Fundamentals of Simulation Methods

Exercise Sheet 4 — FFT-based methods and the particle mesh approach

Daniel Rosenblüh, Janosh Riebesell · Lecturer: Volker Springel · November 20th, 2015

1 FFT-based convolution

As a warm-up for applying FFT methods in the particle mesh force calculation, let's try to convolve a two-dimensional image with a kernel of the form:

$$W(r) = k\begin{cases} 1-6\left(\frac{r}{h}\right)^2+6\left(\frac{r}{h}\right)^3 & \text{for } 0 \le \frac{r}{h} < \frac{1}{2}, \\ 2\left(1-\frac{r}{h}\right)^3 & \text{for } \frac{1}{2} \le \frac{r}{h} < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

(a) Calculate the normalization factor k in $W(r)$ such that the kernel is normalized to unity, i.e. that we have $\int W(|\boldsymbol{r}|)\mathrm{d}^2\boldsymbol{r} = 1$.

(b) Now use either the supplied C-code template `smooth_image.c` or the Python template `smooth_image.py` to smooth the color image `aq-original.ppm` from the Moodle lecture page. For the smoothing length, we shall adopt $h = 10$ pixel. The image has 512 pixel on a side and uses 3 color channels for red, green and blue (we chose the ppm-file format for its simplicity in reading and writing). Modify the code template such that the kernel is properly set up in real space. Also, add code that multiplies the Fourier transformed kernel and the Fourier transforms of the different color channels in Fourier space. The code will write the smoothed image to a file `aq-smoothed.ppm`.

(c) To check your results, determine the sum of all red, green and blue pixel values (carried out in double precision) both before the smoothing and after the smoothing is complete. Report these values as part of your solution (if there is any difference – how could this be avoided?). Provide the smoothed image.

Technical hints: The provided C-template uses the popular FFTW library for the Fourier transforms. If this is not available on your computer, you can download the library and compile it with the following steps:
First do a `tar -zxvf fftw-3.3.4.tar.gz` to unpack the downloaded source package. Change into the `fftw-3.3.4` directory. Configure the code with a command of the form `./configure --prefix=/my/install/dir` where the path given after the prefix-option is your desired installation path. Then do a `make` followed by `make install` to compile and install the library. The files you need are then found in `/my/install/dir/include` and `/my/install/dir/lib`.
If you're getting the error `fftw3.h: No such file or directory` when compiling your image code, the default search path of the compiler does not know about the installation directory of the `fftw3` library. In this case, you can add an explicit search path to the compile command, for example: `cc -I/my/install/dir/include smooth_image.c -o smooth_image`.
Similarly, if you get `undefined symbol: fftw plan dft 2d`, then the `fftw3` library can not be found or is not specified on the compile/link command. You can fix this with something like: `cc -I/my/install/dir/include -L/my/install/dir/lib -lfftw3 smooth_image.c -o smooth_image`.
If you are using Python instead, you can use the `numpy.fft` package instead for the Fourier transforms, which is more convenient to use but not quite as fast as FFTW. Note that it also uses a slightly different normalization convention as FFTW.

(a) Since $W(r)$ depends only on the magnitude of $\boldsymbol{r}$, the normalization factor k is easily calculated by transforming into polar coordinates:

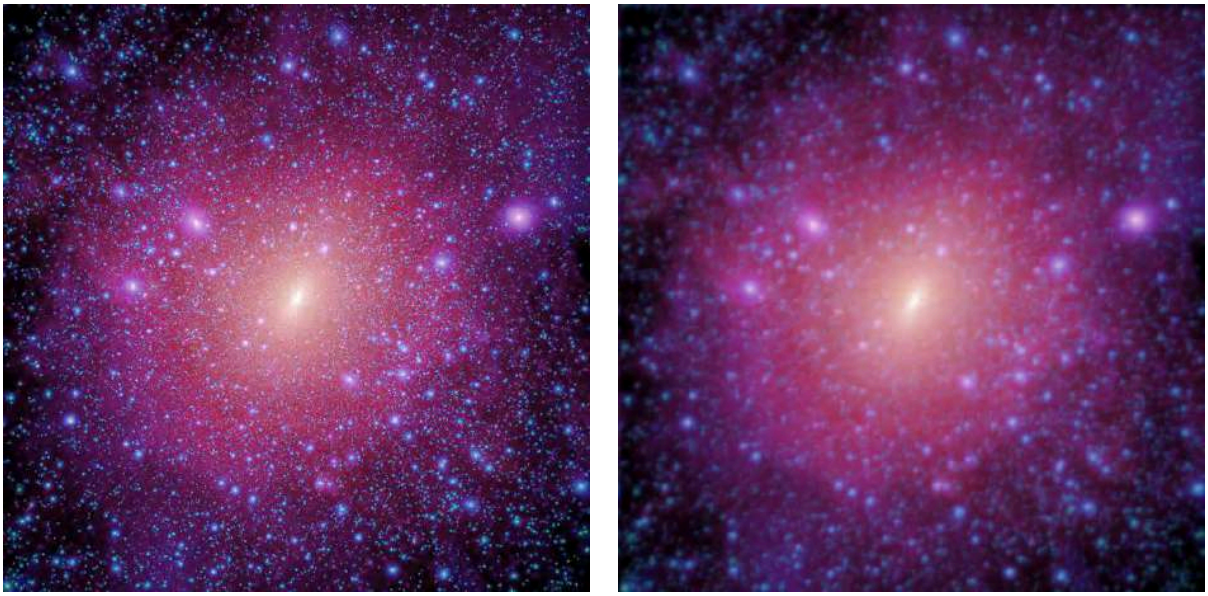
$$\begin{aligned}\int_{\mathbb{R}^2} W(|\boldsymbol{r}|)\mathrm{d}^2\boldsymbol{r} &= \int_0^\infty \int_0^{2\pi} W(|\boldsymbol{r}|) r \,\mathrm{d}\varphi \,\mathrm{d}r \\ &= 2\pi k \int_0^{h/2} \left(1 - \frac{6r^2}{h^2} + \frac{6r^3}{h^3}\right) r \,\mathrm{d}r + 4\pi k \int_{h/2}^{h} \left(1 - \frac{r}{h}\right)^3 r \,\mathrm{d}r \qquad (2) \\ &= \frac{11}{80}\pi h^2 k + \frac{3}{80}\pi h^2 k \overset{!}{=} 1.\end{aligned}$$

Thus we find that a normalized smoothing kernel requires

$$k = \frac{40}{7\pi h^2}. \qquad (3)$$

(b) See `fftconvol.c`.

(c) The original image and the result after performing our smoothing algorithm are shown in Figure 1 (a) and (b), respectively.



(a) Original image (b) Smoothed image

Figure 1: A dark matter distribution before and after processing by our FFT-based convolution

Summing the red, green, and blue values of all pixels before smoothing we obtain

$$\begin{aligned} r_{\mathrm{tot}} &= 25\,182\,443, \\ g_{\mathrm{tot}} &= 12\,287\,845, \\ b_{\mathrm{tot}} &= 24\,795\,222. \end{aligned} \qquad (4)$$

After smoothing, we find

$$\begin{aligned} r_{\mathrm{tot}} &= 25\,182\,442.999\,999\,3, \\ g_{\mathrm{tot}} &= 12\,287\,845.000\,000\,06, \\ b_{\mathrm{tot}} &= 24\,795\,222.000\,000\,7. \end{aligned} \qquad (5)$$

The values are virtually the same, barring a minimal rounding error. We did, however, execute this code on different machines, some of which produced errors up to the order of 10^{-6}. To avoid this, higher precision variables such as `long double` could be used.

2 The particle mesh force law

Based on the code template from above, develop a new code that carries out a gravitational force computation in a two-dimensional periodic space using the particle mesh method with clouds-in-cell (CIC) charge/mass assignment. We would like to measure the effective force law the method delivers, i.e. determine experimentally the force of a particle of unit mass as a function of distance. Because the particle mesh method is linear, the force field for an N-body system will consist of a linear superposition of the effective one-particle force law, so the measurement of the latter is sufficient to characterize the expected force accuracy. Carry out the following steps:

(a) Implement code that constructs a density field for a single particle of unit mass that is randomly placed into a box of unit length, $L = 1$. Take care of periodic wrap-around where needed.

(b) Now implement code that Fourier transforms the density field, multiplies it with the Fourier transform of the Green's function of the Poisson equation, $-4\pi/\boldsymbol{k}^2$ (assuming a periodic space and $G = 1$, also set $\hat{\varphi}_{\boldsymbol{k}} = 0$ for $\boldsymbol{k} = 0$), and transform back to obtain the gravitational potential on the grid. Adopt $N_{\mathrm{grid}} = 256$ for the grid resolution.

(c) Add code that produces force fields in the x- and y-direction from the potential by finite differencing.

(d) Now write code that selects 100 randomly chosen positions in a sphere of radius equal to half the box size around the point you placed in part (a). We would like the distances r to be uniformly distributed in the log between $r_{\mathrm{min}} = 0.3L/N_{\mathrm{grid}}$ and $r_{\mathrm{max}} = L/2$. You can achieve this by calculating displacements

$$\Delta x = r_{\mathrm{min}} \left(\frac{r_{\mathrm{max}}}{r_{\mathrm{min}}} \right)^p \cos(2\pi q), \tag{6}$$

$$\Delta y = r_{\mathrm{min}} \left(\frac{r_{\mathrm{max}}}{r_{\mathrm{min}}} \right)^p \sin(2\pi q), \tag{7}$$

relative to the coordinates of the point placed in part (a), where p and q are uniformly distributed random numbers in the interval $[0, 1)$. Determine the force $\boldsymbol{a} = \left(a_x, a_y\right)$ and its magnitude $a = \sqrt{a_x^2 + a_y^2}$ at the resulting positions by CIC-interpolating from the force fields obtained in part (c). Note that the position may have to be mapped by periodic wrapping into the primary domain in order to allow reading out of the force.

(e) Repeat the whole procedure for 10 different random locations of the point chosen in part (a) such that you obtain 1000 pairs of distances r and corresponding forces a. Plot these points in a scatter plot of a versus r using logarithmic axes for both. Overplot the $a \propto 2/r$ power law expected for Newtonian gravity, and a vertical line at the grid scale L/N_{grid}. Why do we not expect $a \propto 1/r^2$? Interpret the differences at small and large separations, if any.

For parts (a) to (d), see `pmesh.c`.

(e) Figure 2 shows a scatter plot of the forces at the 10×100 randomly generated points throughout our two-dimensional periodic space. The points arrange to form a distinct picture of how the force falls off as a function of the distance from our 10 reference points, each of which bears a massive particle that acts as the origin of gravitational force within the system.

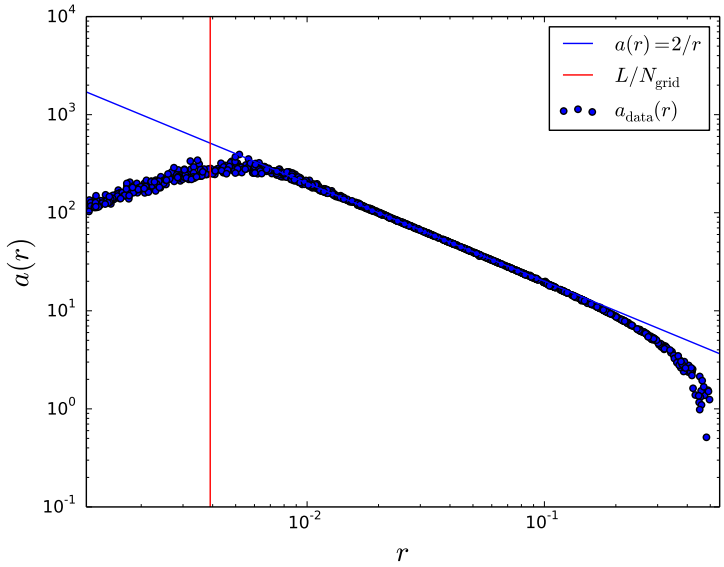


Figure 2: Acceleration $a(r)$ as a function of distance r

We do not expect an inverse square law, i.e. $a \propto 1/r^2$, because our simulation took place in two dimensions, where the surface of a sphere $A_{\text{sph}} = 2\pi r$ grows linearly with the radius.

We see a deviation from the expected $1/r$ behavior of the force both when r approaches the grid scale $L/N_{\text{grid}} = \frac{1}{256}$ and for large distances converging on $r_{\text{max}} = L/2$. We attribute the former to the fact that the PM method forces the particles to have a lower spatial resolution during the force calculation, thereby introducing a significant error for force calculations of nearby particles, where exact relative positions are important. The strange fall-off of the force approaching r_{max} is due to the periodicity of our box. In the extreme case of a point being placed exactly at a distance of r_{max} from the particle, it would feel the "next" particle in the neighbouring box equally strongly and thus not be subject to any net force, $a(r_{\text{max}}) = 0$.