040692 PR Practical in numerical Astronomy

Prof. Dr. Oliver Hahn

Random Sampling & CFD

Dawid Stepanovic

February 21, 2022

Lab Project 1: Generating Plummer Model Initial Conditions for N-body simulation

Develop an N-body simulation program to integrate the trajectories of various bodies under A reasonably good model for a globular cluster is the spherically symmetric Plummer model (Plummer 1911). It is a simple polytrope with index $n = 5$, i.e. $P \propto \rho^{1+1/n} = \rho^{6/5}$. In hydrostatic equilibrium, one finds in this case the density-potential pair (e.g., Binney Tremaine 2011).

$$\rho(r) = \frac{3M}{4\pi b^3}\left(1 + \frac{r^2}{b^2}\right)^{-5/2}, \tag{1}$$

$$\Phi(r) = -\frac{GM}{\sqrt{r^2 + b^2}} \tag{2}$$

where b is the core radius, M the total mass, G the gravitational constant, and r the radial coordinate.

a) Show that dimensionless units can be introduced, in which $G = M = b = 1$. What is the relation between $\hat{r}$, $\hat{\rho}(\hat{r})$ and $\hat{\Phi}(\hat{r})$ in these units and the corresponding quantities in physical units?

Since we assume spherical symmetry in space, the potential $\hat{\Phi}(\hat{r})$ and density $\hat{\rho}(\hat{r})$ only depend on the distance to the center r and are independent of the spherical angles θ and ϕ, such that

$$\hat{\Phi}(\hat{r}) = \Phi(r),$$
$$\hat{\rho}(\hat{r}) = \rho(r).$$

To work out the physical interpretation we write the density $\rho(r)$ in its original representation that basically using the Poisson's equation $\nabla^2\Phi(r) = 4\pi G\rho(r)$, such that

$$\rho(r) = \frac{3M}{4\pi}\frac{b^2}{(r^2 + b^2)^{5/2}}. \tag{3}$$

So, to see how the functions behave we introduce the central or core potential $\Phi_0 = \Phi(0)$ and the central or core density $\rho_0 = \rho(0)$ and represent the density-potential pair in the following way

$$\rho(r) = \frac{3M}{4\pi b^3}\left(1 + \frac{r^2}{b^2}\right)^{-5/2} = \rho_0\left(1 + \frac{r^2}{b^2}\right)^{-5/2}, \tag{4}$$

$$\Phi(r) = -\frac{GM}{b}\left(1 + \frac{r^2}{b^2}\right)^{-1/2} = \Phi_0\left(1 + \frac{r^2}{b^2}\right)^{-1/2}. \tag{5}$$

The central potential corresponds to the surface potential of an object with mass which equals the total mass of the particle divided by the core radius, and the central density is the density that such an object would have. Thus, the physical units for the density equals basically to [mass][volume of a sphere]$^{-1}$ and the the physical units for the potential equals to $[G][\text{mass}][\text{length}]^{-1}$, where G ($= 6.67408 \times 10^{-11} \frac{m^3}{kg\ s^2}$) is the gravitational constant. However, we can easily see that in case in which $G = M = b = 1$ the functions are given in dimensionless units and can be written as

$$\rho(r) = \frac{3}{4\pi}\left(1 + r^2\right)^{-5/2}, \tag{6}$$

$$\Phi(r) = -\left(1 + r^2\right)^{-1/2}, \tag{7}$$

which reveals the relationship between the $\rho(r)$ and $\Phi(r)$ of

$$\rho(r) = -\frac{3}{4\pi}\Phi(r)^5, \tag{8}$$

in dimensionless units. For the sake of completeness it is worth mentioned that we can as well looking at the inverse radial distance squared in units of the core length squared and write

$$\rho(r) = \frac{3Mb^2}{4\pi r^5}\left(1 + \frac{b^2}{r^2}\right)^{-5/2} = \rho_\infty\left(1 + \frac{b^2}{r^2}\right)^{-5/2}, \tag{9}$$

$$\Phi(r) = -\frac{GM}{r}\left(1 + \frac{b^2}{r^2}\right)^{-1/2} = \Phi_\infty\left(1 + \frac{b^2}{r^2}\right)^{-1/2}, \tag{10}$$

where ρ_∞ and Φ_∞ are the factorized constants of an asymptotic behavior of the functions when we go out to very large radii.

In order to make a model of the star cluster, we need to sample from the initial phasespace distribution function $f_0(\boldsymbol{x}, \boldsymbol{v})$ which is six-dimensional. We can break this however down into a simpler problem by exploiting that we can write

$$f_0(\boldsymbol{x}, \boldsymbol{v}) = f_{pos}(\boldsymbol{x}) f_{vel}(\boldsymbol{v}|\boldsymbol{x}) \tag{11}$$

as a product of the distribution of positions, times the conditional distribution of velocities given a position. We can therefore split the problem in two steps, and only consider sampling the positions first.

b) Show that in our dimensionless units, the density $\hat{\rho}(\hat{r})$ is normalised and therefore $p_r(\hat{r}) = 4\pi\hat{r}^2\hat{\rho}(\hat{r})$ is the probability distribution function for $\hat{r}$.

As in a) we are using the fact that $\hat{\Phi}(\hat{r}) = \Phi(r)$ and $\hat{\rho}(\hat{r}) = \rho(r)$. Eq.(3) gives the density function of the plummer model and can be used to calculate the mass of spherical volume of radius r. By calculating the cumulative mass as a function of r ($\mathcal{M}(r)$) we can proof that the density $\rho(r)$ is

normalised. Starting from

$$\begin{aligned}
\mathcal{M}(r) &= \int_0^r p_r(r)dr = \int_0^r 4\pi r^2\rho(r)dr = \int_0^r \frac{3M}{b^3}r^2\left(1+\frac{r^2}{b^2}\right)^{-5/2}dr \overset{(G=M=b=1)}{=} \\
&= \int_0^r 3r^2(1+r^2)^{-5/2}dr \overset{(r^2=y)}{=} \frac{3}{2}\int_0^{r^2} y^{1/2}(1+y)^{-5/2}dy = \\
&= -\int_0^{r^2} y^{1/2}\frac{d}{dy}\left\{(1+y)^{-3/2}\right\}dy \overset{part.Int.}{=} r(1+r^2)^{-3/2} + \frac{1}{2}\int_0^{r^2} y^{-1/2}(1+y)^{-3/2}dy \overset{(x=1/y)}{=} \\
&= r(1+r^2)^{-3/2} - \frac{1}{2}\int_\infty^{r^2}(1+x)^{-3/2}dx = \frac{r^3}{(1+r^2)^{3/2}} = \\
&= \left(1+\frac{1}{r^2}\right)^{-3/2},
\end{aligned}$$

for $M = 1$.

An other way to proof the above mentioned result is by using the Possion's equation, such that

$$\nabla^2\Phi(r) = \frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d}{dr}\Phi(r)\right) = 4\pi G\rho(r). \tag{12}$$

Using the last relationship we can write

$$\begin{aligned}
\mathcal{M}(r) &= \int_0^r p_r(r)dr = \int_0^r 4\pi r^2\rho(r)dr = \\
&= \frac{1}{G}\int_0^r \frac{d}{dr'}\left(r'^2\frac{d}{dr'}\Phi(r')\right)dr' = \\
&= \frac{1}{G}\left(r'^2\frac{d}{dr'}\Phi(r')\right)\Big|_0^r = \frac{1}{G}r^2\frac{d}{dr}\Phi(r) = \\
&= \frac{1}{G}r^2\left(GM\frac{r}{(a^2+r^2)^{3/2}}\right) = M\left(1+\frac{a^2}{r^2}\right)^{-3/2},
\end{aligned}$$

which represents the general form.

By the last derivation we can proof that $M = 1$, since

$$\lim_{r\to\infty}\mathcal{M}(r) = \lim_{r\to\infty} M\left(1+\frac{a^2}{r^2}\right)^{-3/2} = M \overset{\text{by definition}}{=} 1,$$

for fixed $a \in \mathbb{R}$ which proofs that $\rho(r)$ is normalised and therefore $p_r(r) = \frac{d\mathcal{M}(r)}{dr} = 4\pi r^2\rho(r)$.

---

c) Now generate positions $\boldsymbol{x}_i$ for N stars, $i = 1, ..., N$. Assume them to be all of equal mass $\hat{m} = 1/N$ (in N-body units). Due to spherical symmetry, we can draw radius $\hat{r}$ and the two angles θ and ϕ of a star in spherical coordinates independently. Use the inversion method (show that p_r can be inverted in closed form) to sample $\hat{r}$. Use 3D-unit-vector sampling to transform $\hat{r}$ into a 3D vector $\boldsymbol{x}$.

We know from eq. (12) the cumulative mass $\mathcal{M}(r)$ which depends on the radius and have only to invert this relationship, to get the dependence of cumulative mass on radius for the inversion method. For $M = a = 1$ we can write

$$\begin{aligned}
\mathcal{M}(r) = M\left(1+\frac{a^2}{r^2}\right)^{-3/2} \overset{M=a=1}{=} \left(1+\frac{1}{r^2}\right)^{-3/2} &\Longrightarrow \\
\mathcal{M}(r) = \left(1+\frac{1}{r^2}\right)^{-3/2} &\iff \\
\mathcal{M}(r)^{-2/3} = 1+\frac{1}{r^2} &\iff \\
\mathcal{M}(r)^{-2/3} r^2 = 1 + r^2 &\iff \\
r^2(\mathcal{M}(r)^{-2/3} - 1) = 1 &\iff \\
r(m) = \frac{1}{\sqrt{\mathcal{M}(r)^{-2/3} - 1}}.&
\end{aligned}$$

The general form is

$$r(m) = \frac{a}{\sqrt{\left(\frac{m}{M}\right)^{-2/3} - 1}}.$$

Using the last derivations we are able to o sample the radius r. For θ we know that it takes values between 0 and π, such that $\cos(\theta)$ provides values from $+1$ to -1. So, to determine the desired values we have to pick a random number $x \in \{-1, +1\}$ and take the arccosine to sample θ ($y = \arccos(x) \in \{0, \pi\}$). To sample ϕ we simply draw the numbers from $\phi \in \mathcal{U}(0, 2\pi)$. The Cartesian coordinates may be retrieved from the randomly drawn spherical coordinates using

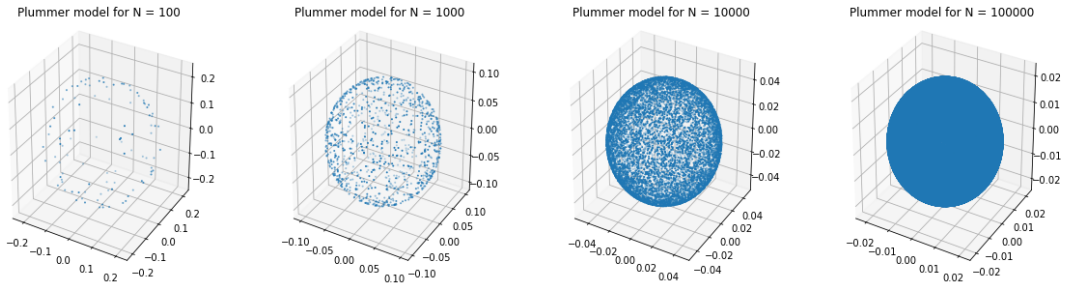
$$\begin{aligned}
x &= r\sin(\phi)\cos(\theta), \\
y &= r\sin(\phi)\sin(\theta), \\
z &= r\cos(\theta).
\end{aligned}$$

However, we have chosen $1/N$ and not zero as lower bound to fulfil the assumption of equal mass for all particles and implement it and implement them in an easy way.

```
import numpy.random
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from math import pi

def plummer_position(number_of_particles,mass_cutoff):

    def calculate_radius():
        random_mass_fraction = np.random.uniform(1/number_of_particles,mass_cutoff,(
            number_of_particles,1))
        radius = 1.0/pow(pow(random_mass_fraction,-2.0/3.0)-1.0,1.0/2.0)
        return radius

    def position_spherical():
        radius = calculate_radius()
        theta  = numpy.arccos(np.random.uniform(-1.0,1.0,(number_of_particles,1)))
        phi    = np.random.uniform(0.0,2*pi,(number_of_particles,1))
        return (radius,theta,phi)

    def spherical_to_cartesian(vec):
        x = vec[0]*np.sin(vec[1])*np.cos(vec[2])
        y = vec[0]*np.sin(vec[1])*np.sin(vec[2])
        z = vec[0]*np.cos(vec[1])
        return (x,y,z)

    return spherical_to_cartesian(position_spherical())
```

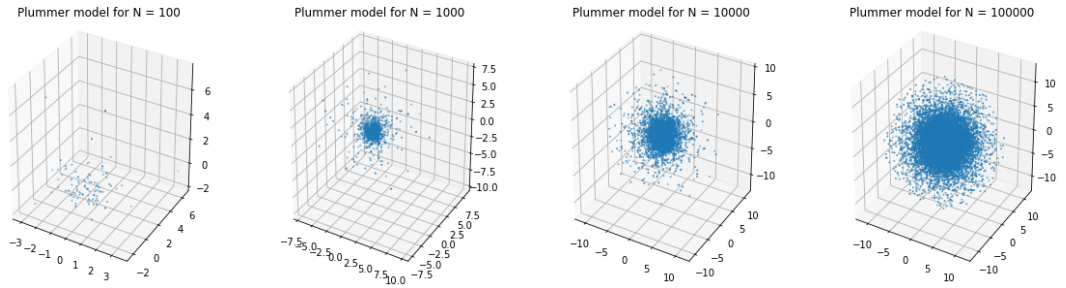
d) Verify that the density profile of the sampled stars approaches the desired profile for large N.

We have analysed the density profile for $N \in \{100, 1000, 10000, 100000\}$ and in the case of $\mathcal{M}(r) = \hat{m} = 1/N$ we obtain the following picture.




```
# Plots - mass cut off = 0.99
N = [100,1000,10000,100000]
fig = plt.figure(figsize=(20,5))
for i  in enumerate(N):
    ax = fig.add_subplot(1,len(N),i[0]+1, projection='3d')
    x,y,z = plummer_position(i[1],0.99)
    ax.scatter3D(x,y,z,lw=0,s=4)
    plt.title("Plummer model for N = %1.0f"%i[1],fontsize=12,x=0.5,y=1.05)
plt.savefig("plummer_1.png")
plt.show()
```

We see that the larger N the better we see the expected isotropic sphere. We have also analysed an uniformly distributed $\mathcal{M}(r)$ between $1/N$ and 1.0 ($\mathcal{M}(r) \in \mathcal{U}(1/N, 1.0)$ which gives us the following density profile evolution.




```
# Plots mass cut off = 1/N
fig = plt.figure(figsize=(20,5))
for i  in enumerate(N):
    ax = fig.add_subplot(1,len(N),i[0]+1, projection='3d')
    x,y,z = plummer_position(i[1],1/i[1])
    ax.scatter3D(x,y,z,lw=0,s=4)
    plt.title("Plummer model for N = %1.0f"%i[1],fontsize=12,x=0.5,y=1.05)
plt.savefig("plummer_2.png")
plt.show()
```

For every star whose position we have already sampled, we now need to sample a velocity from the

conditional distribution $f_{vel}(\boldsymbol{v}|\boldsymbol{x}) = f_{vel}(\boldsymbol{v}|r)$, where the last equality holds in spherical symmetry.

The distribution function of energies in the Plummer model is known (see Binney Tremaine). A star cannot exceed the *escape velocity* v_e corresponding to $E = 0$ at its radius, where $E(v, \hat{r}) = v^2/2 + \hat{\Phi}(\hat{r})$ and therefore $v_e(\hat{r}) = \sqrt{-2\hat{\Phi}(\hat{r})} = \sqrt{2}(1 + \hat{r}^2)^{-1/4}$. Let $q := v/v_e$, then the distribution function of the modulus of the velocity can be written as

$$p_q(q) = \begin{cases} \frac{512}{7\pi} q^2(1-q^2)^{7/2} & 0 < q < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

e) Use von Neumann rejection sampling to sample a $q \sim p_q$ for every star, and use $v = qv_e(\hat{r})$ to turn it into a radius dependent velocity modulus. Finally use the 3D-unit-vector sampling to obtain a 3D isotropic vector.

The normalization factor of $p_q(q)$ does not have to be considered because the scaling factor of the comparing distributions can be dropped. So, the distribution function of the velocity is proportional to $p_q(q) = q^2(1-q^2)^{7/2}$. To limit the sampling to the possible solution, we calculate the upper limit of the distribution such that

$$\frac{dp_q(q)}{dq} = 2(1-q^2)^{7/2} - 7q^3(1-q^2)^{5/2} = 2(1-9q^2)(1-q^2)^{5/2} \stackrel{!}{=} 0 \Longrightarrow q_{max} = \sqrt{\frac{2}{9}}$$

$$p_q(q_{max}) = p_q(\sqrt{\frac{2}{9}}) = 0.092 \sim 0.1,$$

and sample for $q \sim p_q(q)$ out of uniformly distribution up to 0.1. To get the velocity we calculate $v = qv_e(r) = q * \sqrt{2}(1+\hat{r}^2)^{-1/4}$, and sample and ϕ as mentioned in c). We use the 3D-unit-vector sampling as well as mentioned in c) to obtain a 3D isotropic vector by transforming the spherical to Cartesian coordinates.

```
46    def velocity_dens(number_of_particles):
47        Nstore = 0
48        x_value = numpy.zeros(0)
49        y_value = numpy.zeros(0)
50        while (Nstore < number_of_particles):
51            x = np.random.uniform(0,1.0,(number_of_particles-Nstore))
52            y = np.random.uniform(0,0.1,(number_of_particles-Nstore))
53            p = (x**2)*pow(1.0-x**2.0,7.0/2.0)
54            compare = y <= p
55            x_value = np.concatenate((x_value,x.compress(compare)))
56            y_value = np.concatenate((y_value,y.compress(compare)))
57            Nstore  = len(x_value)
58        return (np.atleast_2d(x_value).transpose(),np.atleast_2d(y_value).transpose())
59
60    def velocity_spherical(radius):
61        x,y = velocity_dens(number_of_particles)
62        velocity = x*np.sqrt(2.0)*pow(1.0+radius*radius,-1.0/4.0)
63        theta = np.arccos(np.random.uniform(-1.0,1.0, (number_of_particles,1)))
64        phi = np.random.uniform(0.0,2*pi,(number_of_particles,1))
65        return (velocity,theta,phi)
66
67    def centering(vec,mass):
68        weight = (vec*mass)/sum(mass)
69        center = vec - weight
70        return center
71
72    mass = np.zeros((number_of_particles,1))+(1.0/number_of_particles)
73    radius, theta, phi = position_spherical()
74    position = np.hstack(spherical_to_cartesian(position_spherical()))
75    velocity = np.hstack(spherical_to_cartesian(velocity_spherical(radius)))
76    position = position/scaling
77    velocity = velocity/np.sqrt(1.0/scaling)
```

---

f) Because coordinates and velocities are produced from a random process, center of mass and center of momentum might not be exactly zero for small N. Subtract them, to define the rest frame of the cluster.

To make the center of mass adjustments we have calculated the center of mass position and the center of velocity and subtracted them from each particle's position and form from each particle's velocity, such that

$$\boldsymbol{x}_{center} = \boldsymbol{x}_i - \frac{1}{M}\sum_{i=0}^{N-1} m_i \boldsymbol{x}_i,$$
$$\boldsymbol{v}_{center} = \boldsymbol{v}_i - \frac{1}{M}\sum_{i=0}^{N-1} m_i \boldsymbol{v}_i,$$

where M is the total mass of the system and m_i is the fraction of mass for each star which is in our case $m_i = \frac{1}{M}, \forall i \in N$.

```
78 # centering
79  def centering(vec,mass):
80          weight = (vec*mass)/sum(mass)
81          center = vec - weight
82          return center_1.png")
83      plt.show()
```

---

g) Verify that the distribution function of energies E of the sampled system has the expected scaling $p(E) \propto (E)$ for small energies and large N (plot the histrogram in log-log), and that indeed all your stars are bound, i.e. have $E < 0$ (see e.g., Binney & Tremaine for origin of the $p(E)$ formula, can bonus points for re-deriving it for this system).

Our Plummer model implementation is optionally scaled such that the kinetic and potential energies are 0.25 and -0.5, respectively and thus gives a total energy of -0.25 for the scaled system. For a broader distribution of energies we took a small N-body system of 5 and repeated the sampling 1,000,000 times to obtain the energy distribution. We see that the power-law behavior, especially in the semi-log plot and that energies are clearly bounded by $E < 0$. As mentioned we have plotted the desired log-log distribution and additionally have plotted a semi-log distribution as well. However, for large N-body systems the energy converges to the following numbers.

$$E_{kin} = \frac{3}{2} 4\pi \int_0^\infty \rho(r)\sigma^2(r) r^2 dr = \frac{3\pi}{64}\frac{GM^2}{a}$$

$$E_{pot} = \frac{1}{2} \int_0^\infty \rho(r)\Phi(r) 4\pi r^2 dr = -\frac{3\pi}{32}\frac{GM^2}{a}$$

where $\sigma^2(r)$ is the velocity dispersion, which is given by

$$\sigma^2(r) = \frac{G}{\rho(r)} \int_r^\infty \frac{\rho(r')\mathcal{M}(r')dr'}{r'^2}.$$

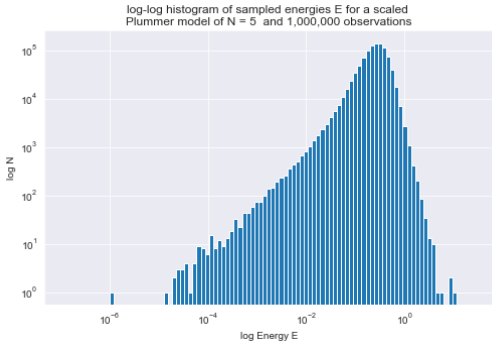
In order to transform the Plummer model into standard units where G=M=a=1 we have to scale the position an velocity vector assuming that $a = \frac{3\pi}{16}$ since the viral radius is $r_{viral} = \frac{16}{3\pi}a = 1$ which is the inverse of the mean distance of particles $i, j \in N$, for $\forall i \neq j$. Doing so we get right scaling by

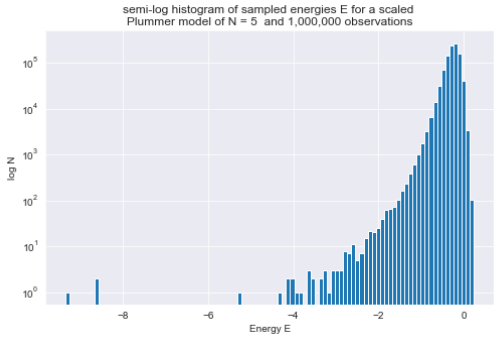
$$\boldsymbol{x_{new}} = \frac{\boldsymbol{x}}{\frac{3\pi}{16}} \text{ and } \boldsymbol{v_{new}} = \boldsymbol{v} * \sqrt{\frac{3\pi}{16}}.$$

This change our expected results for E_{kin} and E_{pot} as well to

$$E_{kin} = \frac{3\pi}{64}\frac{16}{3\pi} = \frac{1}{4},$$
$$E_{pot} = -\frac{3\pi}{32}\frac{16}{3\pi} = -\frac{1}{2},$$

for $G = M = a = 1$ and which results in a total energy of $E_{tot} = -1/2$ for large N-body systems.






```python
# Energy analysis
def calc_energy(N,m,x,v):
       kin = 0.;pot = 0.
       x_ij  = np.zeros_like(x[0])
       for i in range(N):
          kin += 1/2*m[i]*(norm(v[i])**2)
          l = [k for k in range(N) if k > i]
          for j in l:
            x_ij = x[j]-x[i]
            pot  -= (m[i]*m[j])/norm(x_ij)
       return (kin+pot)

def hist_energy(N,repeat):
    result = np.zeros(repeat)
    for i in range(repeat):
        m,x,v = plummer(N,0.99,16.0/(3*pi))
        result[i] = calc_energy(N,m,x,v)
    return result

# Histogram of energies # it takes same minutes
nbody = 5
obs   = 1000000
result = hist_energy(nbody,obs)

# log-log plot
import pylab as pl
data = -result
pl.figure(figsize=(8,5))
pl.hist(data, bins=np.logspace(np.log(0.001),np.log(4.0),100))
pl.gca().set_xscale("log")
pl.gca().set_yscale("log")
pl.title('log-log histogram of sampled energies E for a scaled \n' +\
         'Plummer model of N = %1.f' %nbody + ' and %1.f' %obs + ' observations')
pl.ylabel("log N")
pl.xlabel("log Energy E")
plt.grid()
plt.savefig("histogram_plummer.png")
pl.show()

# semi-log plot
import pylab as pl
data = result
pl.figure(figsize=(8,5))
pl.hist(data,100)
pl.gca().set_yscale("log")
pl.title('semi-log histogram of sampled energies E for a scaled \n' +\
         'Plummer model of N = %1.f' %nbody + ' and %1.f' %obs + ' observations')
pl.ylabel("log N")
pl.xlabel("Energy E")
plt.grid()
plt.savefig("histogram_plummer_semi.png")
pl.show()
```

h) BONUS: run the cluster with the N-body method you developed. Verify that it remains relatively stable, i.e. does not collapse or explode on very short times, and that the energy is conserved. If you run the cluster for a longer time, it should evaporate by ejecting stars through 3-body-interactions.

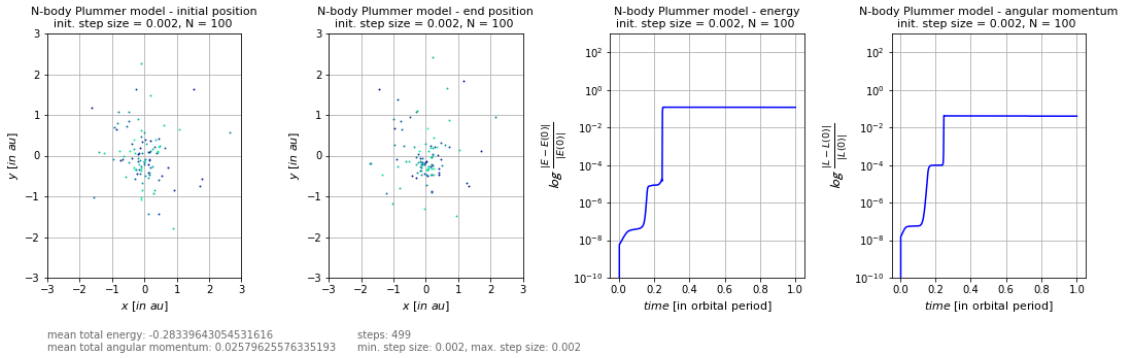
We have implemented a N-body Plummer model simulation with $N = 100$ and a step size of $s = 1/500$, where the Plummer model provides the initial values in standard units $G = M = a = 1$ (position coordinates are in barycentric coordinates). We further have implemented an adaptive step size configuration as well, where the time step may depend on the current curvature of the trajectory. In this case, the time step is calculated by the accelerations and the jerks, such that

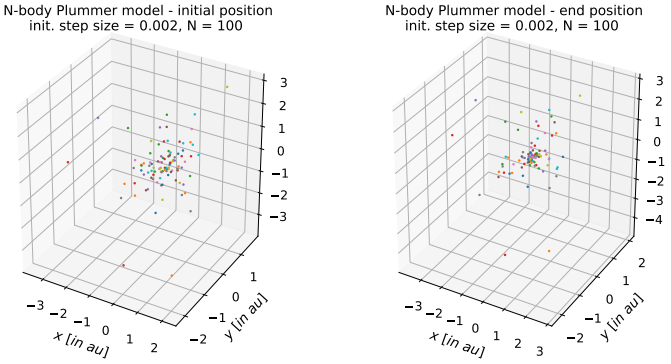
$$\Delta t_{acc} = \tilde{\eta} \min_{i=1,\dots,N} \frac{|\mathbf{a}_i(t_n)|}{|\dot{\mathbf{a}}_i(t_n)|}.$$

We also have implemented an adaptive step size simulation based on the position and the velocity

$$\Delta t_{vel} = \tilde{\eta} \min_{i=1,\dots,N} \frac{|\sum_{i \neq j} r_j - r_i|}{|\sum_{i \neq j} v_j - v_i|}.$$

This simulation uses larger time steps for particles that are far apart or moving slowly with smaller time steps when particles are closer together or moving quickly and is basically build on the same intuition as in the first adaptive time step approach. We see as expected that the total energy and the total angular momentum is conserved. The first two plots show the initial and the final position of our simulation.





In the appendix one can see the programmed Plummer modul and the N-body Plummer implementation with optional step size adapation.

Appendix Lab Project 1 - Code

Plummer modul:

```
# -*- coding: utf-8 -*-
"""
@author: Dawid Stepanovic - Plummer modul
"""
import numpy.random
import numpy as np
from math import pi

# Plummer modul
def plummer(number_of_particles,mass_cutoff, scaling = 1.):
    def calculate_radius():
        random_mass_fraction = np.random.uniform(1/number_of_particles,mass_cutoff,(
            number_of_particles,1))
        radius = 1.0/pow(pow(random_mass_fraction,-2.0/3.0)-1.0,1.0/2.0)
        return radius

    def position_spherical():
        radius = calculate_radius()
        theta  = numpy.arccos(np.random.uniform(-1.0,1.0,(number_of_particles,1)))
        phi    = np.random.uniform(0.0,2*pi,(number_of_particles,1))
        return (radius,theta,phi)

    def spherical_to_cartesian(vec):
        x = vec[0]*np.sin(vec[1])*np.cos(vec[2])
        y = vec[0]*np.sin(vec[1])*np.sin(vec[2])
        z = vec[0]*np.cos(vec[1])
        return (x,y,z)

    def velocity_position(number_of_particles):
        Nstore = 0
        x_value = numpy.zeros(0)
        y_value = numpy.zeros(0)
        while (Nstore < number_of_particles):
            x = np.random.uniform(0,1.0,(number_of_particles-Nstore))
            y = np.random.uniform(0,0.1,(number_of_particles-Nstore))
            p = (x**2)*pow(1.0-x**2.0,7.0/2.0)
            compare = y <= p
            x_value = np.concatenate((x_value,x.compress(compare)))
            y_value = np.concatenate((y_value,y.compress(compare)))
            Nstore  = len(x_value)
        return (np.atleast_2d(x_value).transpose(),np.atleast_2d(y_value).transpose())

    def velocity_spherical(radius):
        x,y = velocity_position(number_of_particles)
        velocity = x*np.sqrt(2.0)*pow(1.0+radius*radius,-1.0/4.0)
        theta = np.arccos(np.random.uniform(-1.0,1.0, (number_of_particles,1)))
        phi = np.random.uniform(0.0,2*pi,(number_of_particles,1))
        return (velocity,theta,phi)

    def centering(vec,mass):
        weight = (vec*mass)/sum(mass)
        center = vec - weight
        return center

    mass = np.zeros((number_of_particles,1))+(1.0/number_of_particles)
    radius, theta, phi = position_spherical()
    position = np.hstack(spherical_to_cartesian(position_spherical()))
    velocity = np.hstack(spherical_to_cartesian(velocity_spherical(radius)))
    position = position/scaling
    velocity = velocity/np.sqrt(1.0/scaling)

    # centering
    position = centering(position,mass)
    velocity = centering(velocity,mass)
    return (mass, position, velocity)
```

N-body Plummer:

```
# -*- coding: utf-8 -*-
"""
@author: Dawid Stepanovic - N-body Plummer
"""
from matplotlib import pyplot as plt
import numpy as np
import plummer_modul as pl
from numpy.linalg import norm
from math import pi
np.set_printoptions(precision=20)

method      = "rk4"                                 # implemented methods: "rk4", "rk4_acc", "rk4_vel"
nbody       = 100                                   # number of bodies
period      = 1                                     # period
step_val    = 1/500                                 # initial step size
norbit_val = 1                                      # number of orbits

def Nbodyproblem_Plumer(method,nbody,period,step_val,norbit_val):

    # general setting
    dt       = step_val                             # time step
    norbits = norbit_val                            # number of orbits
    au       = 1.49597870700*10**11                 # in m
```

```
    G_SI    = 6.67408*10**-11                   # gravitational constant [in m^3/(kg*s^2)]
    daysec  = 86400.                            # in seconds
    solar   = 1.988544*10**30                   # in kg

    # convert the gravitational constant
    G_new = G_SI/au**3*solar*(daysec)**2/np.square(2*np.pi/365.2568983263281) # ~ 1

    # variable setting
    if method == "rk4":
        steps = int(norbits*(period/dt))
    else:
        steps = int(norbits*(period/dt))*100

    t = np.zeros(steps); dtstore = np.zeros(steps); energy = np.zeros(steps); moment  = np.zeros(steps
        )
    a_time = np.zeros(nbody)

    # r array stores the steps, the number of bodies in the system and the (x,y,z) array
    r  = np.zeros((steps,nbody,3))
    v  = np.zeros_like(r)

    # gravitational constant*mass
    mass,pos,vel = pl.plummer(nbody,0.99,16.0/(3*pi))
    m = mass
    # initial position and velocity
    for i in range(nbody):
            r[0,i,:] = pos[i];
            v[0,i,:] = vel[i];

    # acceleration for j != i
    def acceleration(i,r):
      a     = np.zeros_like(r[0])
      r_ij  = np.zeros_like(r[0])
      l = [k for k in range(nbody) if k != i]
      for j in l:
        r_ij = r[j]-r[i]
        a += m[j]/(norm(r_ij))**3*r_ij
      return a

    # jerk for j != i
    def acceleration_dot(i,r,v):
      a_dot = np.zeros_like(r[0])
      r_ij  = np.zeros_like(r[0])
      v_ij  = np.zeros_like(r[0])
      l = [k for k in range(nbody) if k != i]
      for j in l:
        r_ij = r[j]-r[i]
        v_ij = v[j]-v[i]
        a_dot += m[j]/(norm(r_ij))**3*v_ij-(3*np.dot(r_ij,v_ij)/(norm(r_ij))**5*r_ij)
      return a_dot

    # total (kinetic + potential) energy
    def calc_energy(r,v):
      kin = 0.;pot = 0.
      r_ij  = np.zeros_like(r[0])
      for i in range(nbody):
        kin += 1/2*m[i]*(norm(v[i])**2)
        l = [k for k in range(nbody) if k > i]
        for j in l:
          r_ij = r[j]-r[i]
          pot  -= (m[i]*m[j])/norm(r_ij)
      return (kin+pot)/G_new

    # total angular momentum
    def calc_moment(r,v):
      L = np.zeros_like(r[0])
      for i in range(nbody):
        L += m[i]/G_new*np.cross(r[i],v[i])
      return norm(L)

    def diff(i,vr):
      vr_ij  = np.zeros_like(vr[0])
      l = [k for k in range(nbody) if k != i]
      for j in l:
        vr_ij += vr[j]-vr[i]
      return vr_ij

    # Barycenter
    def barycenter(r,v):
        rbar = np.zeros(3)
        vbar = np.zeros(3)
        for i in range(nbody):
            rbar += m[i]*r[i]
            vbar += m[i]*v[i]
        for i in range(nbody):
                r[i] = r[i]-(rbar/G_new)/np.sum(m/G_new)
                v[i] = v[i]-(vbar/G_new)/np.sum(m/G_new)
        return rbar, vbar

    # initial total energy
    energy[0] = calc_energy(r[0],v[0])

    # initial angular momentum
    moment[0] = calc_moment(r[0],v[0])

    # general settings for the RK-methods
    n = 0
```

```
    t[n] = dt
    dtstore[n] = dt

    k1 = np.zeros((nbody,6))
    k2 = np.zeros_like(k1)
    k3 = np.zeros_like(k1)
    k4 = np.zeros_like(k1)

    # initial barycenter values
    barycenter(r[0],v[0])

    if(method == "rk4"):
        for n in range(steps-1):
            for i in range(nbody):
                k1[:,0:3] = dt*v[n,:]
                k1[i,3:6] = dt* acceleration(i,r[n])

                k2[:,0:3] = dt*(v[n,:]+0.5*k1[:,3:6])
                k2[i,3:6] = dt*acceleration(i,r[n]+0.5*k1[:,0:3])

                k3[:,0:3] = dt*(v[n,:]+0.5*k2[:,3:6])
                k3[i,3:6] = dt*acceleration(i,r[n]+0.5*k2[:,0:3])

                k4[:,0:3] = dt*(v[n,:]+k3[:,3:6])
                k4[i,3:6] = dt*acceleration(i,r[n]+k3[:,0:3])

                # next acceleration and position
                r[n+1,i]  = r[n,i]+1/6*(k1[i,0:3]+2*k2[i,0:3]+2*k3[i,0:3]+k4[i,0:3])
                v[n+1,i]  = v[n,i]+1/6*(k1[i,3:6]+2*k2[i,3:6]+2*k3[i,3:6]+k4[i,3:6])

            # barycenter
            barycenter(r[n+1],v[n+1])
            # compute total energy of system
            energy[n+1] = calc_energy(r[n+1],v[n+1])
            # compute angular momentum of system
            moment[n+1] = calc_moment(r[n+1],v[n+1])
            # time step
            t[n+1] = t[n]+dt
            dtstore[n+1] = dt
            n = n+1
            print('t:',t[n])

    elif(method == "rk4_vel"):
        while(t[n]<period*norbits):
            for i in range(nbody):
                k1[:,0:3] = dt*v[n,:]
                k1[i,3:6] = dt* acceleration(i,r[n])

                k2[:,0:3] = dt*(v[n,:]+0.5*k1[:,3:6])
                k2[i,3:6] = dt*acceleration(i,r[n]+0.5*k1[:,0:3])

                k3[:,0:3] = dt*(v[n,:]+0.5*k2[:,3:6])
                k3[i,3:6] = dt*acceleration(i,r[n]+0.5*k2[:,0:3])

                k4[:,0:3] = dt*(v[n,:]+k3[:,3:6])
                k4[i,3:6] = dt*acceleration(i,r[n]+k3[:,0:3])

                # next acceleration and position
                r[n+1,i]  = r[n,i]+1/6*(k1[i,0:3]+2*k2[i,0:3]+2*k3[i,0:3]+k4[i,0:3])
                v[n+1,i]  = v[n,i]+1/6*(k1[i,3:6]+2*k2[i,3:6]+2*k3[i,3:6]+k4[i,3:6])

            # barycenter
            barycenter(r[n+1],v[n+1])
            # compute total energy of system
            energy[n+1] = calc_energy(r[n+1],v[n+1])
            # compute angular momentum of system
            moment[n+1] = calc_moment(r[n+1],v[n+1])
            # time step
            for i in range(nbody):
                a_time[i] = norm(diff(i,r[n+1]))/norm(diff(i,v[n+1]))

            multiplier = 10
            if((np.min(step_val*a_time[a_time > 0])*multiplier)/(t[n]-t[n-1])> 10):
                dt = dt
            else:
                dt = np.min(step_val*a_time[a_time > 0])*multiplier
            t[n+1] = t[n]+dt
            dtstore[n+1] = dt
            n = n+1
            print('t:',t[n])

    elif(method == "rk4_acc"):
        while(t[n]<period*norbits):
            for i in range(nbody):
                k1[:,0:3] = dt*v[n,:]
                k1[i,3:6] = dt* acceleration(i,r[n])

                k2[:,0:3] = dt*(v[n,:]+0.5*k1[:,3:6])
                k2[i,3:6] = dt*acceleration(i,r[n]+0.5*k1[:,0:3])

                k3[:,0:3] = dt*(v[n,:]+0.5*k2[:,3:6])
                k3[i,3:6] = dt*acceleration(i,r[n]+0.5*k2[:,0:3])

                k4[:,0:3] = dt*(v[n,:]+k3[:,3:6])
                k4[i,3:6] = dt*acceleration(i,r[n]+k3[:,0:3])

                # next acceleration and position
```

```
                r[n+1,i]   = r[n,i]+1/6*(k1[i,0:3]+2*k2[i,0:3]+2*k3[i,0:3]+k4[i,0:3])
                v[n+1,i]   = v[n,i]+1/6*(k1[i,3:6]+2*k2[i,3:6]+2*k3[i,3:6]+k4[i,3:6])

            # barycenter
            barycenter(r[n+1],v[n+1])
            # compute total energy of system
            energy[n+1] = calc_energy(r[n+1],v[n+1])
            # compute angular momentum of system
            moment[n+1] = calc_moment(r[n+1],v[n+1])
            # time step
            for i in range(nbody):
                a_time[i] = norm(acceleration(i,r[n+1]))/norm(acceleration_dot(i,r[n+1],v[n+1]))

            multiplier = 1e8
            if((np.min(step_val*a_time[a_time > 0])*multiplier)/(t[n]-t[n-1])> 2):
                dt = dt
            else:
                dt = np.min(step_val*a_time[a_time > 0])*multiplier
            t[n+1] = t[n]+dt
            dtstore[n+1] = dt
            n = n+1
            print('t:',t[n])

    # extract relevant non-zero values
    r=r[:n+1]
    t=t[:n+1]
    dtstore=dtstore[:n+1]
    energy=energy[:n+1]
    moment=moment[:n+1]

    # plots
    # figure 1 - orbit
    plt.figure(figsize=(18,5))
    plt.subplots_adjust(wspace = 0.45,left=1/9, right=1-1/9, bottom=1/4.8, top=1-1/7.5)
    plt.subplot(141)
    for i in range(nbody):
        plt.plot(r[0,i,0],r[0,i,1],color=(0,i/nbody,0.6,1),linewidth=.5,\
                 marker='h',markersize=1, markevery=1)

    plt.title(r"N-body Plummer model - initial position""\n"\
              "init. step size = %1.3f, " %step_val +\
               "N = %1.f " %nbody,fontsize=11)
    plt.xlabel(r"$x~[in~au]$",fontsize=11)
    plt.ylabel(r"$y~[in~au]$",fontsize=11)

    plt.ylim([-3,3])
    plt.xlim([-3,3])
    plt.grid()

    col = 'dimgrey'
    plt.annotate('mean total energy: '+str(np.mean(energy))\
                 , xy=(0.0, -0.25), xycoords='axes fraction',color=col)
    plt.annotate('mean total angular momentum: '+str(np.mean(moment))\
                 , xy=(0.0, -0.3), xycoords='axes fraction',color=col)
    plt.annotate('steps: '+str(n),xy=(1.6, -0.25), xycoords='axes fraction',color=col)
    plt.annotate('min. step size: '+str(np.min(dtstore))+\
                 ', max. step size: '+str(np.max(dtstore)),\
                 xy=(1.6, -0.3), xycoords='axes fraction',color=col)

    plt.subplot(142)
    for i in range(nbody):
        plt.plot(r[n,i,0],r[n,i,1],color=(0,i/nbody,0.6,1),linewidth=.5,\
                 marker='h',markersize=1, markevery=1)
    plt.title(r"N-body Plummer model - end position""\n"\
              "init. step size = %1.3f, " %step_val +\
               "N = %1.f " %nbody,fontsize=11)
    plt.xlabel(r"$x~[in~au]$",fontsize=11)
    plt.ylabel(r"$y~[in~au]$",fontsize=11)

    plt.ylim([-3,3])
    plt.xlim([-3,3])
    plt.grid()

    # figure 3 - total energy
    plt.subplot(143)
    plt.semilogy(t/period,abs(energy-energy[0])/abs(energy[0]),'k',color='b')
    plt.title(r"N-body Plummer model - energy""\n"\
              "init. step size = %1.3f, " %step_val +\
               "N = %1.f " %nbody,fontsize=11)
    plt.xlabel(r"$time~[{\rm in~orbital~period}]$",fontsize=11)
    plt.ylabel(r"$\mathcal{log}~\frac{|E-E(0)|}{|E(0)|}$",fontsize=13.5)
    plt.ylim([1e-10,1000])
    plt.grid()

    # figure 4 - total angular momentum
    plt.subplot(144)
    plt.semilogy(t/period,abs(moment-moment[0])/abs(moment[0]),'k',color='b')
    plt.title(r"N-body Plummer model - angular momentum""\n"\
              "init. step size = %1.3f, " %step_val +\
               "N = %1.f " %nbody,fontsize=11)
    plt.xlabel(r"$time~[{\rm in~orbital~period}]$",fontsize=11)
    plt.ylabel(r"$\mathcal{log}~\frac{|L-L(0)|}{|L(0)|}$",fontsize=13.5)
    plt.ylim([1e-10,1000])
    plt.grid()

    # save figures as pdf
    plt.savefig("Nbody_plummer.png")
```

```
    plt.show()

    # 3D plots
    # figure 1 - initial position
    fig = plt.figure(figsize=(10,5))
    ax = fig.add_subplot(1,2,1, projection='3d')
    for i in range(nbody):
        ax.scatter3D(r[0,i,0],r[0,i,1],r[0,i,2],lw=0,s=4)
    plt.title(r"N-body Plummer model - initial position""\n"\
              "init. step size = %1.3f, " %step_val +\
               "N = %1.f " %nbody,fontsize=11)
    plt.xlabel(r"$x~[in~au]$",fontsize=11)
    plt.ylabel(r"$y~[in~au]$",fontsize=11)

    # figure 2 - end position
    ax = fig.add_subplot(1,2,2, projection='3d')
    for i in range(nbody):
        ax.scatter3D(r[n,i,0],r[n,i,1],r[n,i,2],lw=0,s=4)
    plt.title(r"N-body Plummer model - end position""\n"\
          "init. step size = %1.3f, " %step_val +\
           "N = %1.f " %nbody,fontsize=11)
    plt.xlabel(r"$x~[in~au]$",fontsize=11)
    plt.ylabel(r"$y~[in~au]$",fontsize=11)
    plt.savefig("Nbody_plummer_3D.pdf")
    plt.show()

Nbodyproblem_Plumer(method,nbody,period,step_val,norbit_val)
```