Problem Set 4

Exercises for course Fundamentals of Simulation Methods, WS 2021

Prof. Dr. Mario Flock, Prof. Dr. Friedrich Röpke
Tutors: Brooke Polak (brooke.polak@uni-heidelberg.de | ITA), Glen Hunter (glen.hunter@uni-heidelberg.de | ITA), Jan Henneco (jan.henneco@h-its.org | HITS)
Offices: HITS: Schloss-Wolfsbrunnenweg 35, 69118 Heidelberg; ITA: Albert-Ueberle-Strasse 2, 69120 Heidelberg

Hand in until Wednesday, 17.11.2021, 23:59
Tutorials times: 18-19.11.2021
Group 1: Brooke | Thursday 11:00 - 13:00
Group 2: Glen | Thursday 14:00 - 16:00
Group 3: Jan | Friday 11:00 - 13:00

1. Tree code for gravity calculation [20 pt.]

Tree algorithms can approximately calculate the forces in an N-body system by means of a hierarchical multipole expansion. This reduces the computational cost to something of order $\mathcal{O}(N \ln N)$. On the download page of the lecture, a simple skeleton tree code can be downloaded (a C-version as well as a Python-version are provided). Note that this implementation uses only monopole order and is not optimized for speed or memory consumption. You can use this template for this exercise, either directly or translated to another language, or if you wish, you may also write your own tree code from scratch using the template as an example (it's fun to do! ;-)).

Consider N particles of total mass $M_{\text{tot}} = 1$ placed randomly into a cubical box of unit side-length. For definiteness, we assume all particles have the same mass, and we adopt $G = 1$. Assume that the gravitational potential of individual particles has a gravitational softening length $\epsilon = 0.001$, i.e. we adopt the potential

$$\phi(\vec{r}) = -\frac{m}{(\vec{r}^2 + \epsilon^2)^{1/2}} \tag{1}$$

for a single particle of mass m. This is also known as the Plummer potential.
Calculate the gravitational forces for all N particles with an oct-tree and determine the typical force accuracy and computation time in comparison with direct summation, both as a function of N and of the opening angle parameter $\theta^\star$. To this end, carry out the following steps:

(a) The idea of this exercise is that you use one of the two templates provided (`tree.c` or `tree.py` & `call_tree.py`). But if you like a challenge, you are encouraged to write your own version! As a reference, there is pseudo code provided at the end of the sheet.
Additionally, you can visualize your oct-tree with the provided `plot_tree.py` script.

(b) The provided code template is incomplete, so start by filling in the missing pieces. In particular, you need to add the computation of the centers of mass and total masses of tree nodes from their subnodes, as well as the partial force calculation from a tree node that is used in the tree walk. Also, please add a calculation of the *exact* forces by direct summation. When you are done, verify that the force approximation delivered by the tree code is roughly correct by adding suitable output to the code.

(c) To allow a more quantitative analysis, add an automatic measurement of the average force error after all forces have been calculated by the tree and direct summation. To this end, consider the relative force error

$$\eta = \frac{|\vec{a}_{\rm tree} - \vec{a}_{\rm direct}|}{|\vec{a}_{\rm direct}|} \tag{2}$$

for each particle, and determine a simple arithmetic average $\langle \eta \rangle$ of the mean relative force error[1]. Also, add diagnostic code that tells you the average number of particle-node interactions per particle, i.e. how many nodes the multipole expansion on average involves.

(d) Make a little grid of calculations for $N = 5000, 10000, 20000$ and 40000, and opening angles $\theta^\star = 0.2, 0.4$ and 0.8. In each case, measure the calculation time for the tree-based force calculation and for direction summation, as well as $\langle \eta \rangle$ and the mean number of terms the tree code used per particle. Report the results in a table.

NOTE: For direct summation with $N = 40000$ particles the calculation could take very long; you can extrapolate from smaller N cases since we know it scales as N^2. Especially Python users might need to run smaller resolution problems, e.g. divide the number by ten (however, you should still have at least four different N to be able to extrapolate).

(e) Using the previous results, make a plot of the execution time of the force calculation with the tree as a function of N (for the $\theta_c = 0.4$ case). Use logarithmic axes and fit a regression line (in the log) to the 4 data points you obtained. Also include the results for direct summation. Estimate the time needed for 10^{10} particles for both methods by extrapolating your timing results.

[1]Note that a more careful analysis of the errors should really consider the full distribution of the errors. It could well be that there are occasionally very large, catastrophic errors that go unnoticed in a simple average such as as $\langle \eta \rangle$. To diagnose this, one needs to look at the error distribution function.

Algorithm 1 Build Octree

```
for $i$ in $N$ do
    Insert($i$,$root$)                          ▷ $root$ contains the whole domain
end for

function INSERT($i, node$)                      ▷ Insert particle $i$ to node $node$
    if $node$=empty then
        store $i$ in $node$
    else if 1 particle in $node$ then
        generate children $c$
        store old particle in right child
        Insert(i,c)
    else if #particles in $node > 1$ then
        determine correct child $c$
        Insert(i,c)
    end if
end function
```

Algorithm 2 Calculate Force

```
for $i = 1$ to $N$ do                           ▷ for each particle traverse tree
    $f(i) = \text{Force}(i, root)$              ▷ Compute force on it by calling function Force
end for

function FORCE($i, node$)                       ▷ Compute force on particle $i$ due to $node$
    Force $= 0$
    if $n$ contains one particle then
        Force $=$ $\boldsymbol{F}$ calculated from Plummer-potential
    else
        $R = |\boldsymbol{x}_i - \boldsymbol{x}_{\text{node}}|$      ▷ Distance from particle $i$ to the $node$ under consideration
        $w$ = width of box corresponding to $node$
        if $\frac{w}{R} < \theta$ then          ▷ Criterion for discriminating $far$ from $near$ nodes
            Force $=$ $\boldsymbol{F}$ calculated from Plummer-potential
        else
            for all children $c$ of $node$ do   ▷ Break up node into its children
                Force = Force + Force($i, c$)   ▷ Recursive call of function Force
            end for
        end if
    end if
end function
```