{n+1/2} -> r_{n+1}
   dpn=0.                      ! Gives correct restart from following output
   call output(N,r,v,irun,1)   ! Writes rv[irun].[ips]
end if
call infout(irun,'r')          ! Update log files
if(ips.lt.ipstop)goto 1
return
```

Subroutine `updaterv`. Routine performs the leapfrog updating of positions and velocities using the new stepsize `dpnext`. If `dpnext` is zero, only particle positions will be updated by a half step.

Calls: `grav`

```
      subroutine updaterv(N,r,v,dpnext)
      include 'pinfo.f'
      real r(3,N),v(3,N)

      rL=float(L)
      rL1=10.*rL-1.
      dmp=0.5*(dpn+dpnext)       ! Step r_{n-1/2} -> r_{n+1/2}
      do 1 i=1,N
       r(1,i)=mod(r(1,i)+dmp*v(1,i)+rL1,rL)+1.     ! mod for periodic b.c.'s
1     continue
      if(dpnext.gt.0.)then       ! If v is to be stepped
       p=p+0.5*dpnext            ! Step p to n + 1/2
       dpD=(0.5+alpha)*dpnext/2./alpha/p      ! H(p)Δp; Eq. (11.3)
       dpv=(1.-dpD)/(1.+dpD)
       dpf=abs(9.*dpnext/(2.*alpha*p)**2/(1.+dpD))

       do 2 i=1,N
        v(1,i)=dpv*v(1,i)        ! 'Hubble drag' in comoving coordinates
2      continue

       To=T
       ah=(2.*alpha*p/3.)**(1./alpha)      ! Expansion, a, at half step
       soft=max(sftmin,sftinit/ah)         ! Softening at half step
       call grav(dpf,soft,Uh,perr,nlmx,N,r,v,L)    ! Forces (v_n -> v_{n+1})

       do 3 i=1,N
3      T=T+v(1,i)**2+v(2,i)**2+v(3,i)**2
       T=0.5*T                        ! Kinetic energy - program units

       p=p+0.5*dpnext                 ! Step p to n + 1
       aexpo=aexp
       aexp=(2.*alpha*p/3.)**(1./alpha)
       T=T*aexp**(2.*alpha-1.)             ! Kinetic energy
       daexp=aexp-aexpo                    ! Δa
       Tsum=Tsum+0.5*(T+To)*daexp          ! ∫ T da
       Usum=Usum+Uh*daexp                  ! ∫ U da

       delI=aexp*Tsum-Usum
```

```
 delC=delI-(aexp-1.)*(Tst-Ust)      ! cf. Eq. (11.4)
 dfCU=delC/Usum          ! Estimated fractional accuracy for Layzer–Irvine
end if
return
```

11.3.3 Particle–Mesh Calculation

Routine `mesh` is the standard PM calculation for periodic boundary conditions. Routine `green` returns the appropriate Green's function for `mesh`.

Subroutine `mesh`. Calculates the periodic mesh force for a system of N particles. The calculation is split into the four main steps of the standard mesh calculation: assignment of particles to the grid; convolution; force differencing; and interpolation of forces back to the particles. Force differencing and interpolation are performed together for each dimension. Note that if any of these modules is changed the Green's function `greenc` must be modified according to Eq. (11.6). `U` is the mesh potential energy.

Calls: `assign`, `cnvl`, `gridf10` and `interp`

```
      subroutine mesh(U,N,r,v,L,gp3,diff,d3,frc)
      real d3(L,L,L),frc(L,L,L)
      real gp3(L/2+1,L/2+1,L/2+1),diff(L/2+1,L/2+1,L/2+1)
      real r(3,N),v(3,N)

      call assign(N,r,L,d3)       ! Smooth particles onto grid d3
      call cnvl(d3,L,gp3,diff,Ut)      ! Convolve with Green's function gp3
      do 1 ic=1,3                 ! Loop over three coordinate directions
         call gridf10(ic,d3,frc,L)     ! Difference potential to get forces frc
         call interp(ic,N,r,v,L,frc)     ! Interpolate forces back to particles
1     continue
      return
```

Subroutine `assign`. Assigns particles to a cubic mesh `d3` using Triangular Shaped Cloud smoothing. The accompanying code may be slightly optimized by eliminating repeated multiplications in the particle interpolation.

```
      subroutine assign(N,r,L,d3)
      real d3(L,L,L),r(3,N)

      do 2 i=1,N          ! Loop over particles
       rrx=r(1,i)         ! Only x coordinate shown

       ix= int(rrx+0.5)
       hx = rrx -ix
       ix=mod(ix-1,L)+1           ! Three grid points
       ixp1=mod(ix,L)+1           !    over which TSC
       ixm1=mod(ix-2+L,L)+1       !    extends

       hx0=0.75 - hx*hx                ! Fraction of particle assigned
```

```
         hxp1=0.5* (0.5 + hx)**2        !   to each grid point:
         hxm1=0.5* (0.5 - hx)**2        !   Eq. (11.7)

         d3(ixm1,iy  ,iz  )   = d3(ixm1,iy  ,iz  )+ hxm1*hy0  *hz0
         d3(ix  ,iy  ,iz  )   = d3(ix  ,iy  ,iz  )+ hx0 *hy0  *hz0
         d3(ixp1,iy  ,iz  )   = d3(ixp1,iy  ,iz  )+ hxp1*hy0  *hz0
           .
           .     ! Interpolation to remaining 24 grid points
           .
2       continue
        return
```

Subroutine `gridf10`. Difference the grid potential, `d3`, returned by `cnvl` in the direction `ic` using a 10 point differencing operator.

```
        subroutine gridf10(ic,d3,frc,L)
        real d3(L,L,L),frc(L,L,L)      ! Potential and force arrays
        L1=L-1
        if(ic.eq.1)then        ! Only F_x shown
         do 51 iz=1,L
          do 51 iy=1,L
           do 51 ix=1,L
            ixp1=mod(ix+1+L1,L)+1      ! i+1: Eq. (11.8)
              .
            frc(ix,iy,iz)=2.*(d3(ixp1,iy  ,iz  )-d3(ixm1,iy  ,iz  ))
     x                   +d3(ixp1,iyp1,iz  )-d3(ixm1,iyp1,iz  )
     x                   +d3(ixp1,iym1,iz  )-d3(ixm1,iym1,iz  )
     x                   +d3(ixp1,iy  ,izp1)-d3(ixm1,iy  ,izp1)
     x                   +d3(ixp1,iy  ,izm1)-d3(ixm1,iy  ,izm1)
51       continue
        else
          .         ! Statements for y and z directions
        end if
        return
```

Subroutine `interp`. Interpolates forces in one dimension back onto the particles using TSC interpolation.

```
        subroutine interp(ic,N,r,v,L,frc)
        real r(3,N),v(3,N)
        real frc(L,L,L)      ! Grid forces

        do 1 i=1,N          ! Loop over particles
         rrx=r(1,i)         ! Only x direction shown

         ix= int(rrx+0.5)      ! Same code as in assign
         hx = rrx -ix          !    for TSC
           .
           .
           .
         ac =    frc(ixm1,iym1,izm1)* hxm1*hym1 *hzm1
     x         +frc(ix  ,iym1,izm1)* hx0 *hym1 *hzm1
     x         +frc(ixp1,iym1,izm1)* hxp1*hym1 *hzm1
```

```
     x         +      ! remaining 24 terms...

        v(ic,i)=v(ic,i)+ac       ! Increment velocities

 1      continue
        return
```

Subroutine `green`**.** Reads the appropriate periodic Green's function from disc in the directory `greendir` or, if the disc file does not exist, generates the required function through a call to `greenc` and stores it to disc. Values are saved between calls and updated only as necessary. The Green's function for `L=`l mesh points and force softening `soft=`s is stored in the file `grnL`l`s`s. The parameter `gamp` is used to scale `gp3`. Array `diff` is the effective Green's function for the mesh potential energy calculation.

Calls: `greenc`

```
        subroutine green(soft,gamp,L,gp3,diff)
        dimension gp3(L/2+1,L/2+1,L/2+1),diff(L/2+1,L/2+1,L/2+1)
        character*60 greendir
        common /greendir/greendir
```

Subroutine `greenc`**.** Returns the optimal Green's function for the periodic mesh calculation in array `gp3`. Array `diff` contains the optimum potential energy function. Both functions have six fold permutation symmetry on the three array indices. The coding of the routine need not be especially efficient since the functions are stored on disc.

```
        subroutine greenc(soft,ifl,L,gp3,diff)
        parameter(tpi=6.28318530717959,ipn=2)
        dimension gp3(L/2+1,L/2+1,L/2+1),diff(L/2+1,L/2+1,L/2+1)

        shp2(x)=12.*(2.-2.*cos(x)-x*sin(x))/x**4     ! S~_2

        df=2.*tpi/float(L**3)       ! Normalisation for FFT
        gf=df

        tpiL=tpi/float(L)           ! Fundamental frequency, k_0 = 2π/L
        L2=L/2
        Lc=L/2+1
        Le=L/4+1

        af=2./15.
        do 400 igz=1,Lc             ! Loop up to Nyquist frequency
         rkz=tpiL*float(igz-1)      ! Only k_z shown
         s1z=sin(rkz)
         c1z=cos(rkz)
         s2z=sin(rkz/2.)
         s2z2=s2z**2
         psz=2.*s2z
         do 400 igy=1,igz           ! Cubic symmetry
```

```
      do 400 igx=1,igy

        rk2=rkx*rkx+rky*rky+rkz*rkz     ! k^2

        do 500 nz=-ipn,ipn      ! Alias sums
         rkzn=rkz-tpi*nz        ! k_{n_z}
         qz=1.
         if(rkzn.ne.0.)qz=psz/rkzn
         do 500 ny=-ipn,ipn

          do 500 nx=-ipn,ipn

           qt=(qx*qy*qz)**3                         ! Ũ(k_n) for TSC
           rk2n=rkxn*rkxn+rkyn*rkyn+rkzn*rkzn       ! k_n^2

           Sprf=shp2(soft*sqrt(rk2n)/2.)
           SprfU=Sprf*qt
           st=SprfU*SprfU/rk2n     ! ** cf. igreen

           SUM=SUM+st            ! Σ S̃_2^2(k_n)Ũ(k_n)/k_n^2
           SRz=SRz+rkzn*st       ! Σ k_x S̃_2^2(k_n)Ũ(k_n)/k_n^2
500     continue

        SAa=(1.+s2x2*(af*s2x2-1.))      ! Σ Ũ^2(k_n)
     x     *(1.+s2y2*(af*s2y2-1.))
     x     *(1.+s2z2*(af*s2z2-1.))
        Sd=SAa**2

        Dkz=2.*s1z*(1.+c1x+c1y)      ! D̃_z(k) for 10-point difference operator
        DWR=(SRx*Dkx+SRy*Dky+SRz*Dkz)/(Dkx*Dkx+Dky*Dky+Dkz*Dkz)
        gp3t=gf*DWR/Sd       ! Optimal Green's function
        difft=df*SUM/Sd       ! Optimal potential energy function
       gp3(igx,igy,igz)=gp3t       ! Symmetrize gp3 (diff)
         .

400   continue
      return
```

11.3.4 Isolated Mesh Calculation

The AP^{3}M algorithm requires the mesh force for an isolated set of particles. This is provided by `imesh`. James (1977) has described a more efficient technique than the FFT for calculating the Coulombic gravity for the isolated problem. The Fourier method is essential, however, when force shaping is required.

The calculation steps are exactly as for `mesh`. Routines `iassign`, `igrid10f` and `iinterp` differ from their periodic counterparts only in that the unnecessary `mod` functions have been removed for efficiency. The most significant difference is in the convolution routine `icnvlt`.

Subroutine `imesh` (entry `irim`). Calculates the isolated mesh force for a system of `N` particles. Particles are assigned to a cubic grid with `L+1` cells per side. No particle can be closer to an edge than `rim`. This boundary is necessary so that no interpolation cloud overlaps the edge of the grid and forces are correctly calculated at all required grid points. For the TSC and 10-point differencing `rim=1.5` grid cells. Entry `irim` allows a simple call to return the boundary size for a particular mesh scheme. Array `dt` is a work array – see `force` below.

Calls: `iassign`, `icnvlt`, `igridf10` and `iinterp`

```
      subroutine imesh(U,Uslf,N,r,v,L,gi3,difi,dt,d3,frc)
      parameter(rim=1.5+.0003)     ! grid boundary for TSC + roundoff safety
      real dt(L+1,L+1,2*L+2),frc(L+1,L+1,L+1),d3(L+1,L+1,L+1)
      real gi3(L+1,L+1,L+1),difi(L+1,L+1,L+1)
      real r(3,N),v(3,N)

      call iassign(N,r,L,d3)
      call icnvlt(dt,L,gi3,difi,Ut)     ! Convolution with zero padding
      U=U+Ut
      U=U-Uslf*N     ! Subtract the self-energy of N particles

      do 2 ic=1,3
         call igridf10(ic,d3,frc,L)
         call iinterp(ic,N,r,v,L,frc)
2     continue
      return

      entry irim(rimd)
      rimd=rim
      return
```

Subroutine `igreen`. Reads to, or writes from, disc the appropriate Green's function for the isolated problem. The Green's function for `L=`l mesh points, short range force softening `sfts=`s and long range softening `sftl=`s_l is stored in the file `grnL`l`s`s`sl`s_l in the `greendir` directory.

Calls: `igreenc`

```
      subroutine igreen(sfts,sftl,gamp,L,gi3,difi,gl,Uslf)
      dimension gi3(L+1,L+1,L+1),difi(L+1,L+1,L+1)
      dimension gl(2*L+1,2*L+1,2*L+1)
      character*60 greendir
      common /greendir/greendir
```

Subroutine `igreenc`. Returns the appropriate Green's functions for the isolated mesh calculation. The resulting force as a function of r has the form $R(r/s) - R(r/s_l)$ where s =`sfts` and s_l =`sftl`. The principal difference between the periodic calculation and the isolated case is that the Fourier components for the isolated calculation are not known analytically and must be calculated. Accuracy requires knowledge of the high frequency components which will be aliased by the discrete grid. The first part of the code calculates these Fourier components. The remainder, after the call to **realevenfft**, modifies the Fourier components as in subroutine **greenc** to allow for the effects of interpolation and differencing.

Calls: S2, **realevenfft**

```
      subroutine igreenc(sfts,sftl,L,gi3,difi,gl,Uslf)
      parameter(tpi=6.28318530717959)
      dimension gi3(L+1,L+1,L+1),difi(L+1,L+1,L+1)
      dimension gl(2*L+1,2*L+1,2*L+1)     ! Work array for aliases

      df=1./(float((2*L)**3)*8.)
      do 400 iz=1,2*L+1        ! Fill gl with required real-space force
       do 400 iy=1,iz
        do 400 ix=1,iy
         rad=0.5*sqrt(float((ix-1)**2+(iy-1)**2+(iz-1)**2))
         call S2(rad,sfts,grav,pot1)
         call S2(rad,sftl,grav,pot2)
         gtmp=pot1-pot2      ! Potential energy equivalent of S2(r/s) - S2(r/sl)
         if(rad.eq.0.)Uslf=0.5*gtmp      ! Self energy for pairwise interaction
         gtmp=df*gtmp
         gl(ix,iy,iz)=gtmp      ! Symmetrize

400   continue
      call realevenfft(gl,L4)      ! Calculate Fourier components to first alias
         do 501 inz=0,1      ! Alias sum (cf. loop 500 in greenc)

            st=gl(icx,icy,icz)*(qx*qy*qz)**6      ! cf. ** in greenc

501      continue
      return
```

11.3.5 Particle–Particle Calculation

Routine **list** places particles into the chaining cells and creates a linked list of all particles in each cell. Routine **shgravll** performs the sums over neighbours and updates the velocities of the particles. The linked list structure presented here is ideal for scalar computers but is less suitable for either vector or parallel architectures. If 'gather–scatter' hardware is available on a vector computer the linked list structure may still be used efficiently to load temporary buffers with particles. Otherwise it will be more efficient to create a sorted list of particles in chaining cells to use in loading the temporary buffers.

Subroutine list. The routine creates a linked list of particles in each chaining cell. The particles are linked in order of decreasing z (=r(3,i)) coordinate. A straight insertion method is used to order the particles in each cell. This method may easily be replaced by a more efficient sorting algorithm if chaining cells contain very large numbers of particles. The arrays ihc and nhc are one-dimensional arrays representing a cube with Ls chaining cells on a side. The array `ihc` for each chaining cell contains the index of the first particle in the cell. Subsequent particle indices are stored in the linked list array `ll`. Array `nhc` returns the number of particles in each chaining cell.

```
      subroutine list(N,r,ll,Ls,ihc,nhc,rca,rcb)
      real r(3,N)
      integer ihc(Ls**3),nhc(Ls**3),ll(N)
```

Subroutine `shgravll`. Computes the pairwise sum between all particles in neighbouring chaining-mesh cells. When used as part of the AP^{3}M calculation the sum excludes particle pairs from cells which both lie in the same refinement. A chaining cell, `ibox`, which lies in a refinement stores the refinement label, `ibrf`, in the high order bits of array `ihc`: `ibrf=ihc(ibox)/(N+1)` (see routines `refine`, Sect. 11.3.7 and `arraycheck`, Sect. 11.3.10). If `ibrf=0` the cell is not in a refinement. Only the essential features of this routine are presented so that the structure of the algorithm is clear.

```
      subroutine shgravll(U,N,r,v,ll,L,Ls,ihc,re,ntp,G,D,dptl)
      real G(0:ntp),D(0:ntp)        ! Force and PE tables from shforce
      real r(3,N),v(3,N)
      integer ihc(Ls**3),ll(N)      ! Head of chain and linked lists
      re2=re**2      ! re = size of chaining mesh cells (= r_c)
      delin=1./(re2/float(ntp))     ! Spacing in force table, G
      do 700 ibz=1,Ls         ! Loop over central chaining cells, I
       do 700 iby=1,Ls
        do 700 ibx=1,Ls
         ibox=ibx+Ls*(iby-1)+Ls*Ls*(ibz-1)      ! Central cell index
         ibrf=ihc(ibox)/(N+1)          ! Central cell refinement label
         is=ihc(ibox)-ibrf*(N+1)       ! First particle in cell I
         do 702 nbz=-1,0          ! Loop over neighbour cells, J
          do 702 nby=-1,1         !    inbz=ibz+nbz etc.,
           do 702 nbx=-1,1        !    omitted code allows for periodicity
            nnb=nnb+1             ! No. of neighbour boxes searched so far
            if(nnb.eq.15)goto 712       ! Do each pair of cells once
             nbox=inbx+Ls*(inby-1)+Ls*Ls*(inbz-1)      ! Neighbour cell index
             nbrf=ihc(nbox)/(N+1)       ! Neighbour cell refinement label

             if(ibrf.ne.nbrf .or. ibrf.eq.0)then     ! Not same refinement
               i=is       ! First particle in cell I
701            if(i.gt.0)then       ! Loop over particles in cell I

                  j=ihc(nbox)-nbrf*(N+1)       ! First particle in cell J
703               if(j.eq.0 .or. j.eq.i)goto 710      ! Do particles in cell J

                    rz=r(3,i)-r(3,j)       ! z separation of particles i and j
```

```
                    if(rz.gt.re)goto 710    ! Particle list in decreasing z

                    rad2=rx**2+ry**2+rz**2     ! Radial separation
                    grc=rad2*delin    ! Set up to interpolate R(r)/r
                    igrc=int(grc)     !       from table G
                    if(igrc .lt. ntp)then      ! r < r_c(=re)
                       gfrac=grc-igrc
                       Gr=G(igrc)+(G(igrc+1)-G(igrc))*gfrac
                       v(1,i)=v(1,i)-rx*Gr     ! Equal and opposite forces
                       v(1,j)=v(1,j)+rx*Gr
                    end if
                  j=ll(j)      ! Next particle in neighbour cell, J
                  goto 703
710               continue      ! Finished particles in cell, J
               i=ll(i)      ! Next particle in central cell, I
               goto 701
               end if       ! Finished particles in central cell, I
             end if
702      continue
712      continue      ! Finished neighbour cells
700   continue         ! Finished central cells
      return
```

Subroutine `shforce`. Loads the pairwise force and energy interpolation tables for the short range force calculation. Values are saved between calls and updated only if necessary. The force tabulated is $[R(r/s) - R(r/s_l)]/r$, where $s =$ `sfts` and $s_l =$ `sftl`. Values are stored at equal increments in r^2 of `re**2/float(ntp)`, where `re` is the chaining cell size.

Calls: `S2`

```
      subroutine shforce(sfts,sftl,gamp,re,spc,ntp,G,D,dptl)
      real G(0:ntp),D(0:ntp)      ! Force and potential energy tables
```

11.3.6 Adaptive P^{3}M

The routines described in the previous two sections are the heart of the AP^{3}M algorithm. Subroutine `force` below is the P^{3}M algorithm. Mesh forces are calculated by either `mesh` or `imesh`, depending upon the boundary conditions required. The standard P^{3}M algorithm is completed by calling `shgravll` to calculate the short range direct sum. In AP^{3}M, subroutine `refine` is called before `shgravll` to exclude high density – and thus expensive – regions from the direct sum. Excluded regions are calculated later with further application of an appropriate P^{3}M cycle. The successive P^{3}M cycles are controlled by subroutine `grav`.

Subroutine `grav`. The routine calculates the interparticle forces for `N` particles in a periodic system by a succession of increasingly refined P^{3}M cycles.

Each invocation of `force` calculates P^3M forces for a subset of the particles omitting the PP sum for those in regions which will be further refined. The first call of `force` is treated separately as it operates on all particles. All subsequent calls solve for the self gravity of isolated patches of particles which have been selected for refinement by routine `refine` in previous calls to `force`. The auxiliary routines `rfinit`, `clist`, `load` and `uload` are described below. A list of refinements is stored in common `rfprm`.

Calls: `force`, `rfinit`, `clist`, `load` and `uload`.

```
      subroutine grav(dpf,soft,Uh,err,nlmx,N,r,v,L)
      include 'psize.f'
      real r(3,N),v(3,N),rp(3,Npmax),vp(3,Npmax)
      integer ll(Nmax),ihc(Lsmax3),nhc(Lsmax3),llp(Npmax)

      common /rfprm/ NLins(4,0:mpmax),psf(5,0:mpmax),mp,ip
      omg=float(L**3)/(3.*tpi*N)    ! Cosmological scaling: L^3/6πN

      ip=0              ! Base mesh has index 0
      mp=0              ! No sub-meshes yet created
      ibc=1             ! Periodic boundary conditions for base mesh
      call rfinit(N,L,Ls,rf,srf,rsa,rsb,ibc,err,softm)
      call force(N,r,v,ll,L,Ls,ihc,nhc,rf,srf,rsa,rsb,softm,
     x                      ibc,Uh,err,nlmx,soft,dpf,omg)
      call clist(N,r,ll,Ls,ihc,rsa,rsb)
      do 1 ip=1,mpmax              ! Loop over refinements ip
       if(ip.gt.mp)goto 2          ! Finished refinements
       Np=NLins(1,ip)              ! No. of particles in refinement ip
       call load(N,r,ll,Np,rp,vp,Lp,Lsp,rf,srf,rsa,rsb,err,softm)
       call force(Np,rp,vp,llp,Lp,Lsp,ihc,nhc,rf,srf,rsa,rsb,softm,
     x                              0,Uh,err,nlmx,soft,dpf,omg)
       call uload(N,v,ll,Np,rp,vp,Lsp,ihc,rsa,rsb)
 1    continue
 2    return
```

Subroutine `force`. Accumulates the force for each refinement by performing a P^3M cycle. Periodic or isolated boundary conditions for the mesh force are set by the input parameter `ibc` (`=1` or `0` respectively). Subroutine `list` creates the linked lists, `ll`, head of chain tables, `ihc`, and cell occupancy numbers, `nhc`. The latter array is passed to `refine` which selects regions to exclude from the short range sum.

Calls: `green` & `mesh` or `igreen` & `imesh` for periodic or isolated boundary conditions; `shgravll`, `shforce`, `list` and `refine`.

```
      subroutine force(N,r,v,ll,L,Ls,ihc,nhc,rf,srf,rsa,rsb,softm,
     x                            ibc,U,err,nlmx,soft,dpf,omg)
      include 'psize.f'
      real r(3,N),v(3,N)
      integer ll(N),ihc(Ls**3),nhc(Ls**3)
      real gp3(Lmaxc3),diff(Lmaxc3)
      real d3(Lmax3),frc(Lmax3)
      real gi3(Lpmax13),difi(Lpmax13),gl((2*Lpmax+1)**3)
```

```
real di3(Lpmax13),frci(Lpmax13),dt(2*Lpmax13)
real G(0:ntp),D(0:ntp)

re=1./rsa       ! Short-range cutoff, r_c
softR=rf*soft       ! rf is scaling relative to base mesh
gamp=dpf*rf*rf*omg

sft=max(softR,softm)        ! softm is mesh softening
if(ibc.eq.1)then            ! Periodic boundary conditions
 call green(sft,gamp,L,gp3,diff)
 call mesh(Ut,N,r,v,L,gp3,diff,d3,frc)
else if(ibc.eq.0)then       ! Isolated boundary conditions
 sftl=rf*srf      ! Long range softening = rf* parent's mesh softening
 call igreen(sft,sftl,gamp,L,gi3,difi,gl,Uslf)
 call imesh(Ut,Uslf,N,r,v,L,gi3,difi,dt,di3,frci)
end if
if(softR.lt.sft)then       ! If PP needed to augment PM force
 call list(N,r,ll,Ls,ihc,nhc,rsa,rsb)
 call refine(N,Ls,ihc,nhc,ibc,err,nlmx)      ! Labels ihc with refs.
 call shforce(softR,sft,gamp,re,spc,ntp,G,D,dptl)
 call shgravll(Ut,N,r,v,ll,L,Ls,ihc,re,ntp,G,D,dptl)
end if
return
```

11.3.7 Auxiliary AP^{3}M Routines

The routines `rfinit`, `clist`, `load` and `uload` are used to scale parameters used in `force` and to load and unload temporary buffers and perform book-keeping operations. Data for each refinement is maintained in the common block `rfprm`. Routine `refine` adds new refinements – created by `plref` – to common `rfprm`.

Common `rfprm`. Data for new refinements (up to a maximum number of `mpmax` – see `arraycheck`, Sect. 11.3.10) are accumulated in the arrays in `rfprm` as they are generated by subroutine `refine`.

```
common /rfprm/ NLins(4,0:mpmax),psf(5,0:mpmax),mp,ip
```

Data for each refinement, index `ip`, out of a current total of `mp` are stored as follows:

```
NLins(1,ip)     ! Number of particles
NLins(2,ip)     ! Number of potential mesh cells on a side
NLins(3,ip)     ! Index of refinement's parent
NLins(4,ip)     ! First particle in linked list
psf(1,ip)       ! Coordinate of refinement corner
psf(2,ip)       !    with minimum x, y and z
psf(3,ip)       !    values in base mesh units
psf(4,ip)       ! Size of refinement in base mesh units
```

```
psf(5,ip)         ! Softening of parent's mesh force in base mesh units
```

Subroutine `rfinit`. Initializes data for the base mesh, `ip=0`, in common `rfprm`. Returns force splitting parameters for the P^3M calculation for a given required error through a call to `errfit`.

Calls: `irim`, `errfit`

```
subroutine rfinit(N,L,Ls,rf,srf,rsa,rsb,ibc,err,softm)
include 'psize.f'
common /rfprm/ NLins(4,0:mpmax),psf(5,0:mpmax),mp,ip
```

Subroutine `clist`. Routine called after the base mesh calculation. A linked list of particles in each refinement is constructed and stored in `ll`. The integer `NLins(4,ip)` is the entry point to the list for refinement `ip`.

```
subroutine clist(N,r,ll,Ls,ihc,rsa,rsb)
include 'psize.f'
real r(3,N)
integer ll(N),ihc(Ls**3)
common /rfprm/ NLins(4,0:mpmax),psf(5,0:mpmax),mp,ip
```

Subroutine `load`. Performs the duties of `rfinit` for refinements and, in addition, loads particle positions `r` into a temporary buffer, `rp`, which is passed to `force`.

Calls: `irim`, `errfit`

```
subroutine load(N,r,ll,Np,rp,vp,Lp,Ls,rf,srf,rsa,rsb,err,softm)
include 'psize.f'
real r(3,N),rp(3,Np),vp(3,Np)
integer ll(N)
common /rfprm/ NLins(4,0:mpmax),psf(5,0:mpmax),mp,ip
```

Subroutine `uload`. Called after calculation of refinement forces. The routine creates linked lists for new refinements in the same way as `clist`. In addition the routine updates the velocities, `v`, from the temporary buffer `vp`.

```
subroutine uload(N,v,ll,Np,rp,vp,Ls,ihc,rsa,rsb)
include 'psize.f'
real v(3,N),rp(3,Np),vp(3,Np)
integer ll(N),ihc(Ls**3)
common /rfprm/ NLins(4,0:mpmax),psf(5,0:mpmax),mp,ip
```

Subroutine `refine`. Calls routine `plref` to place new refinements, adds the new refinements to the list in common `rfprm` and labels high order bits of the elements in array `ihc` with the refinement label if appropriate.

Calls: `plref`

```
subroutine refine(N,Ls,ihc,nhc,ibc,err,nlmx)
```

```
include 'psize.f'
integer Lpt(mpmax),isp(mpmax),is(mpmax)
common /rfprm/ NLins(4,0:mpmax),psf(5,0:mpmax),mp,ip
integer ihc(Ls**3),nhc(Ls**3)
```

Subroutine `plref`. Determines where new refined patches should be placed within the current refinement. A simple routine which places refinements over high value cells in array `nhc` is included. `plref` returns `mpt` new refinements, label `ipt`, each with `Lpt(ipt)` potential mesh cells per side covering `is(ipt)` chaining cells starting at cell `isp(ipt)` in the linear array `nhc`.

```
subroutine plref(Ls,nhc,mpt,Lpt,isp,is,N,ibc,err)
include 'psize.f'
integer nhc(Ls**3)
integer Lpt(mpmax),isp(mpmax),is(mpmax)
```

11.3.8 Program `pstart`

Much of the code for `pstart` uses routines which have already been described. Subroutine `initr` sets up an initial distribution of particles which is then perturbed by `zdisp`. The heart of the program is the subroutine `potin` which populates discrete Fourier space and generates the Zel'dovich potential. The potential is differenced to create the displacement field. Less emphasis has been placed on the coding efficiency of `pstart` than of `prun`.

Program `pstart`. Main module.

Calls: `initr`, `zdisp`, `infout` and `output`

```
program pstart
include 'psize.f'
include 'pinfo.f'

character greendir*60,datadir*60

common /greendir/greendir
common /datadir/datadir

real r(3,Nmax),v(3,Nmax)

open(55,file='pstart.dat',status='old')    ! Read program parameters
read(55,'(a)')greendir
read(55,'(a)')datadir
read(55,*)istart
read(55,*)nobj,L
read(55,*)soft,perr,alpha
read(55,*)iru
read(55,*)iseed1,iseed2
read(55,*)rLbox
read(55,*)delta
close(55)

if(nobj.gt.Nmax)stop 'N>Nmax'

call infout(istart,'s')     ! Open logfile ps[istart].log
```

```
      call initr(iru,spc,ial,iseed1,L,nobj,r)      ! Initial distribution
      call zdisp(soft,perr,ial,spc,delta,iseed2,
     x rLbox,Usm,Uf,Us,nobj,r,v,L)      ! Impose fluctuations on distribution

      do 3 i=1,nobj
3     T=T+v(1,i)**2+v(2,i)**2+v(3,i)**2
      T=0.5*T                                    ! Initial kinetic energy

      call output(nobj,r,v,istart,1)  ! Write binary data file rv[istart].0000
      call infout(istart,'s')          ! Write to logfile
      stop
```

Subroutine `initr`. Assigns initial particle positions. The particles can each be randomly placed within the computational cube (`iru = 1`) or placed uniformly on a cubic lattice (`iru = 0`). In the latter case the number of particles, `N`, must be a perfect cube.

```
      subroutine initr(iru,spc,ial,iseed,L,N,r)
      real r(3,N)
```

11.3.9 Zel'dovich Displacement

Given an initial particle distribution set up by `initr`, this routine imposes a fluctuation spectrum on the distribution.

Subroutine `zdisp`. The routine calls `potin` to generate a 'Zel'dovich potential.' This potential is then differenced and used to displace particles according to Eq. (11.11). Of the following routine calls, only `greenf`, `potin` and `displ` are concerned with setting up the particle distribution, all others are required for the calculation of potential energy. The energy calculation is performed exactly as in `prun` and is omitted from the following discussion.

Calls: `errfit`, `shforce`, `list`, `shgravll`, `greenf`, `potin`, `displ`, `green`, `assign` and `cnvl`

```
      subroutine zdisp(soft,perr,ial,spc,delta,iseed2,
     x   rLbox,Usm,Uf,Us,N,r,v,L)
      include 'psize.f'
      real r(3,N),v(3,N)
      real pf(Lmaxc3)      ! Components of Z: Eq. (11.11)
      complex dc(Lmax,Lmax,Lmax)
      real d3(Lmax,Lmax,Lmax),frc(Lmax,Lmax,Lmax)
      equivalence(dc,d3),(dc(1,1,Lmaxc),frc)

      call greenf(ial,L,pf)      ! Calculate Green's function, Z̃
      call potin(sft,Ui,Usmth,delta,iseed2,
     x     rLbox,L,dc,d3,pf,df)      ! Calculate 'Zel'dovich potential'
      call displ(N,r,v,L,d3,frc)      ! Displace particles
      return
```

Subroutine `potin`. Calculates the displacement potential for a power spectrum returned by function `pow`. Fourier space is populated with an Hermitian field which, when transformed, results in a real displacement potential. There is some inefficiency in the routine since a complex transform is performed to produce the real field.

Calls: `pow`

```
      subroutine potin(softg,Ui,Usmth,delta,iseed2,
     x     rLbox,L,dc,d3,pf)
      parameter(tpi=6.28318530717959)
      complex dc(L,L,L)
      real d3(L,L,L),pf(L/2+1,L/2+1,L/2+1)

      rL=float(L)
      tpiL=tpi/rL        ! k_0, fundamental frequency - program units
      rLb=rL/rLbox       ! Scaling from program to physical wavenumber
      iseedt=iseed2

      do 300 iz=1,L/2+1          ! k_z
       icz=mod(L+1-iz,L)+1       ! k_{-z}
       do 300 iy=1,L/2+1
        do 300 ix=1,L/2+1
         rk=tpiL*sqrt(float((ix-1)**2+(iy-1)**2+(iz-1)**2))     ! k
         rkrl=rk*rLb      ! Physical wavenumber

         amp=delta*sqrt(pow(rkrl))        ! |\tilde{\delta}(k)|
         amp=amp*pf(ix,iy,iz)             ! pf = Z(k), cf. Eq. (11.11)
         arg1=tpi*ran3(iseedt)            ! \phi_k
         dc(ix ,iy ,iz )=amp*cmplx(cos(arg1),sin(arg1))
         dc(icx,icy,icz)=conjg(dc(ix ,iy ,iz ))
            .
            .        ! Populate remaining 3 independent octants of k space
            .
300   continue
      call fourn(dc,nn,ndim,-1)      ! Real displacement potential
      do 210 iz=1,L
       do 210 iy=1,L
        do 210 ix=1,L
         d3(ix,iy,iz)=real(dc(ix,iy,iz))      ! Compress into real array d3
 210  continue
      return
```

Function `pow`. Returns the amplitude of the power spectrum for given input *physical* wavenumber. A specific example of `pow` is discussed in Sect. 11.4.

```
      real function pow(rk)
```

Subroutine `disp`. Differences the ‘Zel’dovich potential’ to produce a displacement field at grid points which is then interpolated to the particles. This routine is similar to the second part of `mesh` after the call to `cnvl`.

Calls: `gridf10`, `interp`

```
      subroutine displ(N,r,v,L,d3,frc)
      real d3(L,L,L),frc(L,L,L)
      real r(3,N),v(3,N)

      rL=float(L)
      rL1=10.*rL-1.
      hst=2./3.      ! (ȧ/a) for Einstein–de Sitter at a = 1
      do 2 ic=1,3
         call gridf10(ic,d3,frc,L)
         call interp(ic,N,r,v,L,frc)      ! Store displacements, s, in v
 2    continue
      do 3 i=1,N      ! Loop over particles
         r(1,i)=mod(r(1,i)+v(1,i)+rL1,rL)+1.     ! Move particles
         v(1,i)=hst*v(1,i)      ! Assign velocities, v = (2/3)s
 3    continue
      return
```

Subroutine `greenf`. Controls the disc input and output of `greenfc`. The Green's function for `L` $= l$ and `ial` $= i$ is stored in file `grnfL`l`i`i in directory `greendir`.

Calls: `greenfc`

```
      subroutine greenf(ial,L,pf)
      dimension pf(L/2+1,L/2+1,L/2+1)
      character*60 greendir
      common /greendir/greendir
```

Subroutine `greenfc`. Calculates the Green's function for the Zel'dovich displacement potential. The routine is very similar to `greenc` and only the essential differences are shown.

```
      subroutine greenfc(ial,L,gp3)
      parameter(tpi=6.28318530717959,ipn=2,isgn=1-2*mod(ipn,2))
      dimension gp3(L/2+1,L/2+1,L/2+1)

      shp2(x)=12.*(2.-2.*cos(x)-x*sin(x))/x**4

      tpiL=tpi/float(L)
      do 400 igz=1,L/2+1
       rkz=tpiL*float(igz-1)

       pz=1.
       if(igz.ne.1)pz=psz/rkz       ! sin(k/2)/(k/2)

          Smp=0.
          do 500 nz=-ipn,ipn

             qt=(qx*qy*qz)**3       ! Ũ(k_n) for TSC

             Smp=qt-Smp
 500      continue
```

```
            Smp=isgn*Smp      ! ∑_n Ũ(k_n)
            if(ial.eq.0)Smp=(px*py*pz)**3
            gp3t=0.5/((rkx*Dkx+rky*Dky+rkz*Dkz)*Smp)    ! (k·D̃ ∑_n Ũ)^-1
           gp3(igx,igy,igz)=gp3t      ! Symmetrize

 400    continue
        return
```

11.3.10 Library, I/O and Utility Routines

Subroutine `infout`**.** Opens the logfiles and positions file pointers at the end of each file. Each step the run logfile (`p[cd][irst].log`) is updated with program parameters and then closed. The refinement logfile (`prf[irst].log` – opened by `prun`) remains open on unit 77 when the routine returns. It is written to by routines `updaterv`, `grav`, `force` and `refine`. It is closed in routine `updaterv` after `grav` returns.

```
        subroutine infout(irst,cd)
        include 'pinfo.f'
        character cd*1
```

Subroutine `output`**.** Outputs the buffer `info` from include file `pinfo.f` together with the current particle positions and velocities. The output is unformatted. If `ioflag=1`, data is written to a new file labelled with the current step: `rv[irst].[ips]`. If `ioflag=0` data overwrites the backup file `rvdump.dat1`. Files are written in the directory `datadir`.

```
        subroutine output(N,r,v,irst,ioflag)
        include 'pinfo.f'
        common /datadir/datadir
        real r(3,N),v(3,N)
```

Subroutine `input`**.** Reads unformatted data as written by `output` from the file `startfile`.

```
        subroutine input(N,r,v,startfile)
        real r(3,N),v(3,N)
        character startfile*(*)
        include 'pinfo.f'
```

Function `Nget`**.** Returns the number of particles stored in the binary file `startfile`.

```
        integer function Nget(startfile)
        character startfile*(*)
```

Subroutine `errfit`**.** The routine returns the mesh softening `softm` and chaining mesh size `Ls` for given requested mesh force error `err` and potential mesh size `rLa(=float(L))`. The routine assumes TSC interpolation and the 10 point force differencing schemes described above.

```
subroutine errfit(err,ibc,rLa,softm,Ls)
```

Subroutine `S2`. Returns the pairwise force, `grav`, and potential energy, `dpot`, for two S_2 shaped particles separation `r` and softening `a`.

```
subroutine S2(r,a,grav,dpot)
```

Subroutine `cnvl`. Performs an in-place real discrete convolution of the input density, `d3`, and Green's function, `g3`, using FFTs. The routine returns the mesh potential energy.

Calls: `fourn`

```
subroutine cnvl(d3,L,g3,diff,U)
dimension d3(L,L,L),g3(L/2+1,L/2+1,L/2+1),diff(L/2+1,L/2+1,L/2+1)
```

Subroutine `icnvlt`. Performs a 3D FFT convolution for the isolated problem using zero padding in each dimension.

Calls: `fourn`

```
subroutine icnvlt(dt,L,gi3,difi,U)
dimension dt(L+1,L+1,2*L+2),gi3(L+1,L+1,L+1),difi(L+1,L+1,L+1)
```

Subroutine `realevenfft`. Calculates the 3D FFT of a real even function with cubic symmetry.

Calls: `fourn`

```
subroutine realevenfft(g,L)
real g(L/2+1,L/2+1,L/2+1)
```

Subroutine `arraycheck`. The subroutine will abort the program if on entry incorrect array sizes have been set in the include file `psize.f`. Parameter `mpmax` in `rfprm` and `psize.f` must not overflow the bits remaining in the integers of `ihc` above `N`(= the number of particles).

```
subroutine arraycheck
parameter(nbit=32)
include 'psize.f'
```

Function `lblnk`. Returns the length of a character string to the first blank character. Uses the Fortran intrinsic `INDEX`.

```
integer function lblnk(char)
character char*(*)
```

Function `cputime`. Function to return cumulative cpu time for program. This routine is a dummy as provided. It should be tailored to call an appropriate timing routine.

Calls: *System-dependent timing routine*

```
real function cputime()
```

Subroutine `FOURN`, **function** `RAN3`. These routines are from 'Numerical Recipes' (Press et al. 1986). Routine `FOURN` is a standard three-dimensional FFT. Function `RAN3` returns a random number.

```
SUBROUTINE FOURN(DATA,NN,NDIM,ISIGN)
DIMENSION NN(NDIM),DATA(*)

FUNCTION RAN3(IDUM)
```

11.4 Using the Programs – A Test Case

The machine readable code and data files are set up to run a simple test case. The test is performed with the following input parameters. The two files must have the names indicated below.

Input file `pstart.dat`. Read by program `pstart`.

`greenfn/`	! Directory for stored Green's functions, `greendir`
`data/`	! Directory for data file storage, `datadir`
`9000`	! Start label, `istart`
`32768 32`	! Number of particles, `nobj`; mesh points per side, `L`
`0.6 6.0 1.5`	! Softening, `soft`; max. force error (%), `perr`; α, Eq. (11.2), `alpha`
`0`	! Initial particle distribution; `iru=`, `0` – uniform, `1` – random
`-1001 -10001`	! Random number seeds: particles, `iseed1`; $\phi_{\mathbf{k}}$, `iseed2`
`80`	! Physical scaling, `rLbox` (for input to `pow`)
`0.0004`	! Amplitude applied to input Fourier modes, `delta`

The maximum pairwise force error is set here to 6%. The RMS error is ~2%. For a distribution of particles the RMS error is $\sim 0.2\%$ (Couchman 1991). The parameter `rLbox`$[= D]$ allows scaling of the input power spectrum. Thus, if wavenumbers in `pow`$[= P(k)]$ are measured in units of Mpc^{-1}, `rLbox` would be the size of the computational cube in Mpc. The parameter `delta`$[= \delta]$ adjusts the normalization of the spectrum. The input amplitudes are $\delta\sqrt{P(k)}$, where $k = (2\pi/D)\,|\mathbf{n}|$; $\mathbf{n}$ is an integer triple representing the wavenumber in the discretely sampled Fourier space in units of the fundamental. For the test case the input power spectrum is given by `pow=110.*exp(-(rk/0.96)**16)/rk`. For `rLbox=80` the exponential cut-off is at ≈ 0.76 of the Nyquist frequency for $L = 32$.

The corresponding continuous spectrum is

$$P(k) = \left(\frac{D}{2\pi}\right)^3 \delta^2 110\, k^{-1} \exp\left[-(k/0.96)^{16}\right]. \tag{11.12}$$

The RMS overdensity fluctuations, $\delta\rho/\rho$, on small scales for this input are ≈ 0.5. This is too large for good accuracy using `pstart` but will serve for the test case.

Input file `prun.dat`. Read by program `prun`.

```
greenfn/
data/
data/rv9000.0000      ! Initial data file startfile
9001                  ! Run label, irun
75 25 4            ! Final step, ipstop; output, ipout, and dump, ipdump, intervals
0.84 1.5              ! Δp, dpnp1; α – Eq. (11.2), alpha
0.6  0.03             ! Softening: initial, sftinit; minimum, sftmin
6.0  4                ! Max. force error (%), perr; max refinement level, nlmx
```

The scaling used here is $p = a^{3/2}$; thus $p \equiv t$. At the end of the simulation $t = 1 + 75\Delta p = 64$, corresponding to expansion by a factor of $a = 16$. For a power law spectrum $\propto k^{-1}$ the RMS overdensity fluctuations, σ, scale as $\sigma \sim k$. After expansion by a factor of 16, the longest modes in the box will therefore be either non-linear or close to it. The simulation will not correctly represent the evolution of the spectrum in Eq. (11.12) at late times. Furthermore the timesteps, Δp, are much too large for accurate integration as evidenced by the poor energy conservation.

11.4.1 Environment

In the following discussion the current directory (here called `cwd`) should contain the source code, include files – named `psize.f` and `pinfo.f` – and input data files – named `pstart.dat` and `prun.dat`. It is convenient to set up two subdirectories of `cwd` for the data files and Green's functions. In the example input files above, a Unix style directory hierarchy has been assumed and the two subdirectories have been named `data/` and `greenfn/` respectively. These entries should be edited if necessary to suit the local directory structure. Filenames are constructed by concatenating the given strings with the appropriate character endings. Thus, the full name of the directory must be given including any directory delimiters (here `/`). The directories must exist before the programs are run. Leaving these fields blank will place files in the current directory. The storage required for all data and Green's functions is <5 Mb. The executables require about 1 Mb of memory for execution. (Memory requirements assume 32 bit words.)

11.4.2 Compilation and Execution

The two programs are provided in a single file to ease initial compilation. A runtime switch selects either `pstart` or `prun`. If possible edit the source code to include a FORTRAN callable timing routine in function `cputime`.

This function is a dummy as provided but contains a number of suggested routines for various operating systems. The following steps will run the test:

1. Compile the program with the include files `psize.f` and `pinfo.f`.
2. Check that the input files are named `pstart.dat` and `prun.dat`, and that the subdirectories named in them (if any) exist. The rest of this test assumes that the remainder of these files are unmodified.
3. Run `pstart`. It will create a log file `ps9000.log` in `cwd` and a data file `rv9000.000` in `data/`.
4. Run `prun`. It will create two log files, `pr9001.log` and `prf9001.log` in `cwd` and the following data files in subdirectory `data/`: `rv9001.0025`, `rv9001.0050`, `rv9001.0075` and `rvdump.dat1`.

11.4.3 Timing

The timings given here are only approximate. On a ∼10 Mflop machine `pstart` will run in ∼ 40 s. The first few steps of `prun` each take ∼ 20 s, the final step ∼ 65 s. The whole run takes about 50 minutes. The time saving of the AP^{3}M algorithm can be estimated by setting the parameter `nlmx` to `0` in `prun.dat`. This switches off all refinements and forces the algorithm to run as standard P^{3}M. The saving achieved by using AP^{3}M in this test case is roughly a factor of 40% for the final steps. Setting `sftinit` = `sftmin`≥ 3.5 in `prun.dat` runs the code as PM. Each step will take roughly 5 s in this case.

For larger numbers of particles the time per step will scale roughly as the particle number provided clustering is light and $N \simeq L^3$. Under heavy clustering, for large N, the time per step for the P^{3}M algorithm can easily increase by a factor of at least 100 compared with the time for a uniform distribution. For the AP^{3}M algorithm the increase under heavy clustering should be no more than a factor of 3–5. This number is somewhat dependent upon the sophistication of routine `plref`. The scaling properties of the P^{3}M and AP^{3}M algorithms are discussed by Couchman (1991).

The efficiency of P^{3}M and AP^{3}M is influenced by the choice of the the maximum pairwise force error `perr`. Smaller errors require large mesh-force softenings and hence large chaining cells, thereby extending the short range search over a larger number of particles. The error in these programs ranges from 3% to 10%. Demanding a value outside this range sets the error to the closest end of the range. The cut-off radius r_c is the ratio of the number of potential-mesh cells to chaining-mesh cells; `L/Ls=re`. The ratios are 32/12, 32/13 and 32/16 for 10%, 6% and 3% errors respectively.

11.4.4 Storage and Memory Requirements

Each output file contains $100 + 6N$ words in three unformatted records. The total memory requirement of the executable is close to $10N$ words. For large N (and hence L) simulations $\sim$ 100 Green's functions can be generated and stored. The periodic functions occupy roughly $2(L/2)^3$ words. Isolated Green's functions occupy up to $2L_{\max}^3$ words, where $L_{\max}$ is the maximum refinement mesh size. These sizes can be reduced by a factor of 6 if advantage is taken of the full cubic symmetry.

11.4.5 Output

The binary output files in subdirectory `data` may be read with the following code fragment (from routine `input`):

```
include 'pinfo.f'
real r(3,nobj),v(3,nobj)
read(11) ibuf
read(11) r
read(11) v
```

The logfiles `ps9000.log`, `pr9001.log` and `prf9001.log` in `cwd` are written in ASCII. File `prf9001.log` records the refinement cycles of the AP^{3}M algorithm. The output for timestep 75 from `pr9001.log` is given below.

```
Step    75
CPU time: step=     63.690   total=      3195.270
 step no     75    a=     15.99998    soft=   0.03767
 p=    63.99986 dpn=     0.00000 dpnp1=     0.84000
 Uinit= 0.21596E+04 Uhs= 0.56727E+06 Usum= 0.32546E+07
 Tinit= 0.14743E+04    T= 0.25498E+05 Tsum= 0.22653E+06
 aTsum-Usum= 0.36986E+06 dC= 0.38014E+06 dC/Usum= 0.11680E+00
```

Both `pr9001.log` and `prf9001.log` are updated each step and may be used to monitor `prun` during execution.

11.4.6 Practical Considerations

Several general considerations relevant to cosmological simulations were discussed in the introduction. The following additional remarks are provided as a starting point for those who are unfamiliar with this kind of simulation and would like to experiment with the codes.

Constructing initial conditions for a cosmological simulation must be done with care. Most input spectra must be filtered before being applied to the particle distribution in order that a realistic spectrum results in the initial conditions. This is especially important if the grid attenuation corrections, derived in Sect. 11.2.5, are applied. A rough starting point for using `pstart` is that the RMS overdensity fluctuations should never exceed ~ 0.1 on the smallest scales in the initial conditions. The simplest way to monitor the initial conditions and their subsequent evolution is to measure the power spectrum of the particle distribution. A basic spectrum measurement program may be constructed by calling `assign` followed by `fourn`.

The characteristics which determine the evolution of the particle distribution are the stepsize and the softening. The program `prun` has no internal determination of the appropriate stepsize. The maximum stepsize is a function of the softening employed in the pairwise force and of the initial conditions being evolved. Efstathiou et al. (1985) give a useful discussion. If it is desired to retain high overdensities in the code the softening must be chosen small enough to retain bound groups yet not so small that two-body scattering dominates the structure on small scales. The correct softening must be estimated by determining the epoch at which the scales of interest are collapsing and setting the softening to be of order the interparticle spacing in physical units at that time.

11.5 Further Information

If you require further information, or wish to receive information concerning future changes or improvements to the code, please send e-mail to `couchman@cita.utoronto.ca` with the subject heading `AP3M updates`.

Acknowledgments

I would like to thank Ray Carlberg for introducing me to particle–mesh methods and cosmological simulations. The book by Hockney and Eastwood, *Computer Simulation Using Particles* (1981), is an indispensable reference for the techniques used in all the particle–mesh methods described above. The article by Efstathiou et al. (1985) serves as a foundation for much of the work done using the P^3M algorithm in cosmology and provided several ideas for the present code. I am grateful to *Numerical Recipes Software* for their permission to include the routines `FOURN` and `RAN3`.

References

Carlberg, R. G., Couchman, H. M. P. (1989) Mergers and bias in a Cold Dark Matter cosmology. Astrophysical Journal **340** 47–68

Couchman, H. M. P. (1991) Mesh-refined P^3M: a fast adaptive N-body algorithm. Astrophysical Journal Letters **368** 23–26

Eastwood, J. W., Brownrigg, D. R. K. (1979) Remarks on the solution of Poisson's equation for isolated systems. Journal of Computational Physics **32** 24–38.

Efstathiou, G., Eastwood, J. W. (1981) On the clustering of particles in an expanding universe. Monthly Notices of the Royal Astronomical Society **194** 503–526

Efstathiou, G., Davis, M., Frenk, C., White, S. D. M. (1985) Numerical techniques for large cosmological N-body simulations. Astrophysical Journal Supplement Series **57** 241–260

Hernquist, L., Bouchet, F., Suto, Y. (1991) Application of the Ewald method to cosmological N-body simulations. Astrophysical Journal Supplement Series **75** 231–240

Hockney, R., Eastwood, J. W. (1981), Computer Simulation Using Particles. McGraw–Hill, New York

James, R. A. (1977) The Solution of the Poisson equation for isolated source distributions. Journal of Computational Physics **25** 71–93

Peebles, P. J. E. (1980), The Large-Scale Structure of the Universe. Princeton University Press, Princeton

Peebles, P. J. E. (1987) A fluid model for the evolution of large-scale structure. Astrophysical Journal **317** 576–587

Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T., (1986) Numerical recipes: the art of scientific computing. Cambridge University Press, Cambridge

Zel'dovich, Ya. B. (1970) Gravitational instability: an approximate theory for large density perturbations. Astronomy and Astrophysics **5** 84–89

This article was processed by the author using the TEX macro package from Springer-Verlag.