Calderon_Tianna_PS4

April 11, 2025

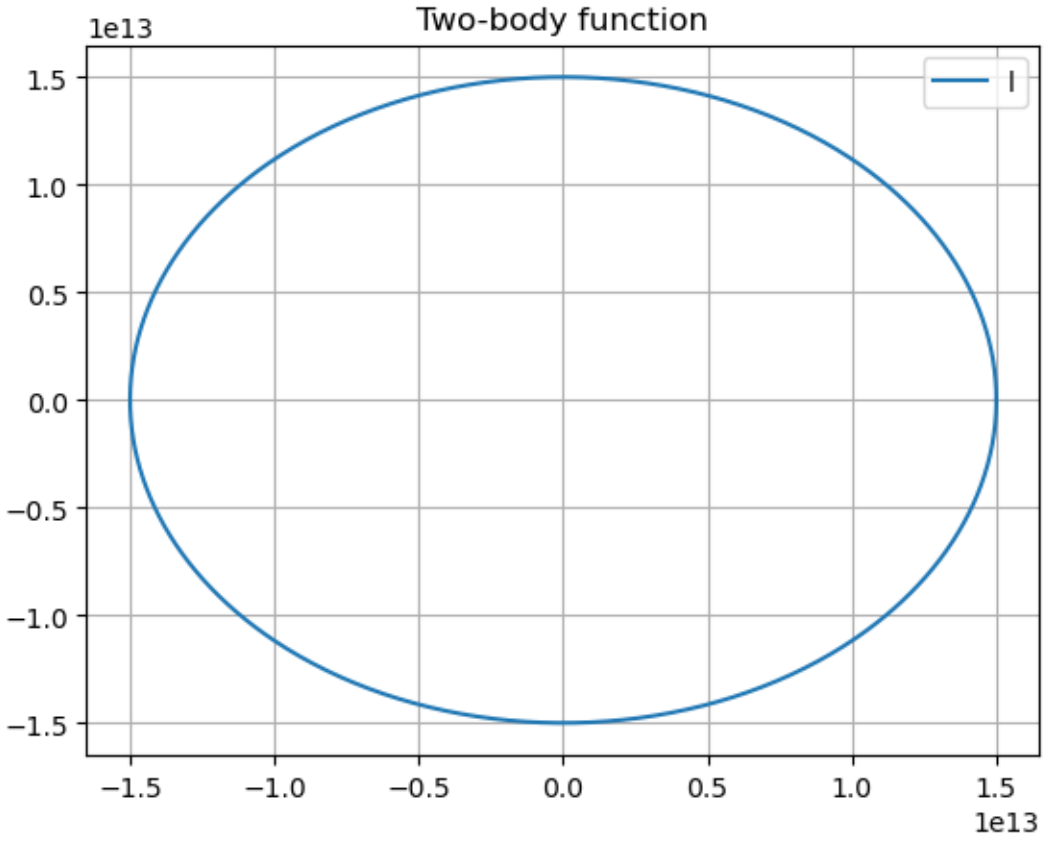
[1]:
```
import numpy as np
import my_numerical_methods as nm
import my_ode_solvers as os
import matplotlib.pyplot as plt
import my_star_functions as sf
from mpl_toolkits.mplot3d import Axes3D
from my_astro_constants import *
import my_nbody_methods as nbody

from numba import njit
```

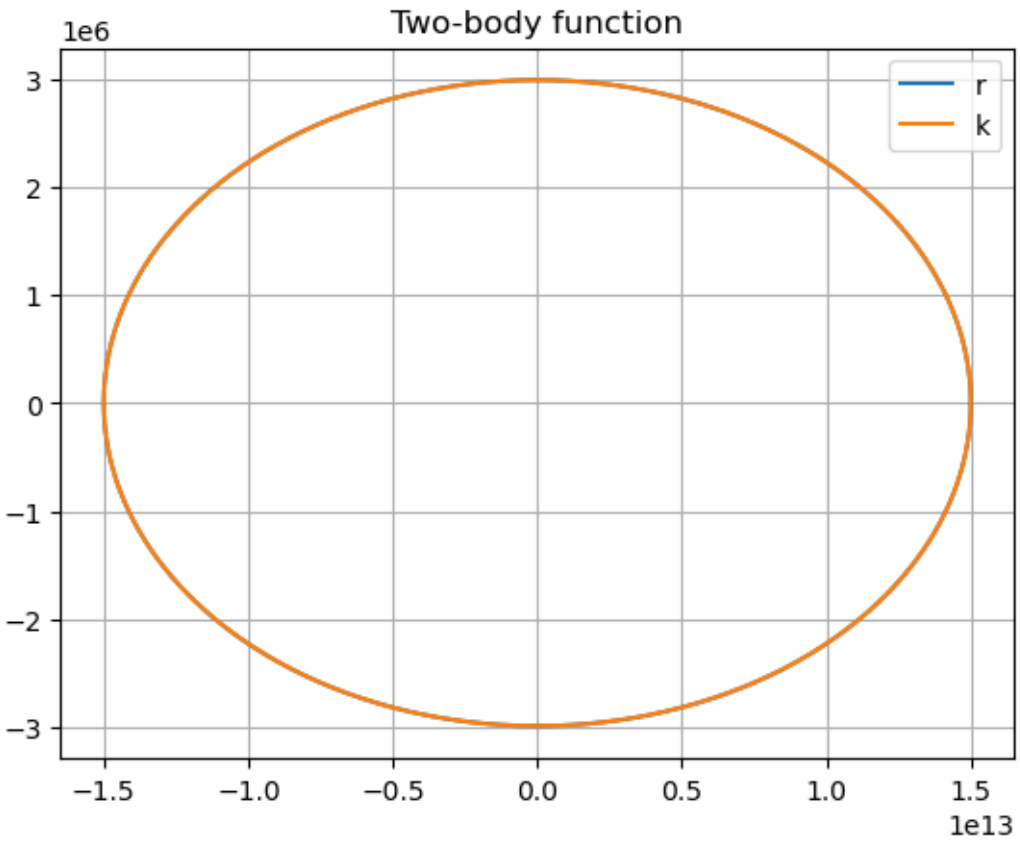
[2]:
```
#test case 1: two-body test: sun-earth system
#initial conditions
m1 = earth_mass
m2 = solar_mass
r_cm = np.array([au, 0.0])  # 1 AU along x-axis in cm
v_c = np.array([0.0, nbody.velocity_circ(m2, np.linalg.norm(r_cm))])  #␣
 ↪velocity along y-axis

#initial state vector [pos, velo]
y0 = np.array([r_cm, v_c], dtype=float)
print(y0.shape)

#function for n_body as an ode..
def f(t, y):
    position = y[0]
    velocity = y[1]

    a = nbody.n_body(m2, np.array([0.0, 0.0]), position)
    return np.array([velocity, a])

#simulation parameters
N = 1000
p = nbody.period(r_cm[0], v_c[1])  # 1 orbital period
dt = p/N

t0 = 0
```

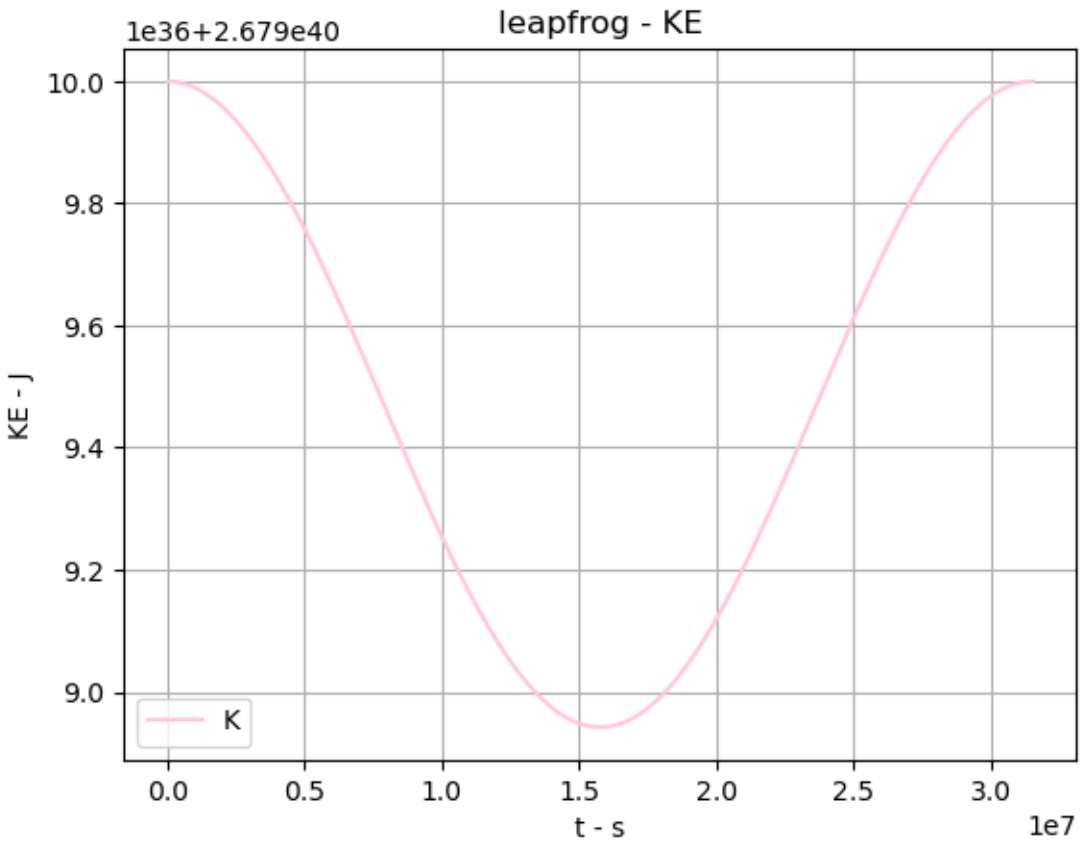
```
tN = p
# Create result array
results = np.zeros((N+1, 2, 2))

solver = os.ODE_solver(func=f, state=y0, dt=dt, arr=results)
t, results_lf = solver.leapfrog(y0, t0=0, tN=p, N=N)

positions = results_lf[0]
velocities = results_lf[1]

x = positions[:, 0]
y = positions[:, 1]


#rk4
results_rk4 = np.zeros((N+1, 2))
results_rk4[0] = y0[0]
results_rk4 = solver.integrate(t0=0, tstop=p, integrator='runge-kutta')


x_rk4 = results_rk4[:, 0]
y_rk4 = results_rk4[:, 1]

#plot
sf.plot_func(x, y, label = "leapfrog integration" , Title = "Two-body function")
sf.plot_func(x_rk4, y_rk4, label = "rk4 integration" , Title = "Two-body␣
 ↪function")


#energy tests
energies = np.zeros((3, N+1))  # 3 rows: KE, PE, TE
energies_rk4 = np.zeros((3, N))


energies = nbody.energy_calc(energies, positions, velocities, m1, m2)
energies_rk4 = nbody.energy_calc(energies_rk4, results_rk4, velocities, m1, m2)
t_rk4 = np.linspace(t0, tN, num = len(energies_rk4[0]))


#plotting energy test.. leapfrog
sf.plot_func(t, energies[0], xlabel = "t - s", ylabel = "KE - J", label =␣
 ↪"Kinetic Energy", color = "#ffc8dd", Title = "leapfrog - KE")
sf.plot_func(t, energies[1], xlabel = "t - s", ylabel = "KE - J", label =␣
 ↪"Potential Energy", color = "#bde0fe", Title = "leapfrog - PE")
sf.plot_func(t, energies[2], xlabel = "t - s", ylabel = "KE - J", label =␣
 ↪"Total Energy", color ="#8A4452", Title = "leapfrog - TE")
```

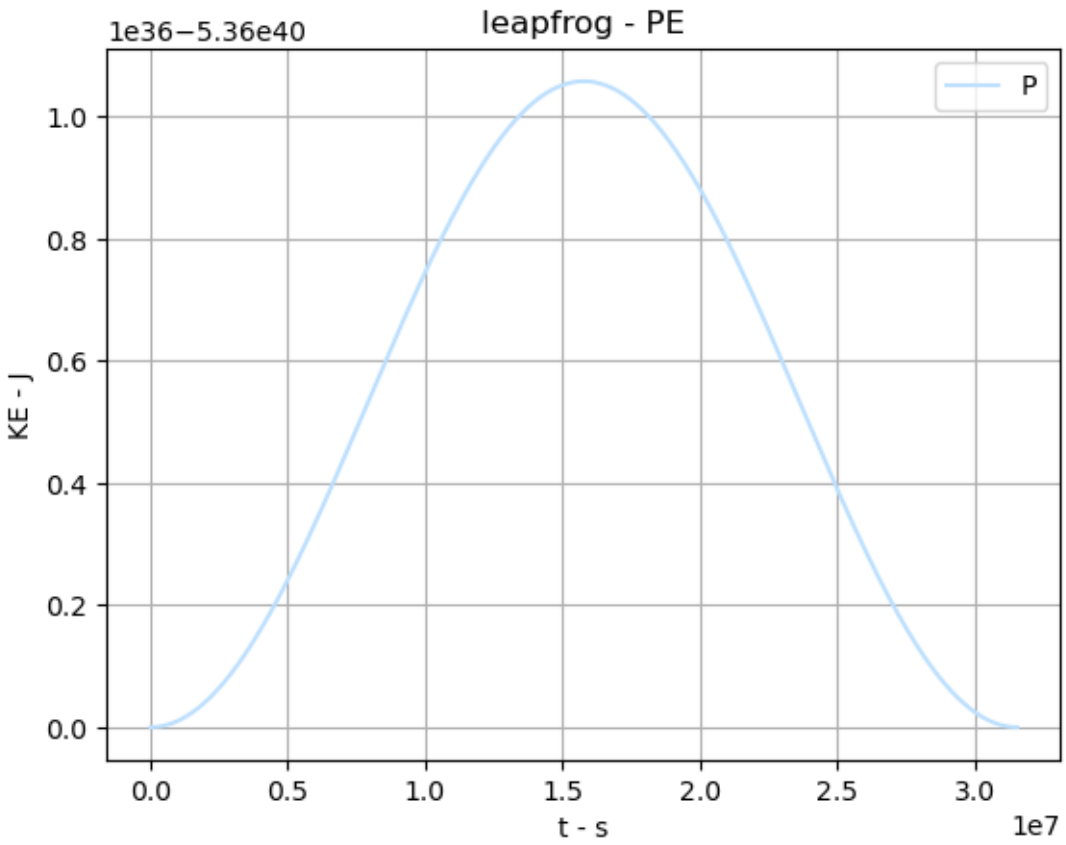
```
#rk-4
sf.plot_func(t_rk4, energies_rk4[0], xlabel = "t - s", ylabel = "KE - J", label␣
 ↪= "Kinetic Energy", color = "#ffc8dd", Title = "RK4 - KE")
sf.plot_func(t_rk4, energies_rk4[1], xlabel = "t - s", ylabel = "KE - J", label␣
 ↪= "Potential Energy", color = "#bde0fe", Title = "RK4 - PE")
sf.plot_func(t_rk4, energies_rk4[2], xlabel = "t - s", ylabel = "KE - J", label␣
 ↪= "Total Energy", color ="#8A4452",Title = "RK4 - TE")
```

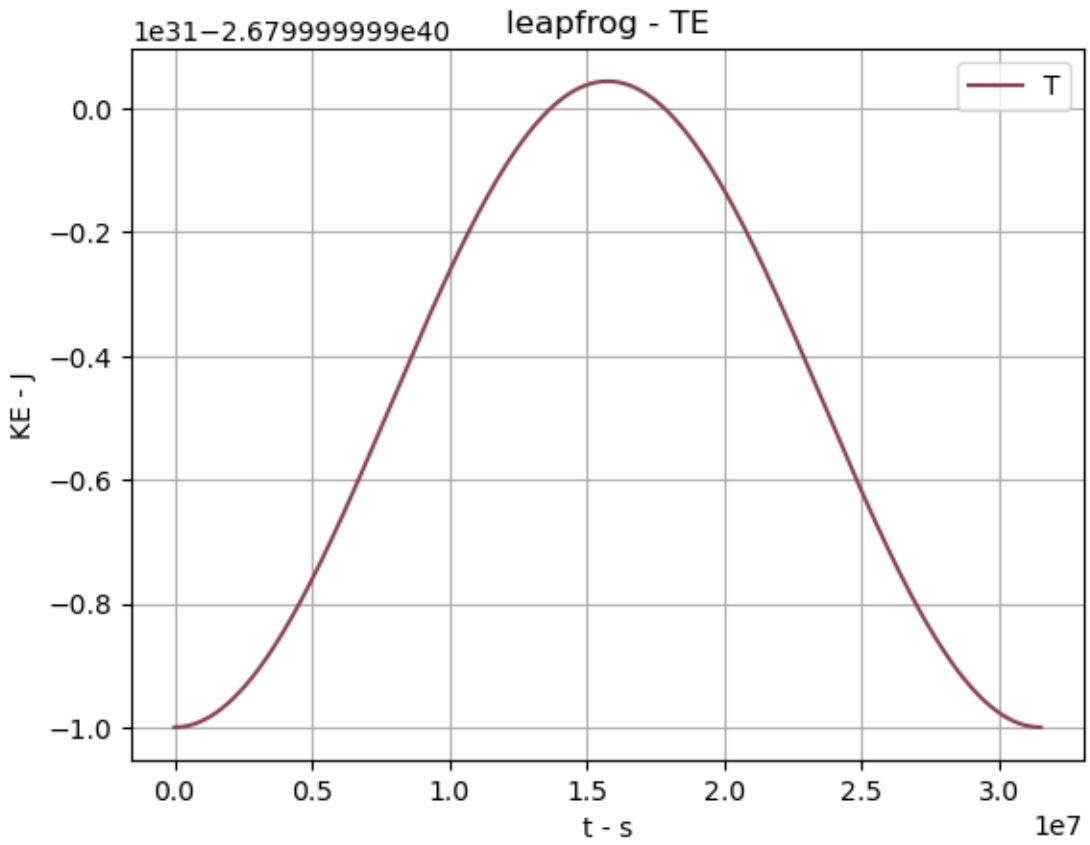
```
(2, 2)
```

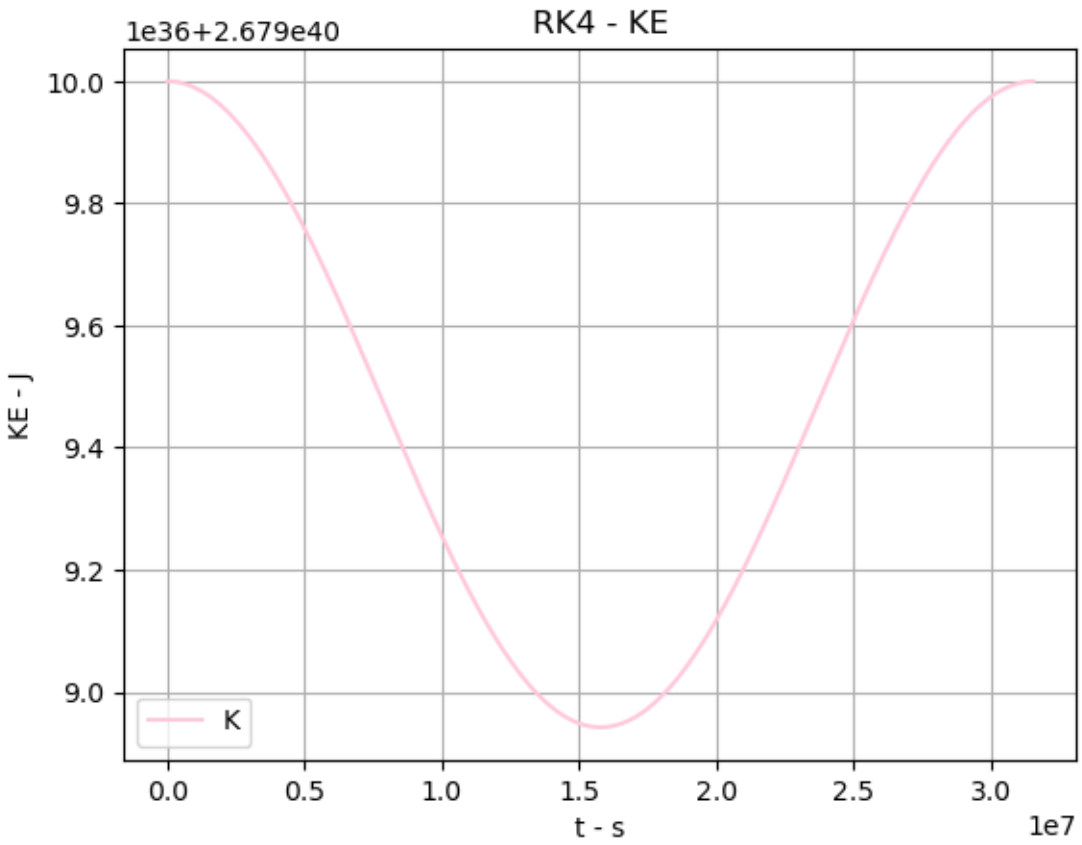


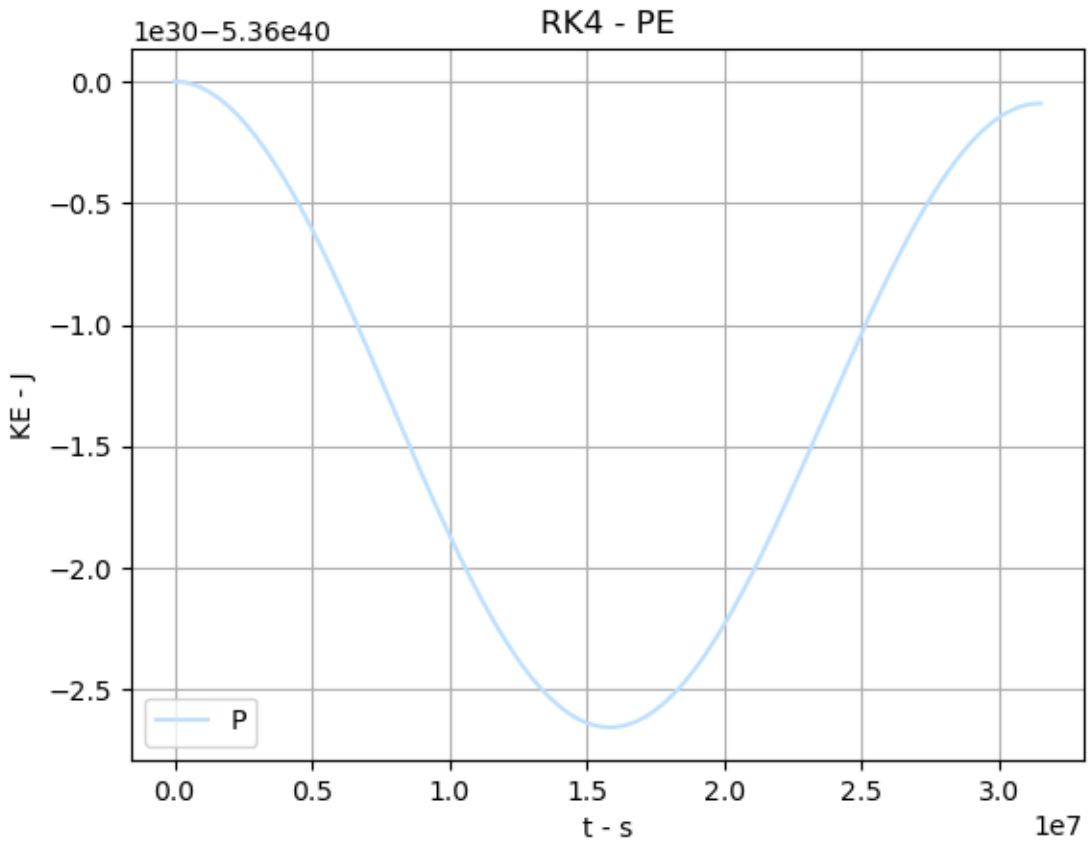

Two-body function
1e6
3
2
1
0
−1
−2
−3
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
1e13
r
k

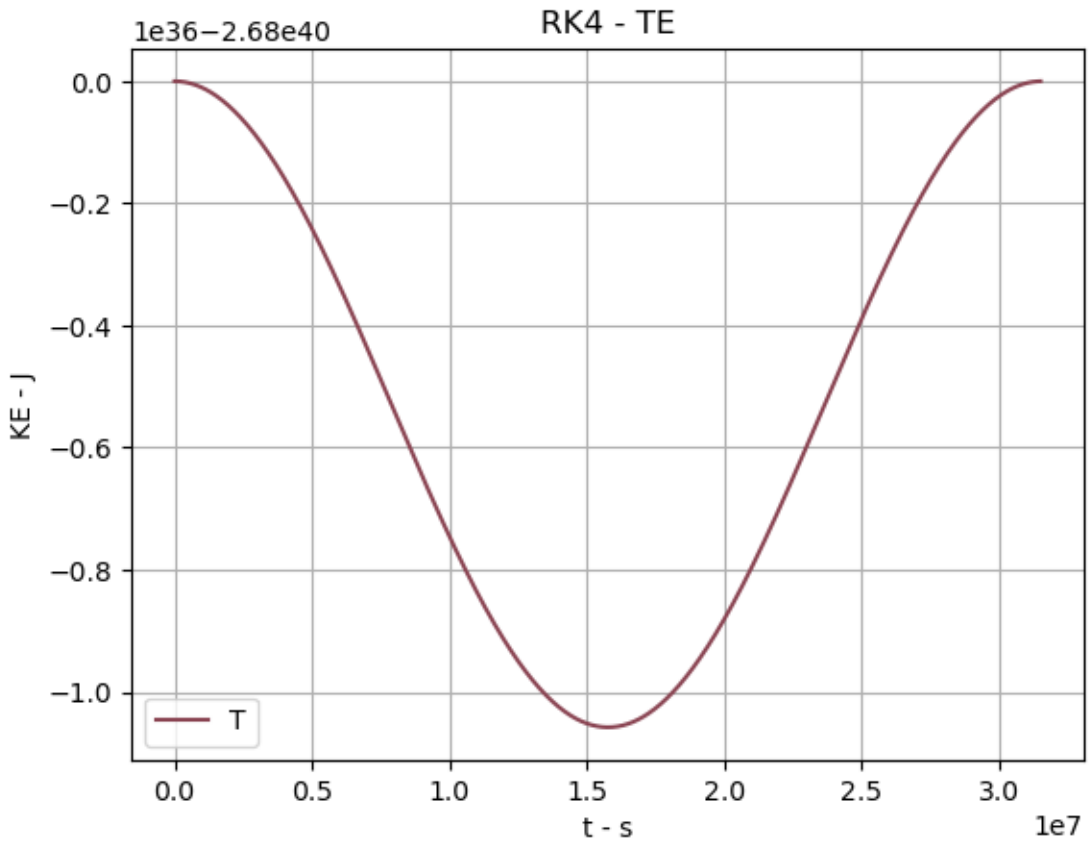

leapfrog - KE
1e36+2.679e40
10.0
9.8
9.6
9.4
9.2
9.0
KE - J
K
0.0
0.5
1.0
1.5
2.0
2.5
3.0
t - s
1e7


leapfrog - PE
1e36−5.36e40
P
1.0
0.8
0.6
0.4
0.2
0.0
KE - J
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e7
t - s


leapfrog - TE
1e31−2.679999999e40
T
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
KE - J
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e7
t - s


RK4 - KE
1e36+2.679e40
10.0
9.8
9.6
9.4
9.2
9.0
KE - J
K
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e7
t - s


RK4 - PE
1e30−5.36e40
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
KE - J
P
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e7
t - s



0.1 Review of the Two-Body Test

1) Which method is more accurate over short timescales? After doing some research and reviewing the lectures, Runge-Kutta(4) is more accurate at shorter time scales. It uses a error per step of O(h^4) for total accumulated error. Compared to local of O(h^5). Found from (https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods)

   It does a weighted average (due to this it might slows things down as this can create drift because the equation is looking at a small fraction of the overall equation to make averages).

2) Which method is better at conserving energy over long timescales?

   Leapfrog is better at conserving energy, it takes into consideration the implications certain variables have for their designated equation. It doesn't show artificial damping or growth because it handles the "bigger picture" of equations. However, it will lose some local accuracy to make sure that already determined points are in the same geometric area/path. Due to this, it allows for equations like in Hamiltonian systems to be preserved as it doesn't change the volume/area of the enivorment in phase space, it keeps the parameters of the equation intact.

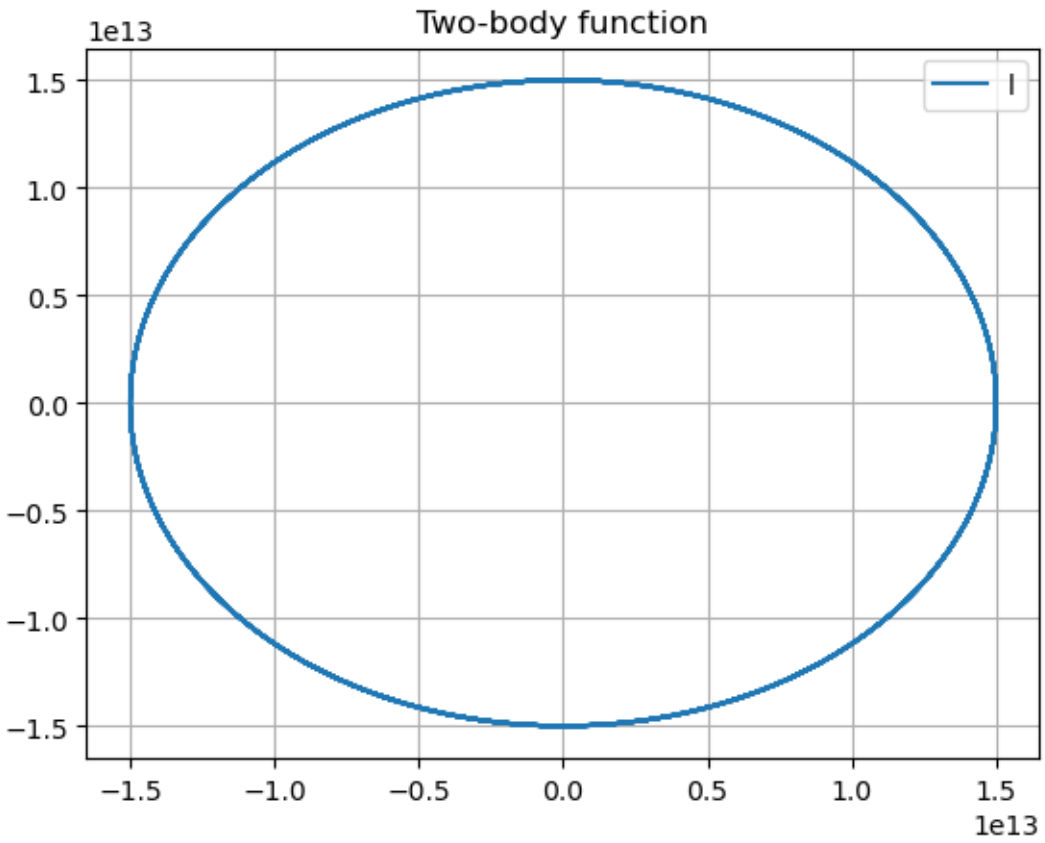
3) Explain your findings based on energy conservation and the behavior of each method.

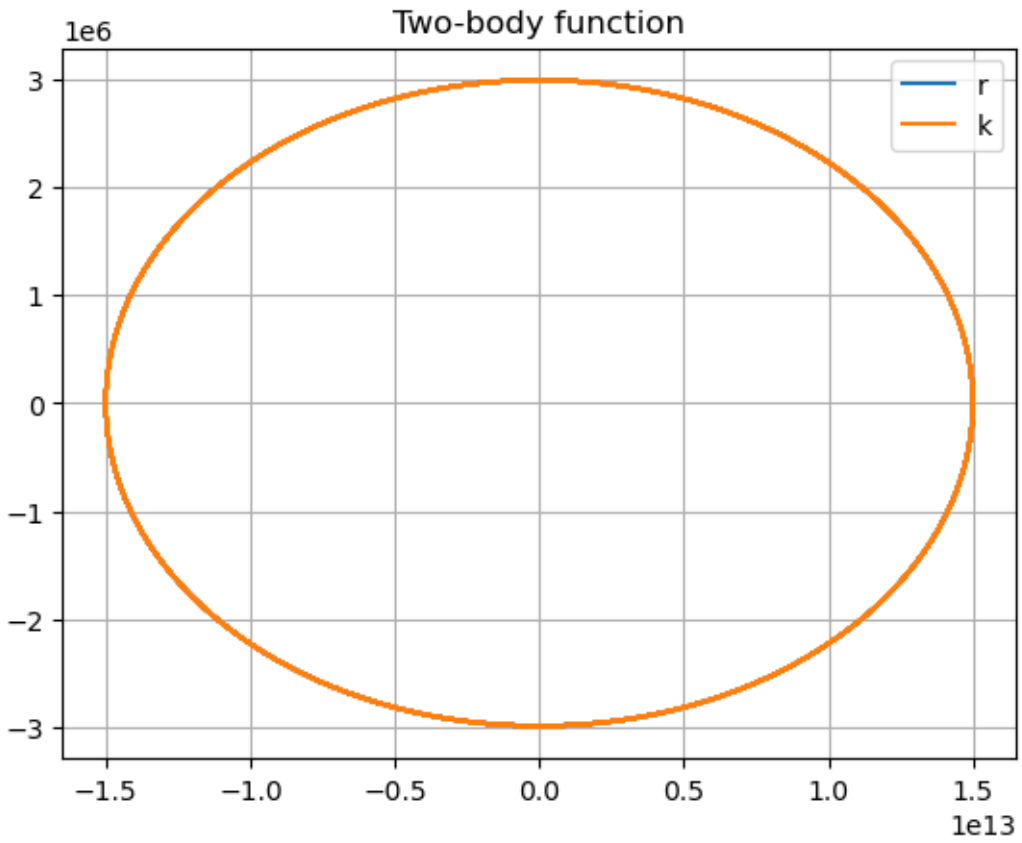
   Looking at the energy conservation between leapfrog and RK-4 I can see that the range for

both of them is the same which means that at least the range for KE was correct. There is some deviation in the PE and TE in terms of scale. I think this was probably how I am recieving the values for leapfrog and rk-4 if you look at the y-axis for leapfrog it is on a scale of 1e13 for the x,y plot while RK-4 for x,y plot is 1E6.

[4]:
```
#stability tests
tN = 10*p
N = 10000
dt = tN / N
results_10 = np.zeros((N+1, 2, 2))

#leapfrog
solver = os.ODE_solver(func=f, state=y0, dt=dt, arr=results_10)
t, output = solver.leapfrog(y0, t0=0, tN=tN, N=N)
%timeit t, output = solver.leapfrog(y0, t0=0, tN=tN, N=N)

positions = output[0]
velocities = output[1]

x = positions[:, 0]
y = positions[:, 1]

#rk-4
solver = os.ODE_solver(func=f, state=y0, dt=dt, arr=results_10)
results_rk4 = solver.integrate(t0=0, tstop=tN, integrator='runge-kutta')
%timeit results_rk4 = solver.integrate(t0=0, tstop=tN, integrator='runge-kutta')


x_rk4 = results_rk4[:, 0]
y_rk4 = results_rk4[:, 1]

#plot
sf.plot_func(x, y, label = "leapfrog integration" , Title = "Two-body function")
sf.plot_func(x_rk4, y_rk4, label = "rk4 integration" , Title = "Two-body␣
 ↪function")
```

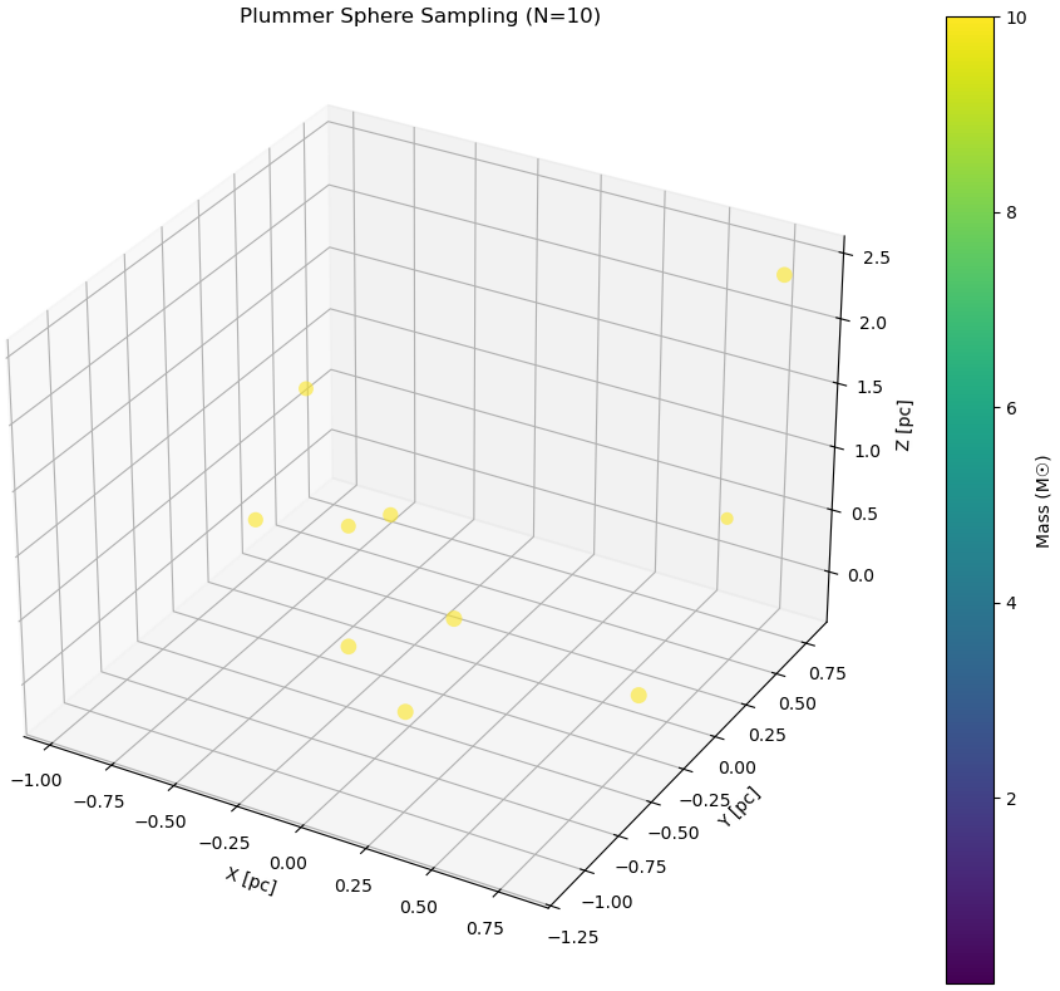
```
548 ms ± 80.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.69 s ± 114 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

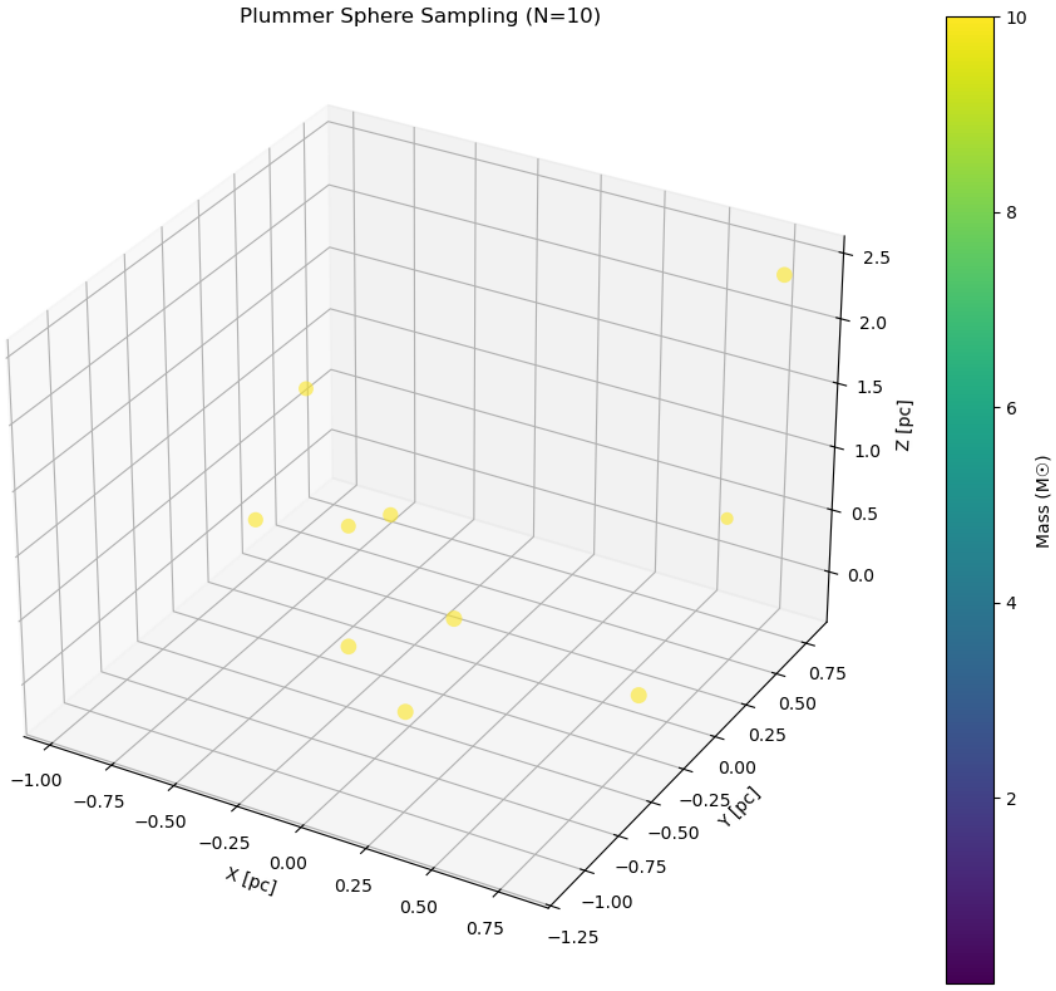

Two-body function
1e13
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
1e13
I

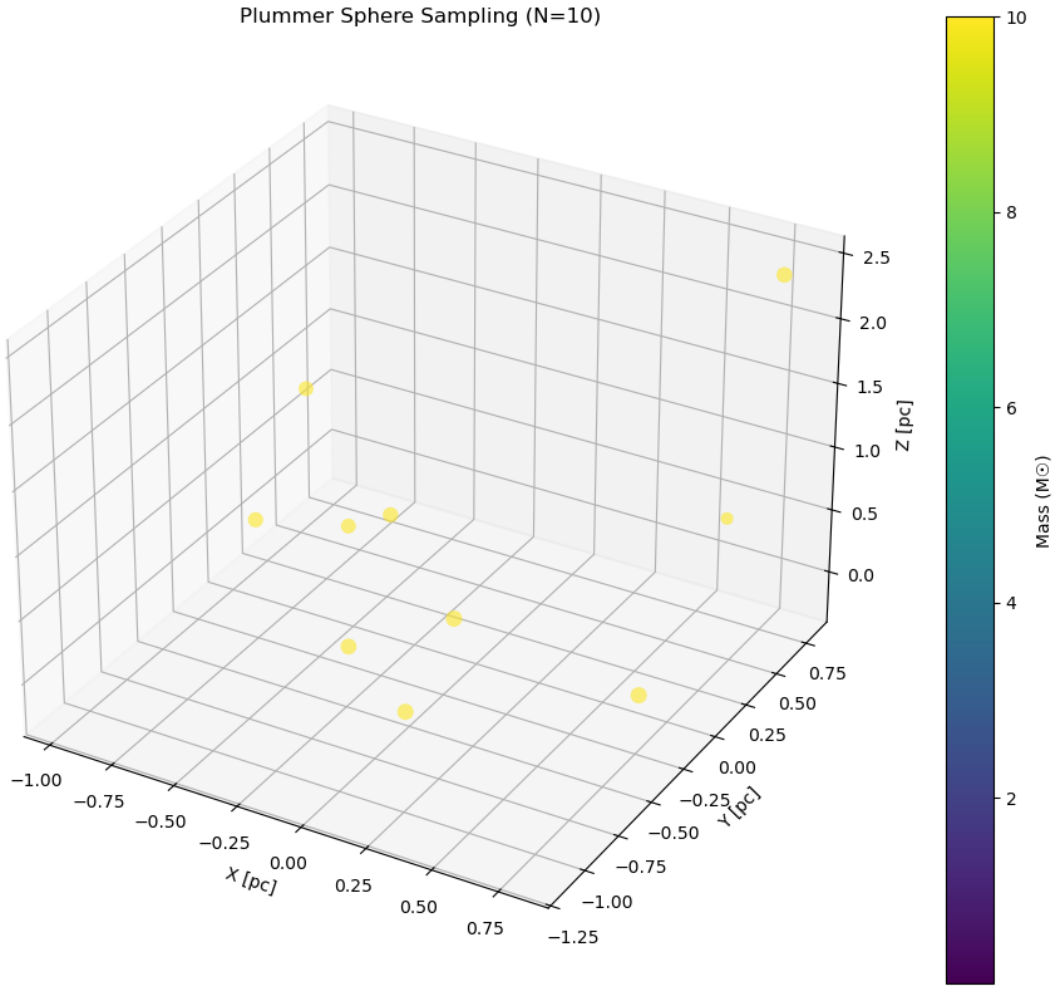


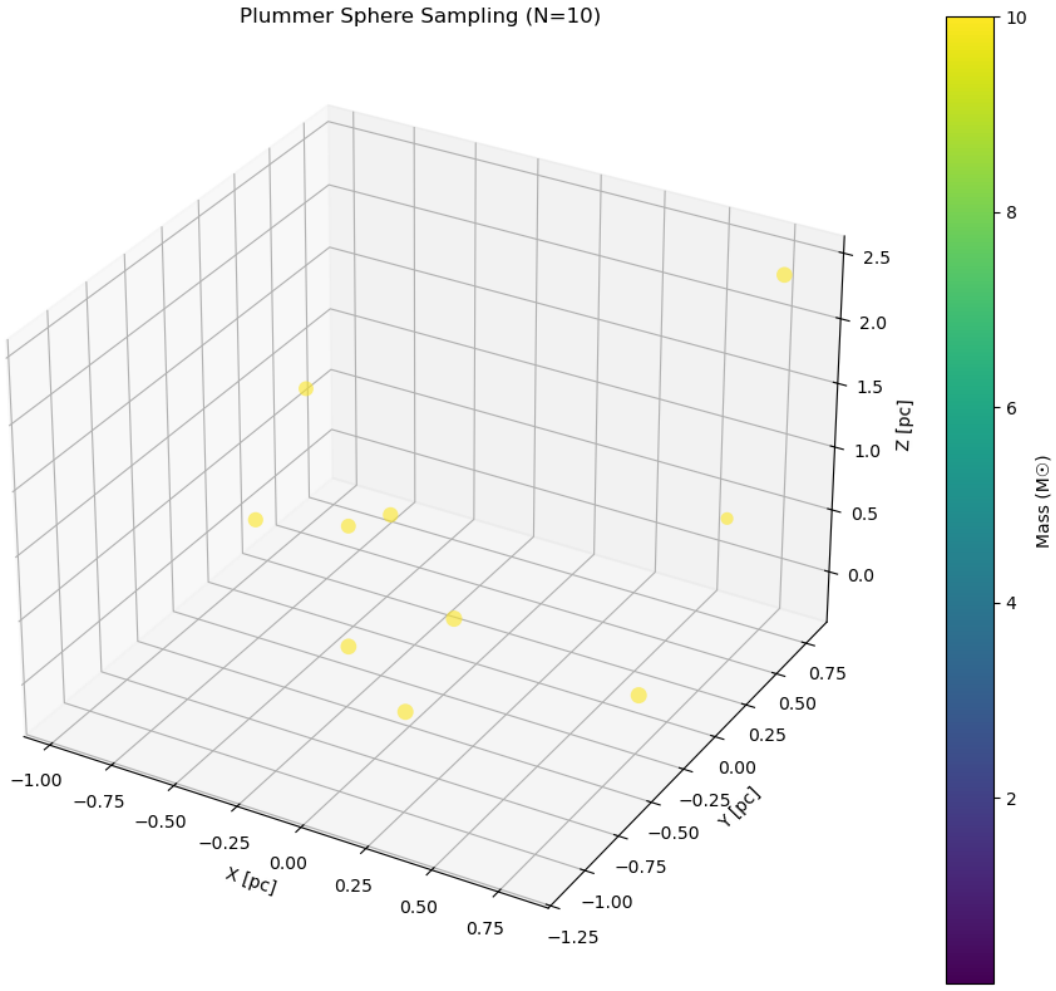
[5]:
```
#monte-carlo sampling
n = [10, 50, 100, 500, 1000]
for i in range(len(n)):
    pos, vel, masses = nbody.sample_plummer(n[i], M_total=500.0)
    %timeit nbody.plot_plummer_sphere(pos, masses, save_path='plummer_sphere.
 ↪png')
```

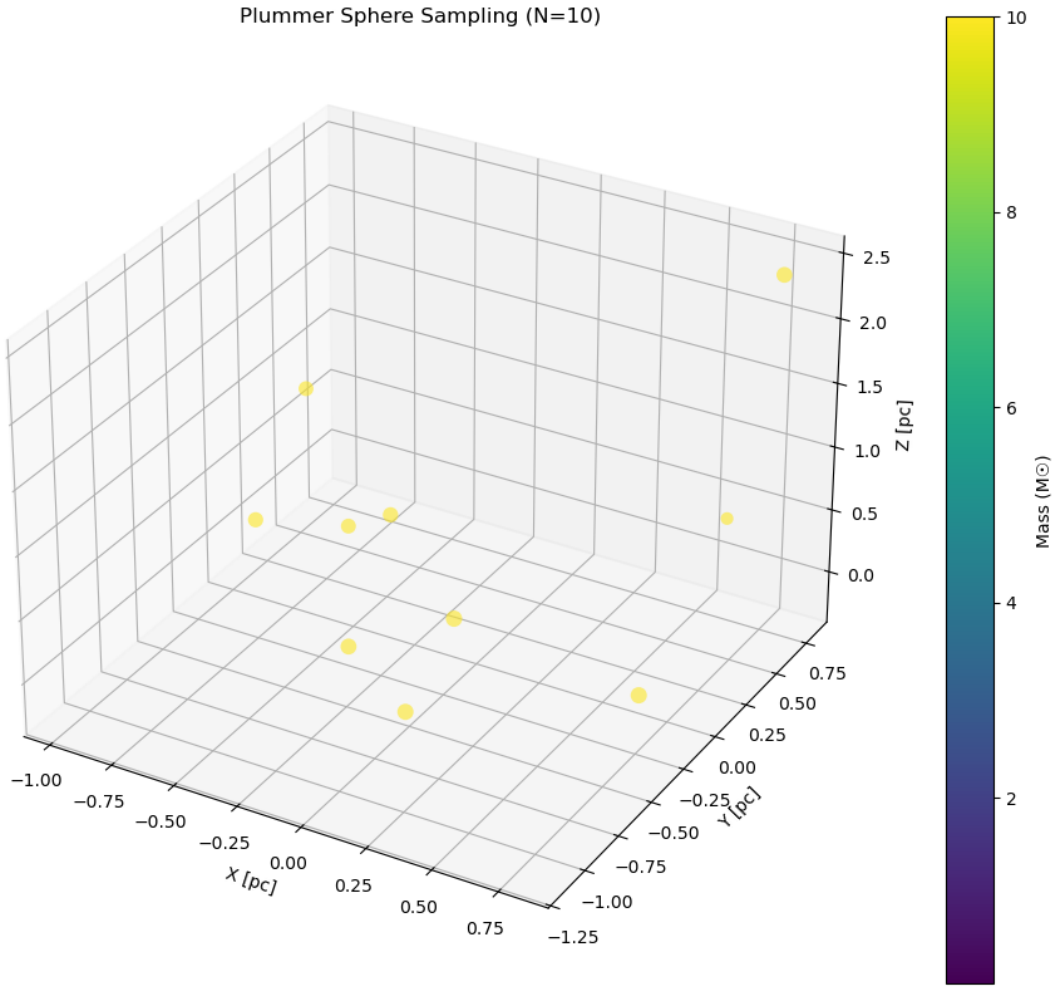
```
[[ 0.8060969   0.72684122  2.43550309]
 [ 0.73362265 -0.19628958 -0.00767862]
 [-0.36211434 -0.28204937 -0.21292392]
 [-0.90900473  0.01635283  0.20113729]
 [ 0.05972873 -0.63747294 -0.11568821]
 [-0.60251114  0.46832222 -0.00365057]
 [-0.13868251  0.0907303  -0.2173162 ]
 [ 0.11651435 -1.12688982  1.82291045]
 [ 0.59375226  0.78760038  0.36215609]
 [-0.97537863  0.51115915  0.77925218]]
```

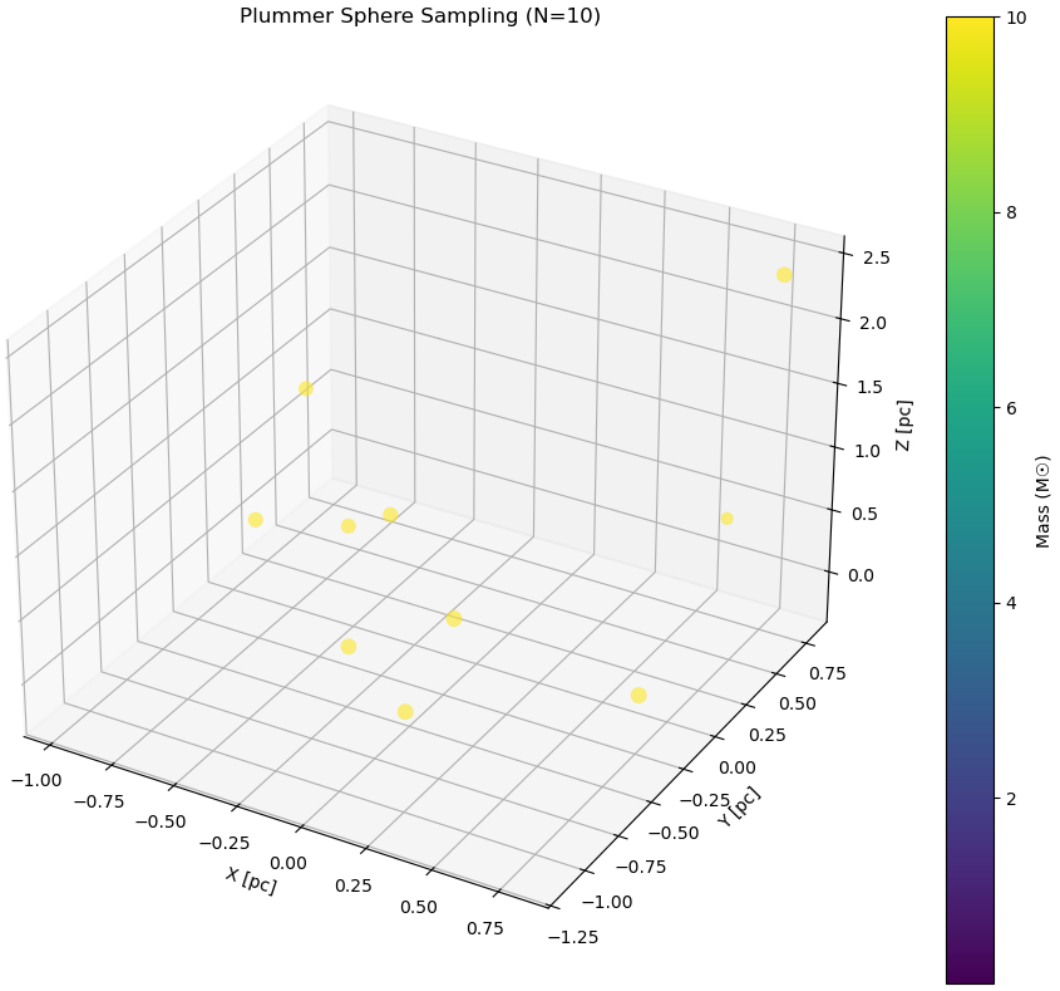

Plummer Sphere Sampling (N=10)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)

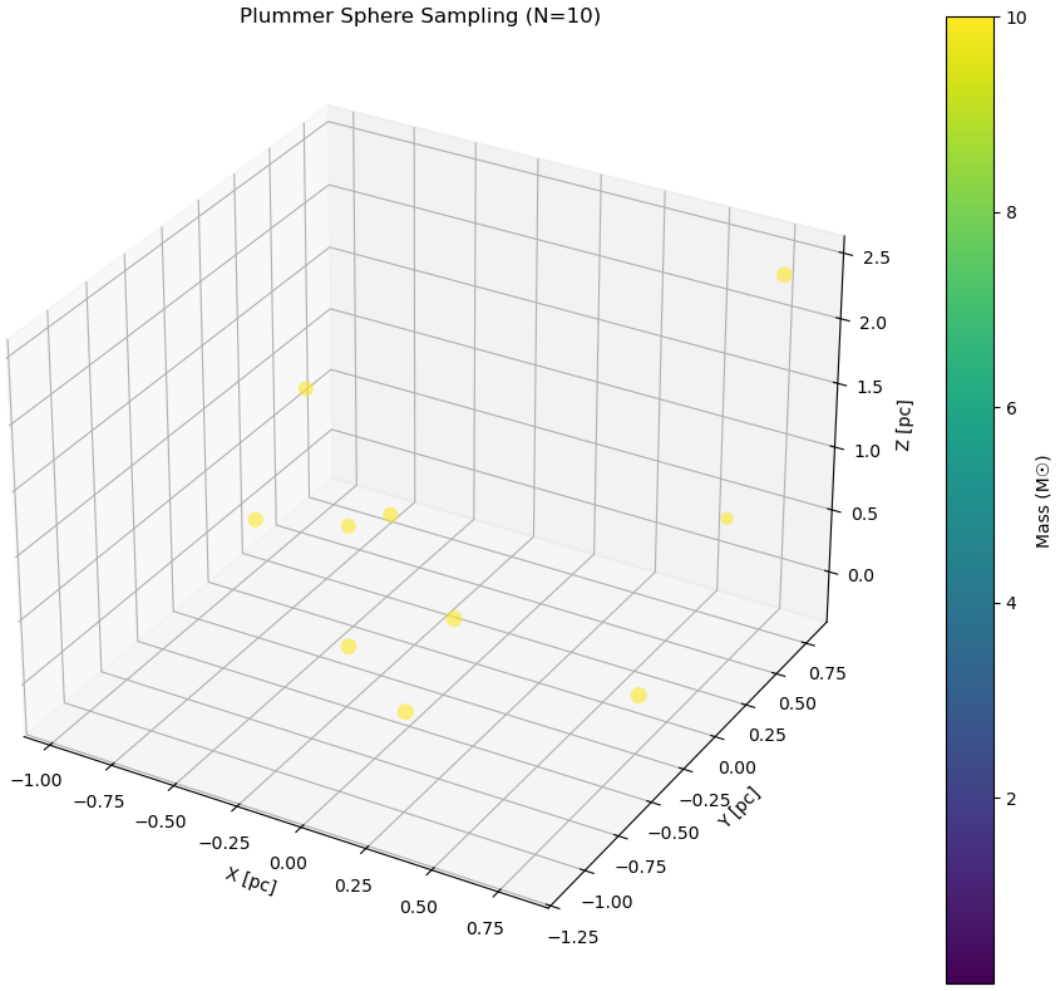

Plummer Sphere Sampling (N=10)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)

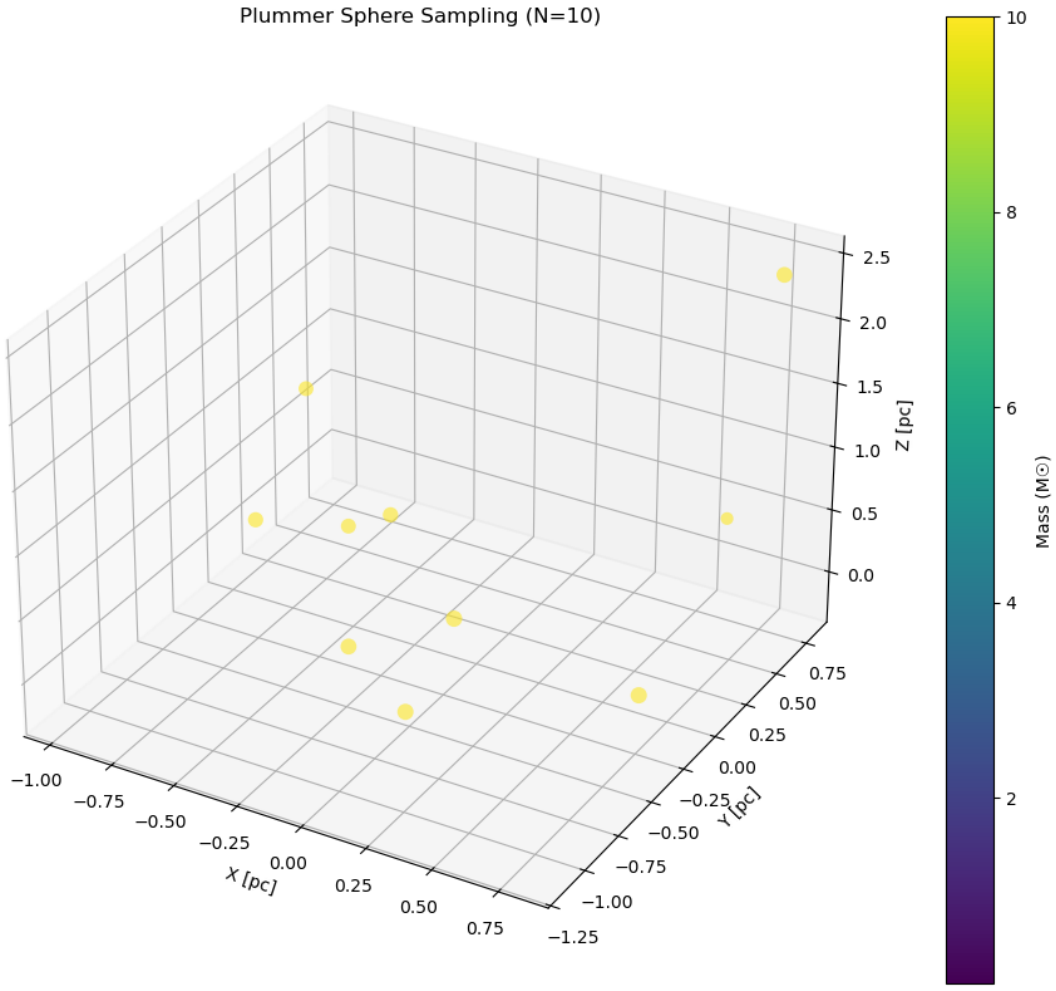

Plummer Sphere Sampling (N=10)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)


Plummer Sphere Sampling (N=10)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)


Plummer Sphere Sampling (N=10)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)

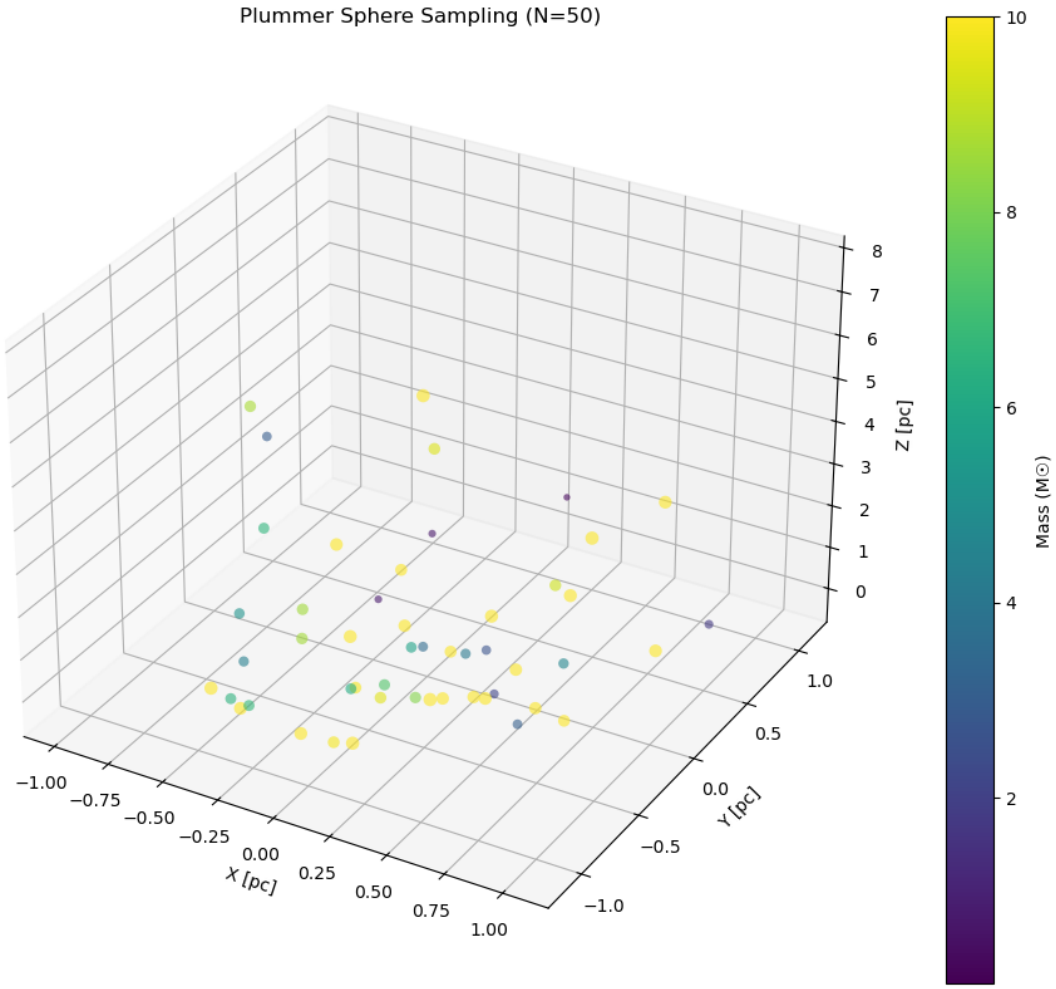

Plummer Sphere Sampling (N=10)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)

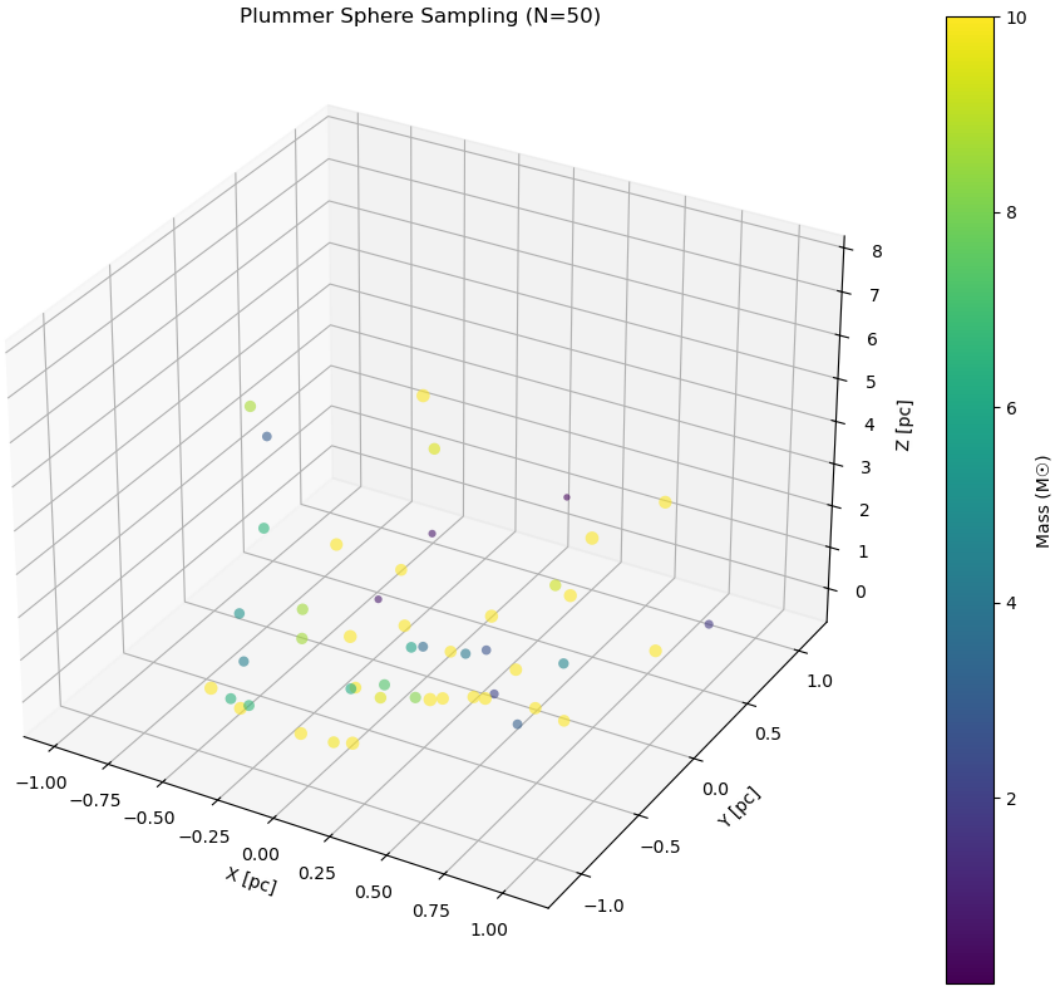

Plummer Sphere Sampling (N=10)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)

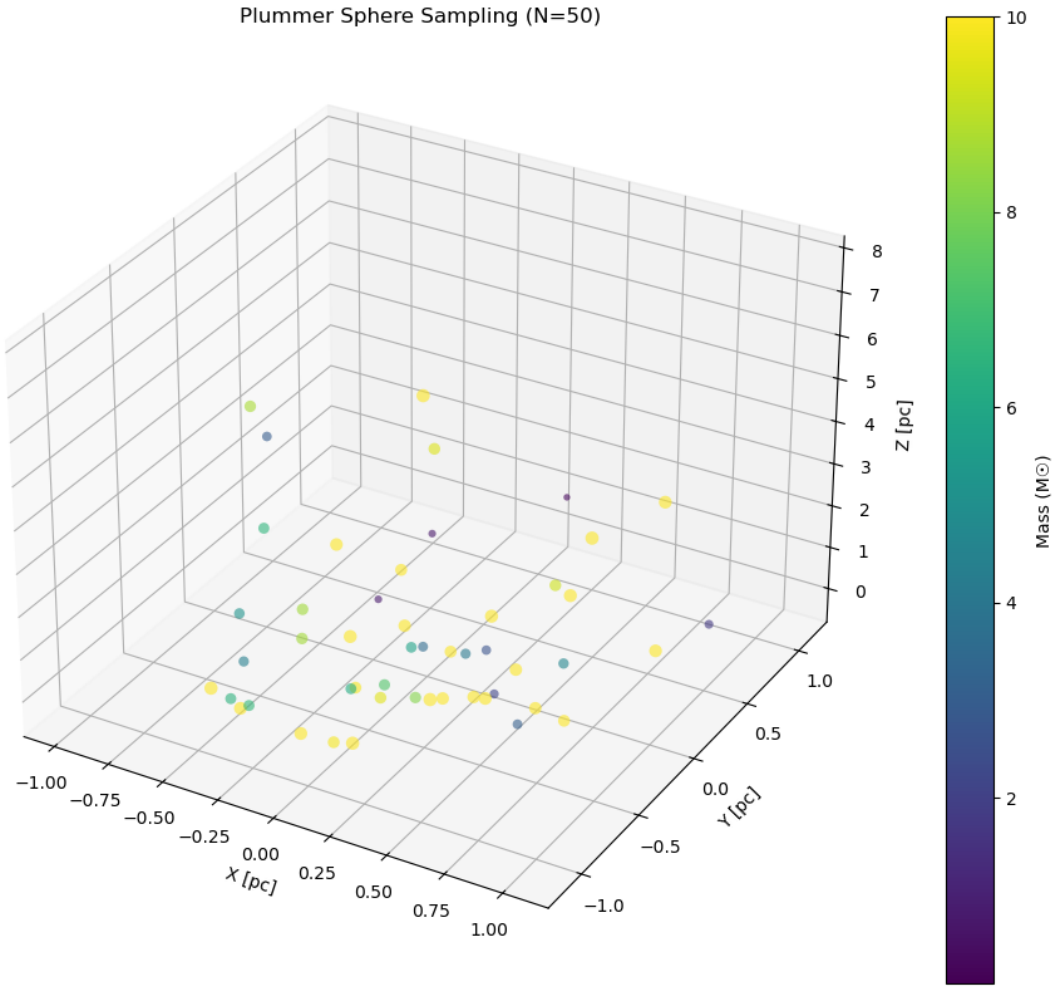


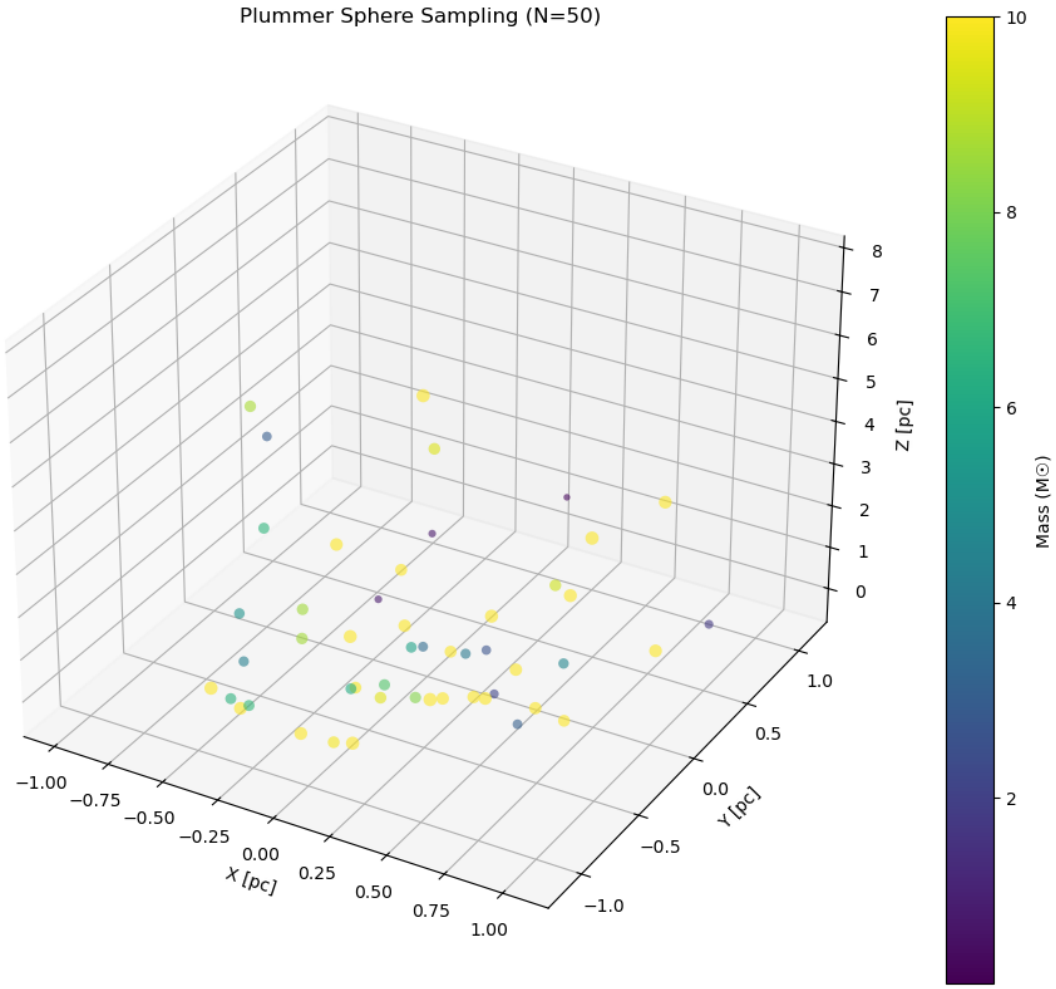

```
1.55 s ± 198 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[[-3.70299926e-01  2.69717713e-01 -2.13224735e-01]
 [-3.26311214e-01 -5.36115725e-02 -2.39675822e-01]
 [ 3.14668214e-01  6.66353058e-01 -3.13003906e-02]
 [-1.79044243e-01  1.41745589e-01 -2.34657368e-01]
 [-3.87913160e-01 -9.26213878e-01  4.07977516e-01]
 [-1.11033926e-01 -4.05370299e-01 -2.24610972e-01]
 [-9.93212681e-01  4.56209447e-01  2.35126091e+00]
 [ 6.00874865e-01  9.48410122e-01  1.98193814e+00]
 [ 3.82540935e-01 -1.50712529e-01 -2.26920814e-01]
 [-5.84724776e-01  1.19113500e-02 -1.53193578e-01]
 [ 5.89934973e-01 -3.53700023e-01 -7.73861862e-02]
 [ 3.67710697e-01  6.93161716e-02 -2.34487742e-01]
 [-5.55209620e-01 -5.29924613e-01  1.21554996e-03]
 [ 1.48080136e-01 -1.05740018e+00  6.16679887e-01]
 [ 8.44137187e-01  3.68810161e-01  2.23058666e-01]
 [ 1.04560743e+00  4.32383082e-01  1.01849178e+00]
```

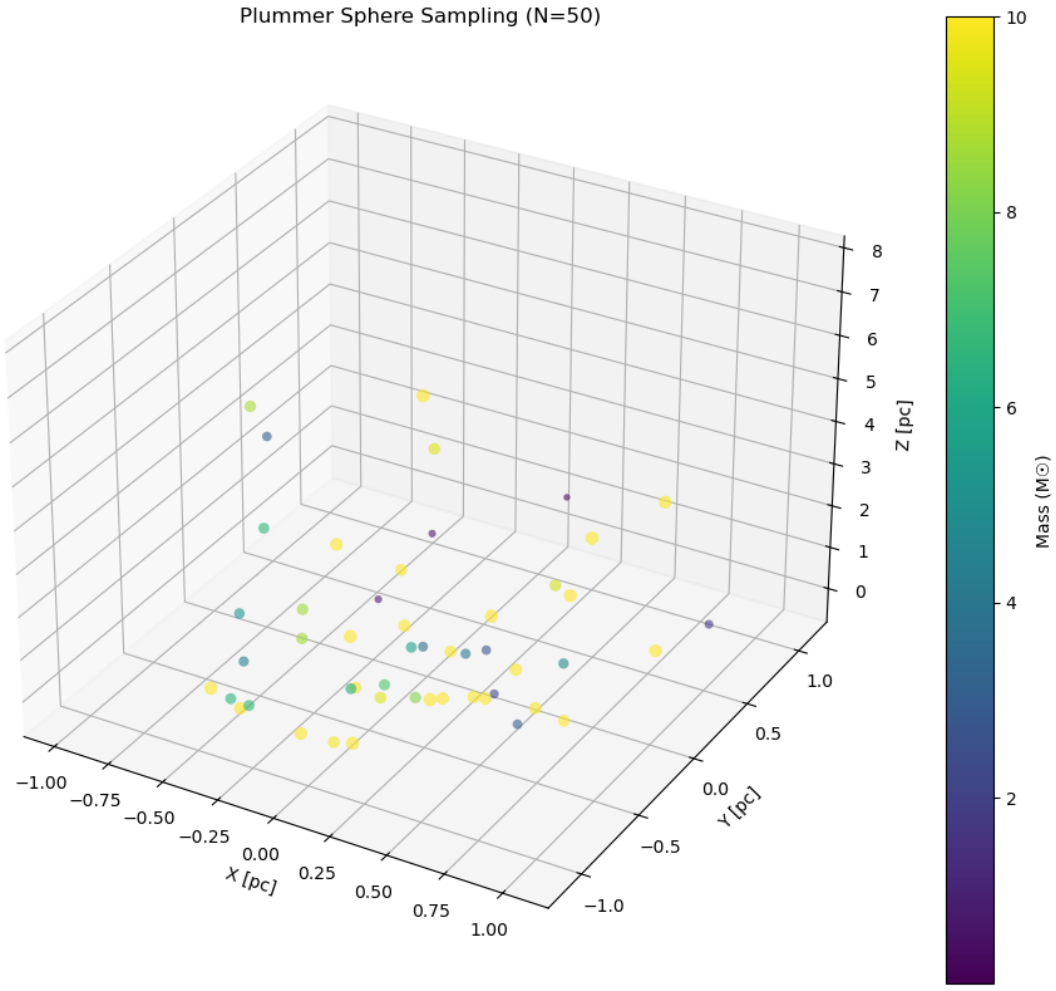
```
[ 2.24641141e-01  7.08453004e-01 -2.48545922e-02]
[ 3.23078106e-01 -2.14379760e-01 -2.32046909e-01]
[-1.20319489e-02 -1.32266668e-02 -8.39885510e-02]
[-1.21417020e-01 -4.24677765e-01 -2.18531840e-01]
[-4.77300962e-01 -1.87139254e-01 -1.91576468e-01]
[-6.86870311e-01  4.89871437e-01  9.70639540e-02]
[ 7.49551085e-01 -2.63241135e-01  3.24938200e-02]
[ 8.66184190e-02  3.92666248e-02 -1.78847829e-01]
[-9.40471182e-01  2.31397317e-01  3.66521645e+00]
[ 2.19613647e-03 -1.05777355e+00  5.77648599e-01]
[ 2.22868769e-01 -2.89758777e-01 -2.35840674e-01]
[ 1.57166732e-01 -9.21768015e-01  2.49627277e-01]
[ 1.40371692e-01  6.73238382e-02 -2.13267653e-01]
[ 3.68554826e-01 -2.01669874e-01 -2.24657785e-01]
[ 7.10285891e-02  1.11699505e+00  9.02325637e-01]
[-6.07130963e-02 -2.29835809e-02 -1.52628195e-01]
[ 1.94544891e-01  1.45286506e-01 -2.36970250e-01]
[-5.07881190e-01  1.02763610e+00  1.43911007e+00]
[ 5.85545225e-01 -1.88847629e-01 -1.33526666e-01]
[ 3.18635451e-01 -6.34731158e-01  7.68528888e+00]
[-1.93418432e-01 -1.11884685e+00  1.06794587e+00]
[-3.86139272e-01  7.76755895e-01  1.32166885e-01]
[-7.57812552e-01 -2.08488963e-01  2.23748732e-02]
[-3.90740995e-01  5.08821255e-01 -1.14735587e-01]
[ 2.50169143e-01  9.94321034e-01  4.68253555e-01]
[ 5.31749915e-01 -9.83416754e-01  2.04772388e+00]
[ 4.80635074e-01 -1.00731999e+00  2.07247848e+00]
[ 8.74595907e-02  3.99894898e-01 -2.27358044e-01]
[-2.86462950e-01 -1.02563411e+00  6.05241990e-01]
[ 5.18058167e-01  1.97395868e-01 -1.70781945e-01]
[-9.53365081e-01  3.46467121e-01  4.36175347e-01]
[-4.72037200e-01 -9.35384218e-01  5.41198071e-01]
[ 3.58427439e-01 -1.06193786e+00  2.00922154e+00]
[-2.54616666e-02 -3.19024528e-01 -2.40315092e-01]]
```

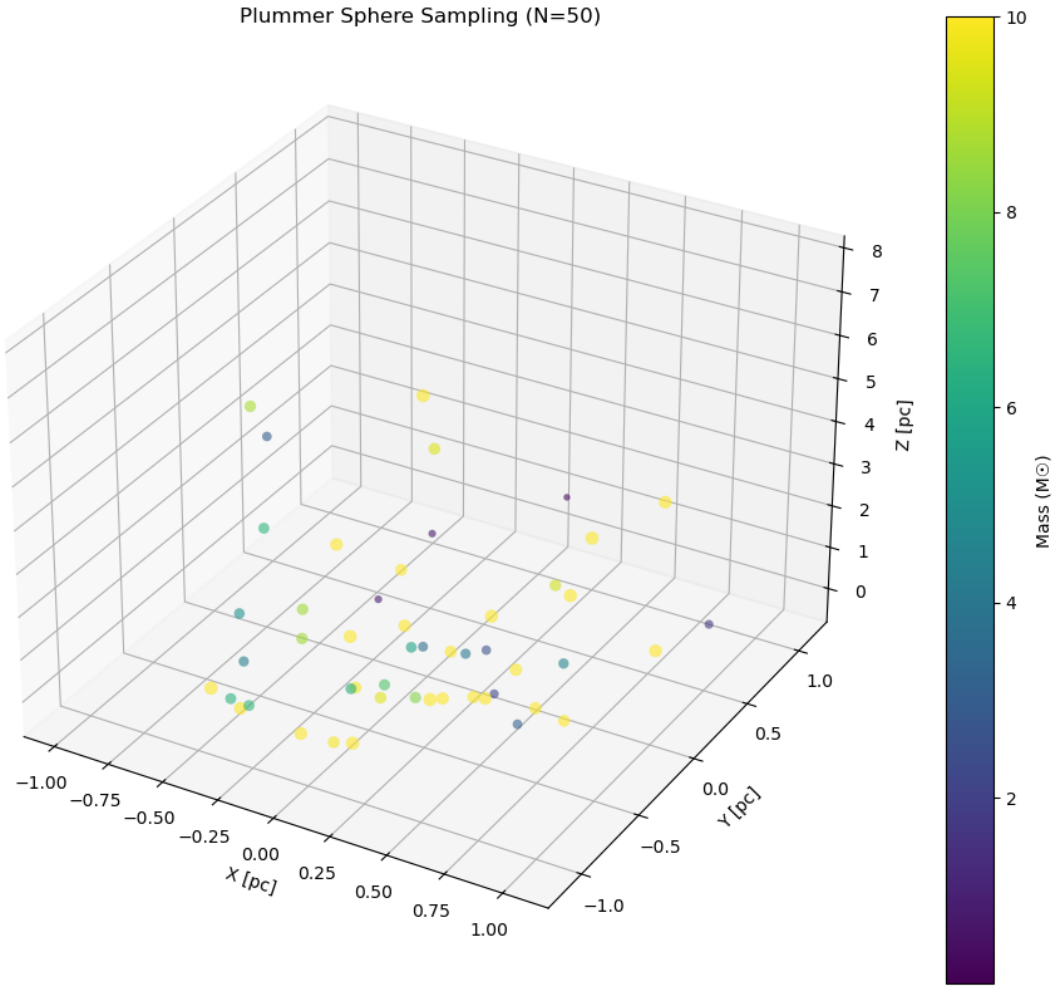

Plummer Sphere Sampling (N=50)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)

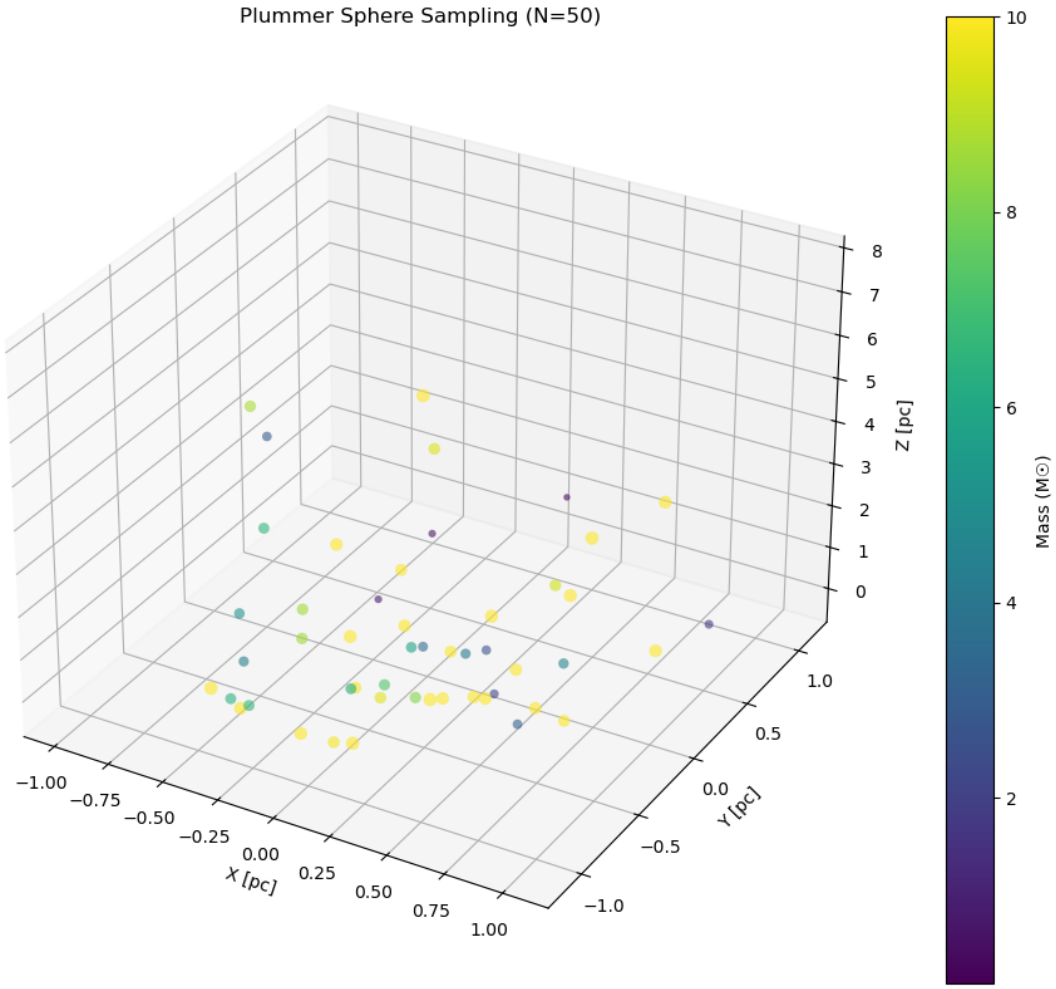

Plummer Sphere Sampling (N=50)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)

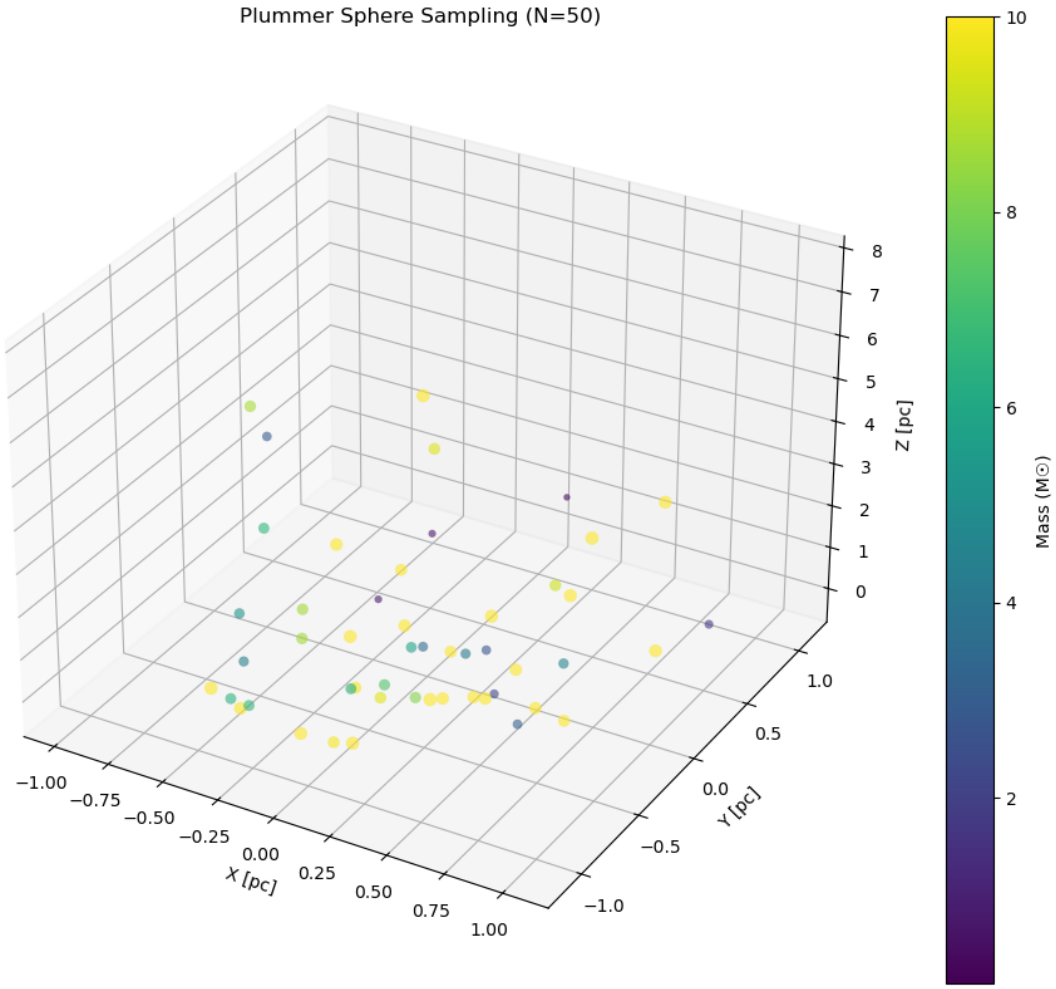

Plummer Sphere Sampling (N=50)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)


Plummer Sphere Sampling (N=50)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)


Plummer Sphere Sampling (N=50)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)


Plummer Sphere Sampling (N=50)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)

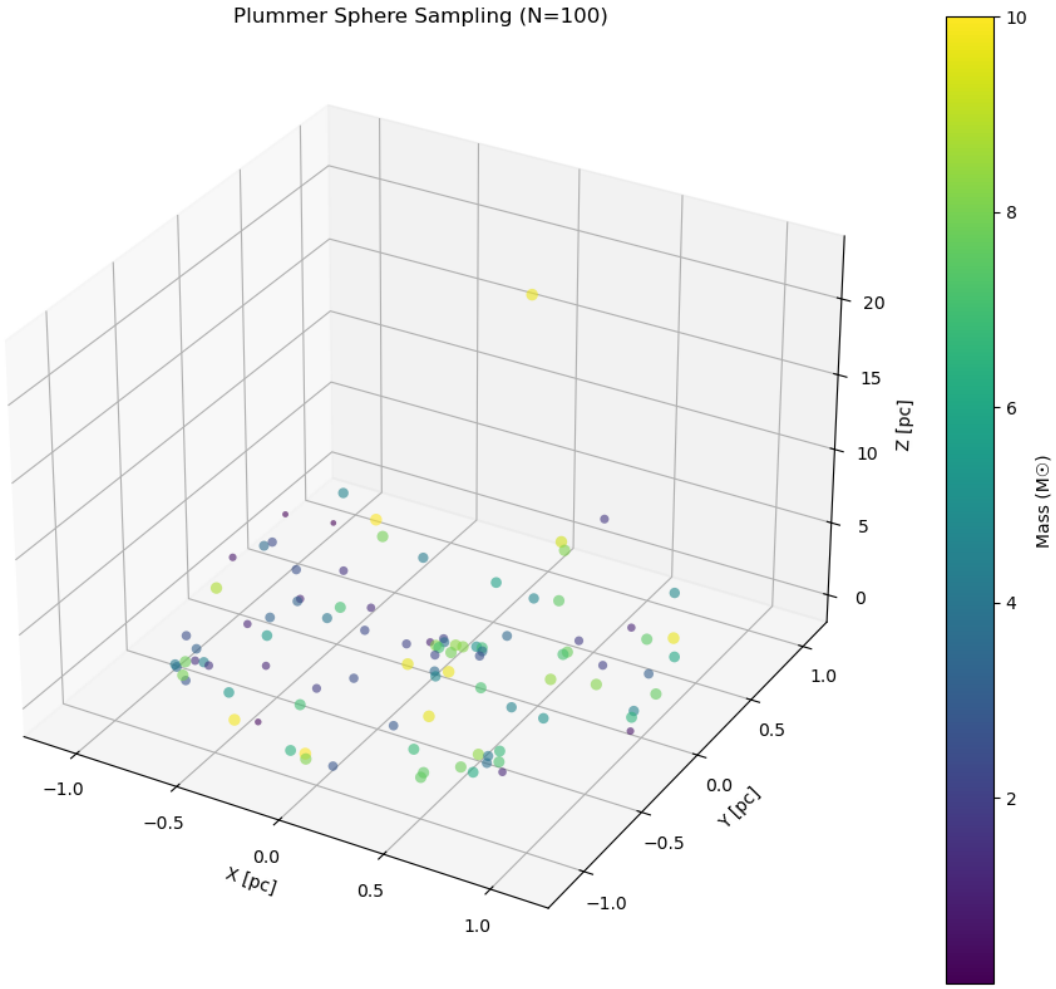

Plummer Sphere Sampling (N=50)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)

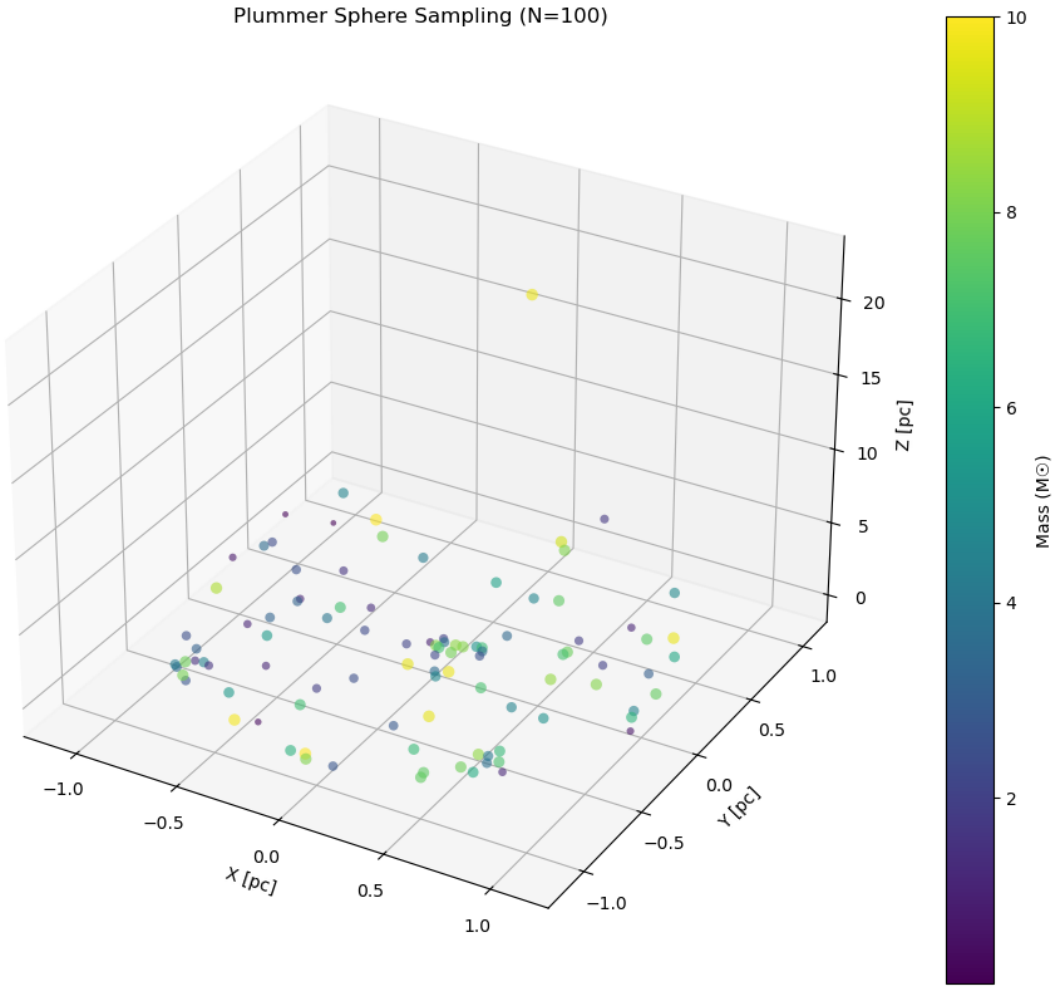


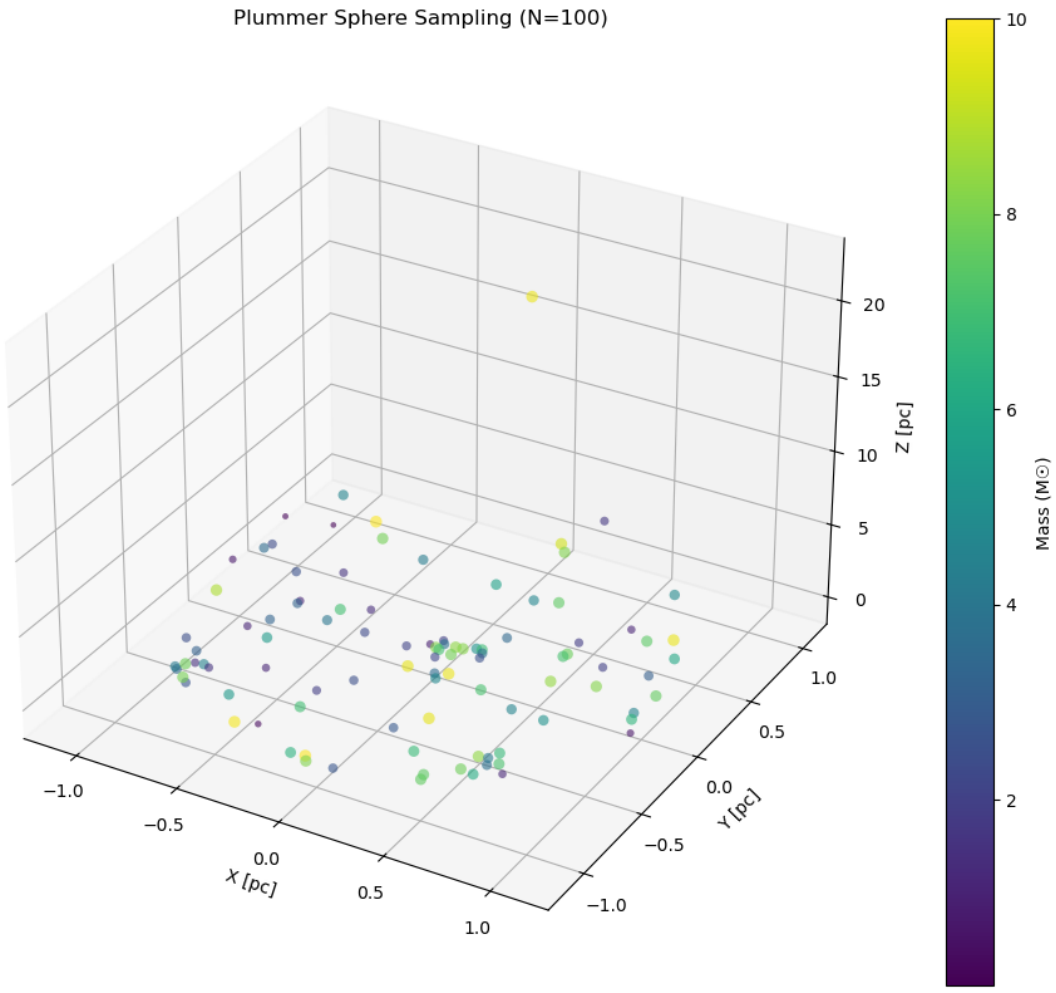

```
1.55 s ± 108 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[[-8.24012036e-02 -1.06798421e+00  6.30959788e-01]
 [-6.74741753e-01  3.52029500e-01 -5.90996033e-03]
 [ 4.96806110e-01  2.66771472e-01 -1.65512423e-01]
 [-2.15252011e-02 -1.05087931e+00  5.53097994e-01]
 [ 1.33723939e-01  1.14770429e-01 -2.21137616e-01]
 [ 5.79903042e-01 -6.56743301e-01  1.45710008e-01]
 [-8.19825348e-01 -4.60369770e-01  3.98261998e+00]
 [-4.97762909e-02  6.14614114e-01 -1.32581430e-01]
 [ 1.57999679e-01  5.80739200e-01 -1.42459335e-01]
 [-8.49637868e-01  2.38676222e-01  1.55947282e-01]
 [ 6.91607211e-01 -6.81019095e-01  3.24931302e-01]
 [-5.58031398e-01 -4.90841012e-01 -2.48861134e-02]
 [ 2.36633476e-01 -4.73083040e-01 -1.83902209e-01]
 [ 6.18053710e-02  1.00600611e+00  4.18125475e-01]
 [ 5.90657649e-01 -6.58139201e-01  4.66802609e+00]
 [-4.09837310e-01 -9.81900748e-01  6.01708933e-01]
```

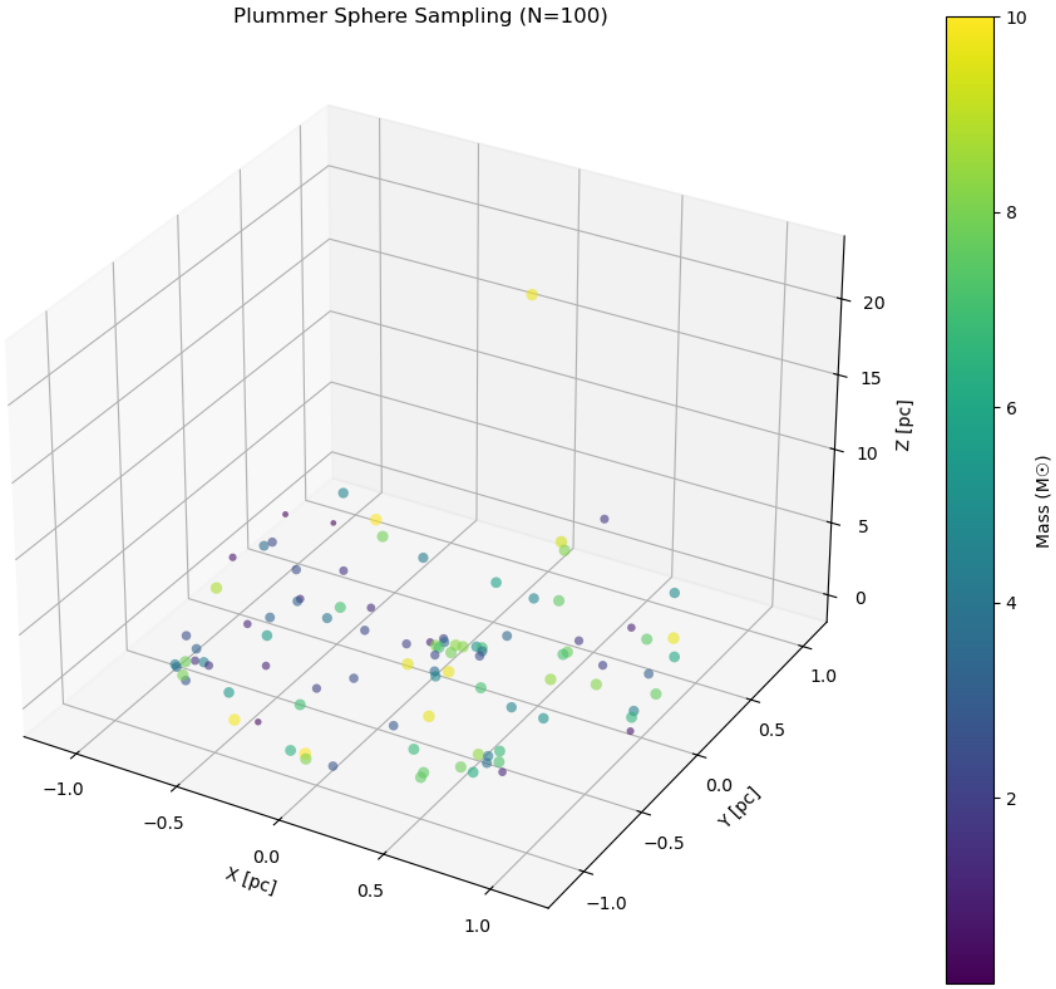
```
[ 5.08822418e-01 -1.01137077e+00  1.02629738e+00]
[-9.57138033e-04  7.11480114e-02 -1.58747059e-01]
[ 6.60778680e-02  1.19312749e-02 -1.54863759e-01]
[-7.98994377e-01 -2.51650408e-01  8.86709458e-02]
[-1.10346810e+00  1.18144533e-01  8.32943297e-01]
[ 2.40116281e-01  1.07421313e+00  2.26235447e+00]
[-1.93698635e-01 -3.68878213e-01 -2.25556439e-01]
[ 6.97668817e-01 -9.09651636e-01  1.42419974e+00]
[-7.41011715e-01  7.46973463e-01  5.56962264e-01]
[-7.18761499e-01  5.55345957e-02 -4.71209481e-02]
[ 4.97630994e-01  3.69035806e-01 -1.30575875e-01]
[ 5.55502414e-03  1.24311732e-02 -7.35943178e-02]
[-8.08833449e-01 -8.02666544e-01  1.13090185e+00]
[-8.07047697e-01 -6.61884928e-01  5.27420688e-01]
[-4.26711969e-01  6.25082287e-01 -1.04762775e-02]
[ 2.65078620e-01  6.13438445e-01 -9.38700776e-02]
[ 1.03777770e-01  9.59705893e-01  3.12917276e-01]
[ 6.86366667e-01  2.35365104e-01 -4.25614284e-02]
[-1.37812277e-01 -1.78896583e-02 -2.05608384e-01]
[-5.97941677e-01 -8.77751155e-01  2.68448953e+00]
[-6.83425848e-01 -2.78460819e-01 -3.02211426e-02]
[ 6.23356210e-01 -6.51314849e-01  1.87781777e-01]
[ 5.39896714e-01  3.86942581e-02 -1.77721413e-01]
[ 1.47794087e-01 -6.16877397e-01 -1.20054827e-01]
[ 3.22686101e-01 -7.48253651e-01  5.80693431e-02]
[ 1.82363210e-01  5.64925326e-02 -2.25828374e-01]
[ 1.07701743e+00 -2.49779454e-01  8.05925639e-01]
[-7.41281635e-01 -8.38062530e-01  2.03577832e+00]
[-1.04082275e+00  4.62694762e-01  1.70281234e+00]
[ 1.01283580e+00 -1.17562455e-01  4.51365794e-01]
[ 5.75389367e-01 -7.96480448e-01  3.52998677e-01]
[-1.04570606e+00  2.86929807e-01  6.90190502e-01]
[ 5.04036704e-01 -2.38618527e-01 -1.68989404e-01]
[ 6.56454653e-01  5.51570233e-01  1.17027539e-01]
[ 1.79631104e-02 -4.26637518e-02 -1.31382140e-01]
[ 1.00152372e-01 -1.80203515e-01 -2.29942132e-01]
[ 4.71967592e-01 -9.25208065e-01  5.10294658e-01]
[-9.40432670e-01 -5.21398056e-01  6.48745403e-01]
[-2.71690524e-01 -6.81470122e-01 -3.46072249e-02]
[ 7.35857878e-01  8.52378515e-02 -2.73627679e-02]
[ 1.30788213e-01 -1.21278666e-01 -2.21868388e-01]
[-6.59371751e-01  6.66504482e-01  2.54525517e-01]
[ 6.68800881e-01 -2.58407641e-01 -5.08733615e-02]
[-5.14665267e-02 -1.56205466e-01 -2.16830046e-01]
[ 1.73556500e-01  9.50438522e-02 -2.27798937e-01]
[ 6.43977120e-01 -5.90056536e-01  1.41455244e-01]
[ 5.27367252e-02  7.59636708e-02 -1.76869143e-01]
[ 7.34911474e-01 -8.61323598e-01  1.83427194e+00]
```

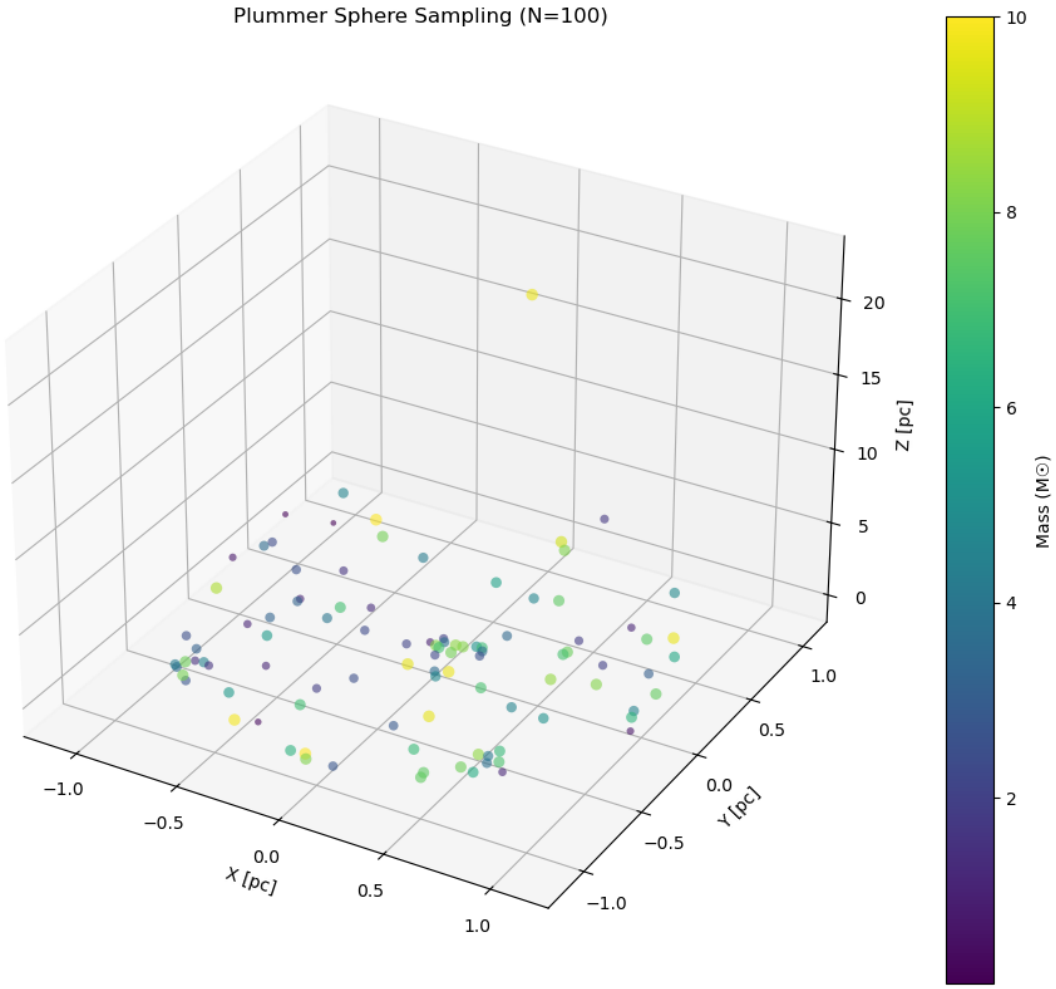
```
[ 1.58134472e-01 -1.12647912e+00  1.09813105e+00]
[-4.23281236e-01  1.64172265e-01 -2.14605673e-01]
[ 9.47407288e-01  3.95174813e-01  4.71932456e-01]
[ 9.93880281e-01 -6.19709364e-02  3.85993729e-01]
[ 1.10583551e+00 -8.10955835e-02  2.16538058e+00]
[-8.56115710e-01 -5.73663827e-01  4.84357413e-01]
[ 7.19805392e-01  8.31520775e-01  7.71060321e-01]
[ 2.03433343e-01  2.59737704e-01 -2.39730078e-01]
[ 1.58544832e-01  1.20049087e-01 -2.28066542e-01]
[ 7.70500393e-02 -1.48728365e-01 -2.17991695e-01]
[ 7.63349815e-01  4.92025873e-01  1.99411537e-01]
[-1.93123271e-02  9.70111563e-02 -1.81630361e-01]
[ 2.75975354e-01  3.18839691e-01  2.24216380e+01]
[-5.42052564e-01 -9.65908780e-03 -1.77279152e-01]
[-5.33965355e-02  4.02702597e-02 -1.54599092e-01]
[ 8.51272087e-02  8.21859006e-02 -1.94355393e-01]
[-6.34370252e-01 -8.59495912e-01  2.61955858e+00]
[-8.28433470e-01  6.11028065e-01  3.02239800e+00]
[-2.84323870e-01 -5.25270243e-01 -1.45406445e-01]
[-3.30955839e-01 -9.21687988e-01  3.45167387e-01]
[ 1.00402587e+00  2.69615998e-01  2.91725801e+00]
[ 3.48315902e-02 -1.13840793e+00  1.12101039e+00]
[ 4.89406086e-01  2.41444341e-01 -1.75412246e-01]
[-5.38445122e-01  9.56359090e-02 -1.74807753e-01]
[-8.84377333e-01  6.15412400e-01  6.58164764e-01]
[ 9.03914660e-01  2.42801168e-01  2.51371236e-01]
[-8.16822434e-01 -8.02374613e-01  1.26820258e+00]
[-3.52555474e-01 -8.74921438e-03 -2.37565293e-01]
[-7.24510170e-01 -8.85910294e-01  1.54938739e+00]
[-7.48987699e-01 -1.47072736e-01 -3.45789207e-03]
[-1.03109985e+00  3.33234536e-01  6.86630841e-01]
[-7.28905393e-01 -8.57970013e-01  9.59379951e-01]
[ 7.62904677e-01 -7.79800434e-01  7.22788526e-01]
[-1.58872319e-02  2.42671011e-02 -1.05812952e-01]
[-7.18975589e-01  3.28166460e-02 -4.82556405e-02]
[-5.61444935e-01 -7.83601890e-01  3.09973681e-01]]
```

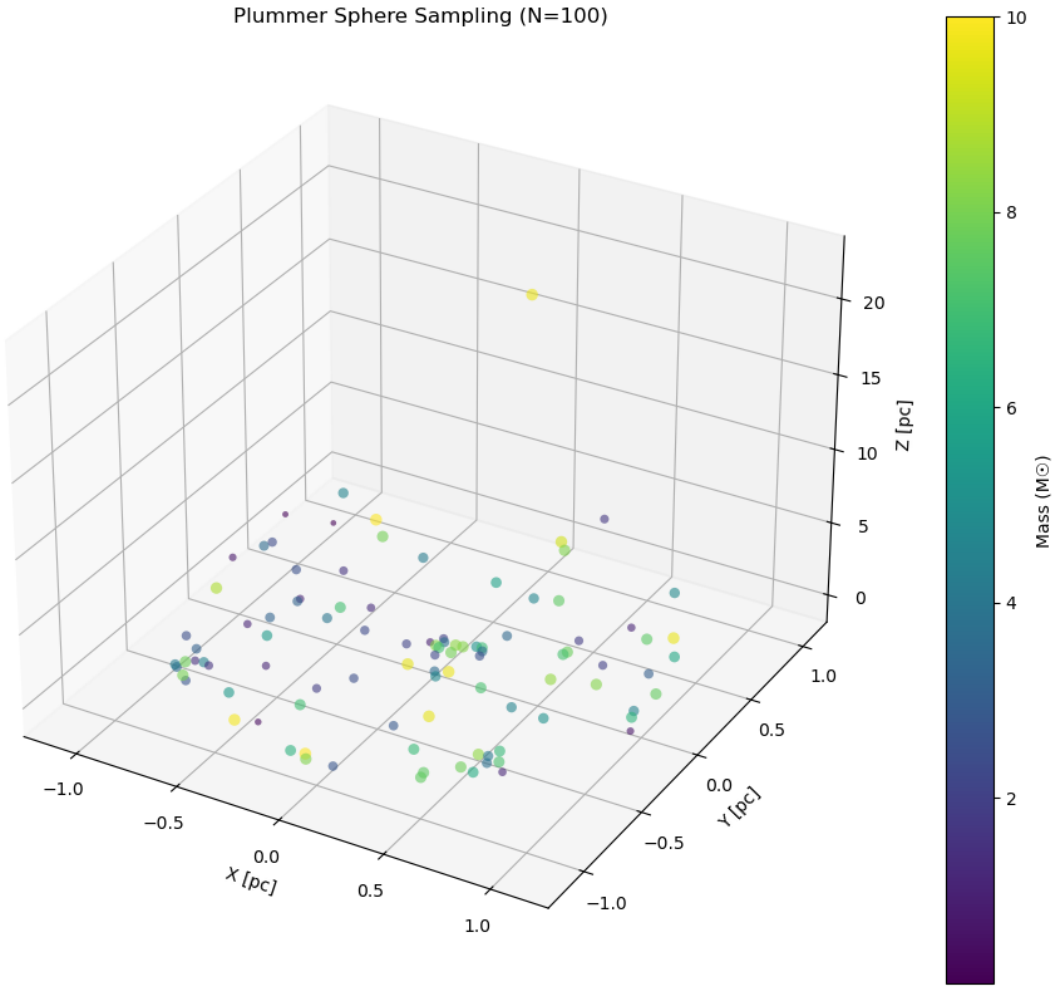

Plummer Sphere Sampling (N=100)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
2
4
6
8
10

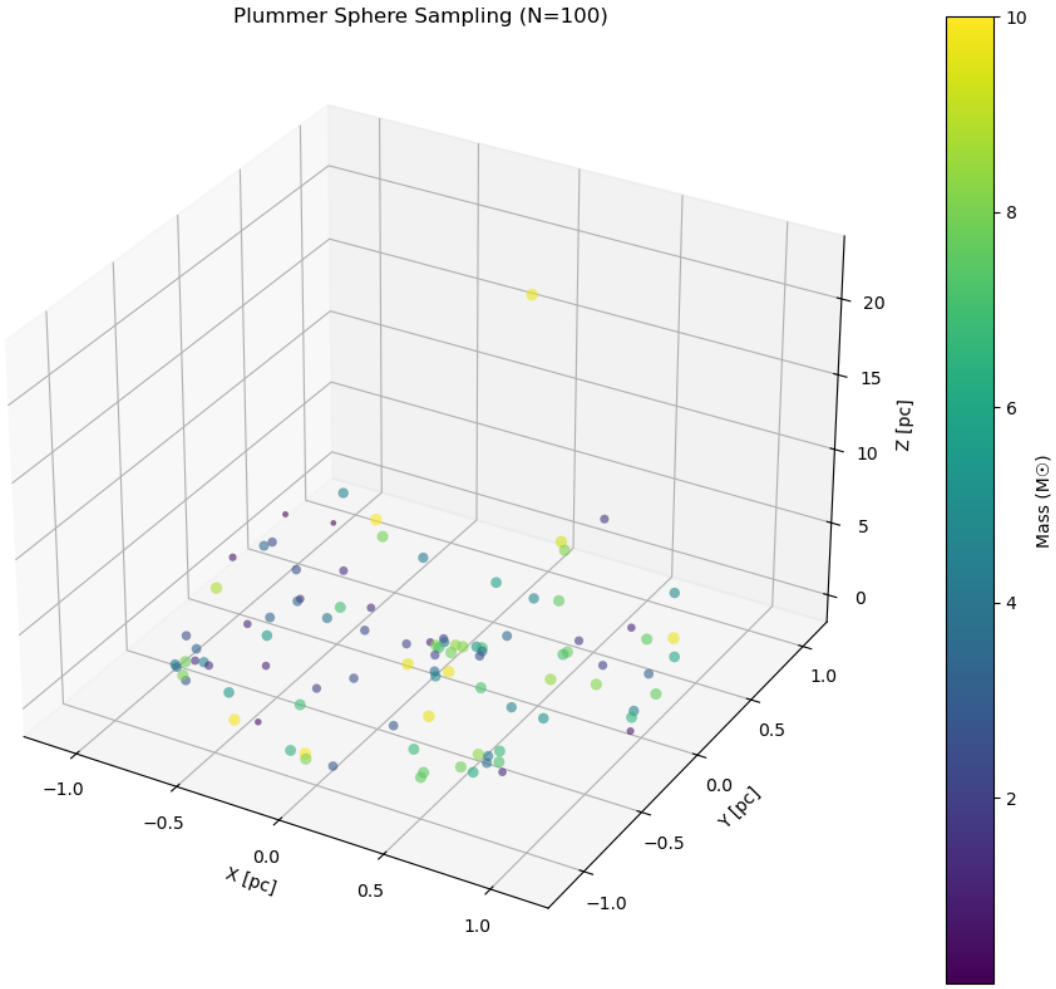

Plummer Sphere Sampling (N=100)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
2
4
6
8
10

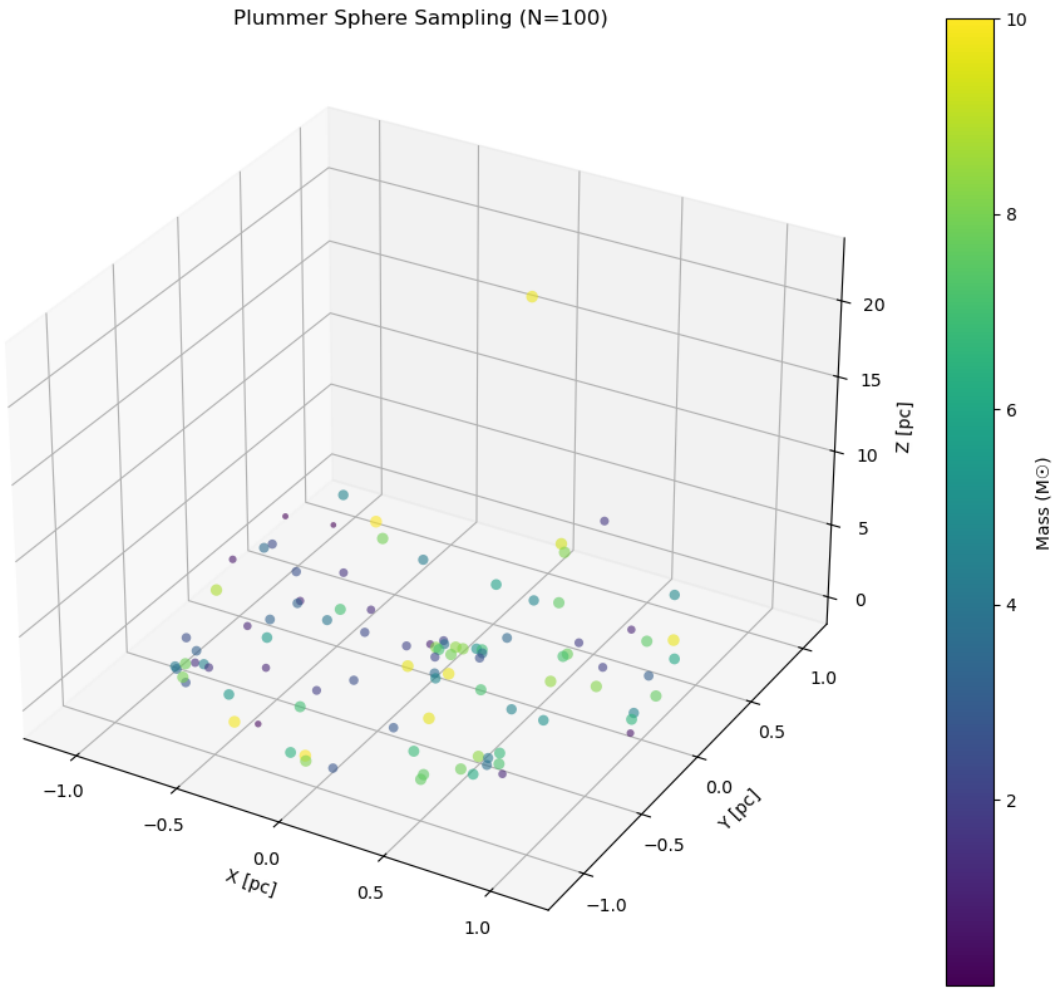

Plummer Sphere Sampling (N=100)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
2
4
6
8
10


Plummer Sphere Sampling (N=100)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
2
4
6
8
10

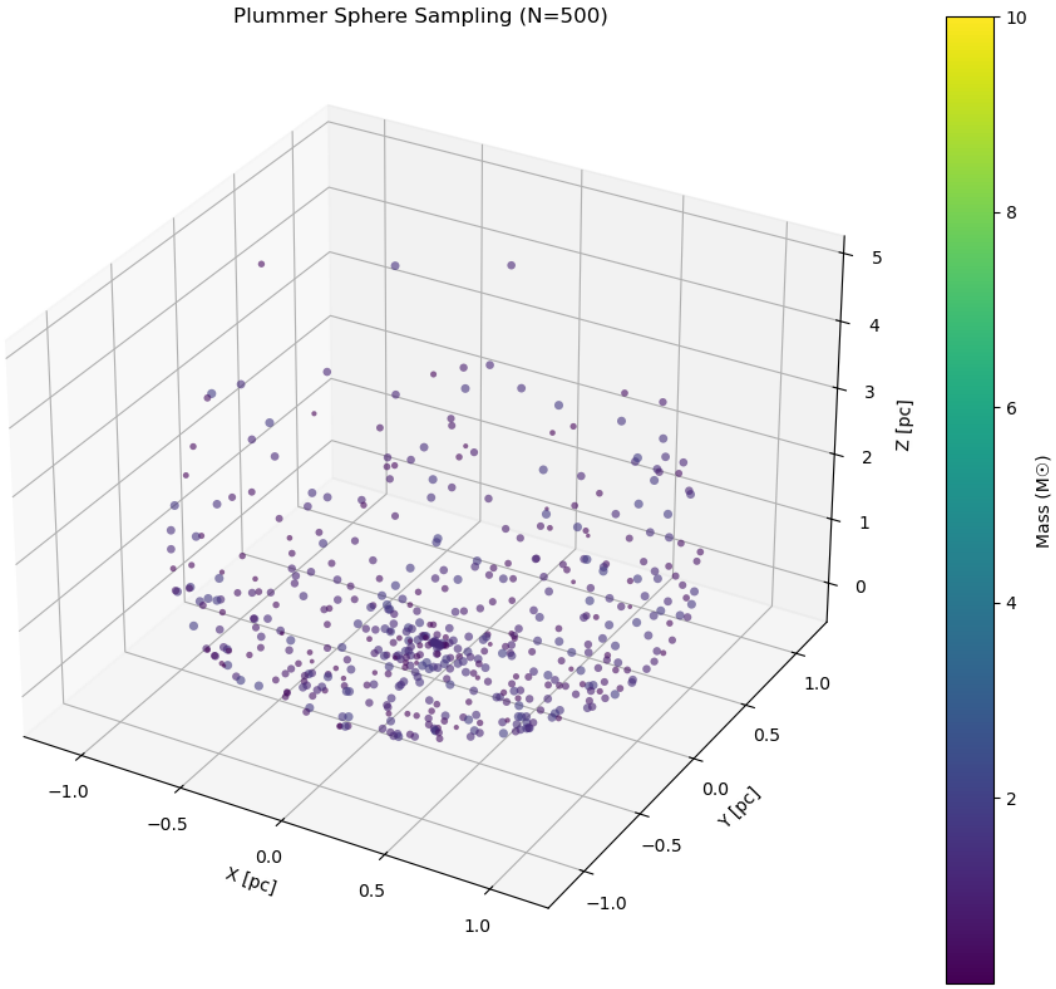

Plummer Sphere Sampling (N=100)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
2
4
6
8
10

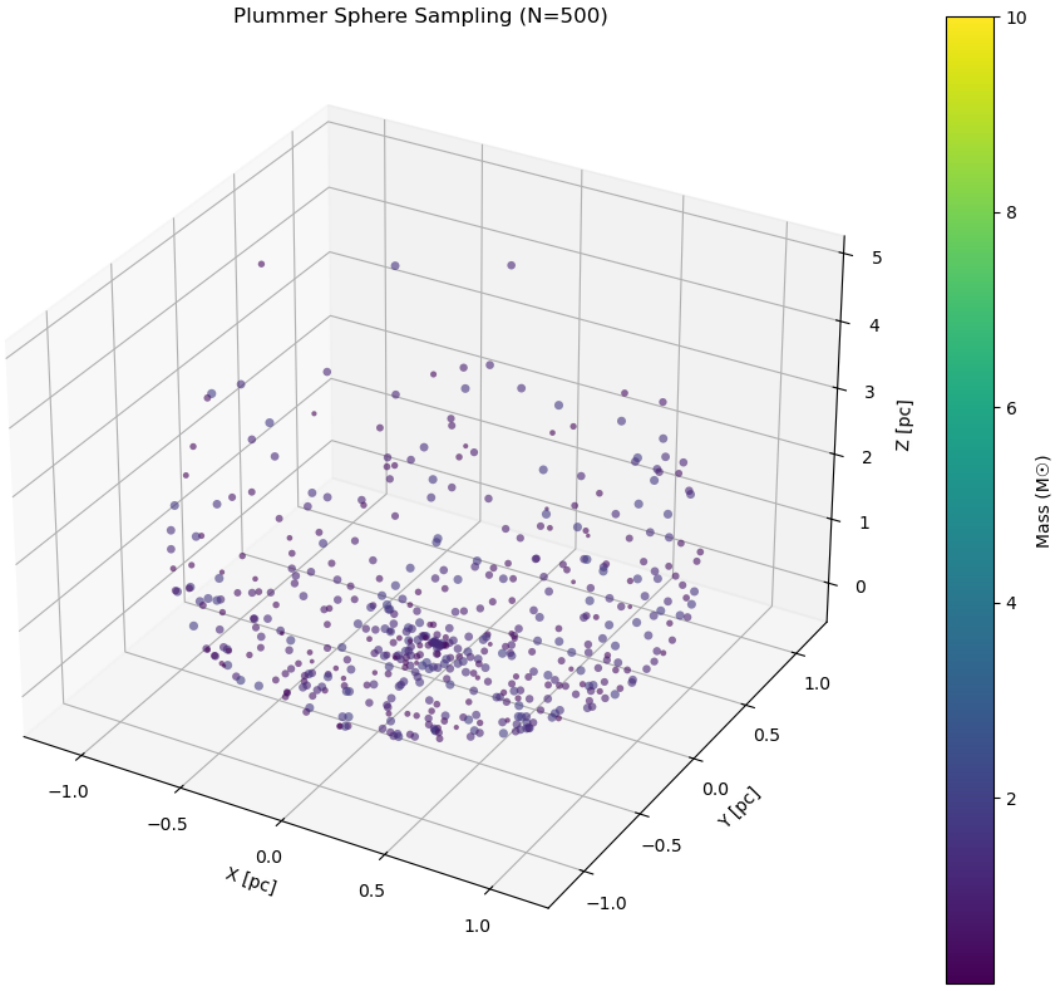

Plummer Sphere Sampling (N=100)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
2
4
6
8
10

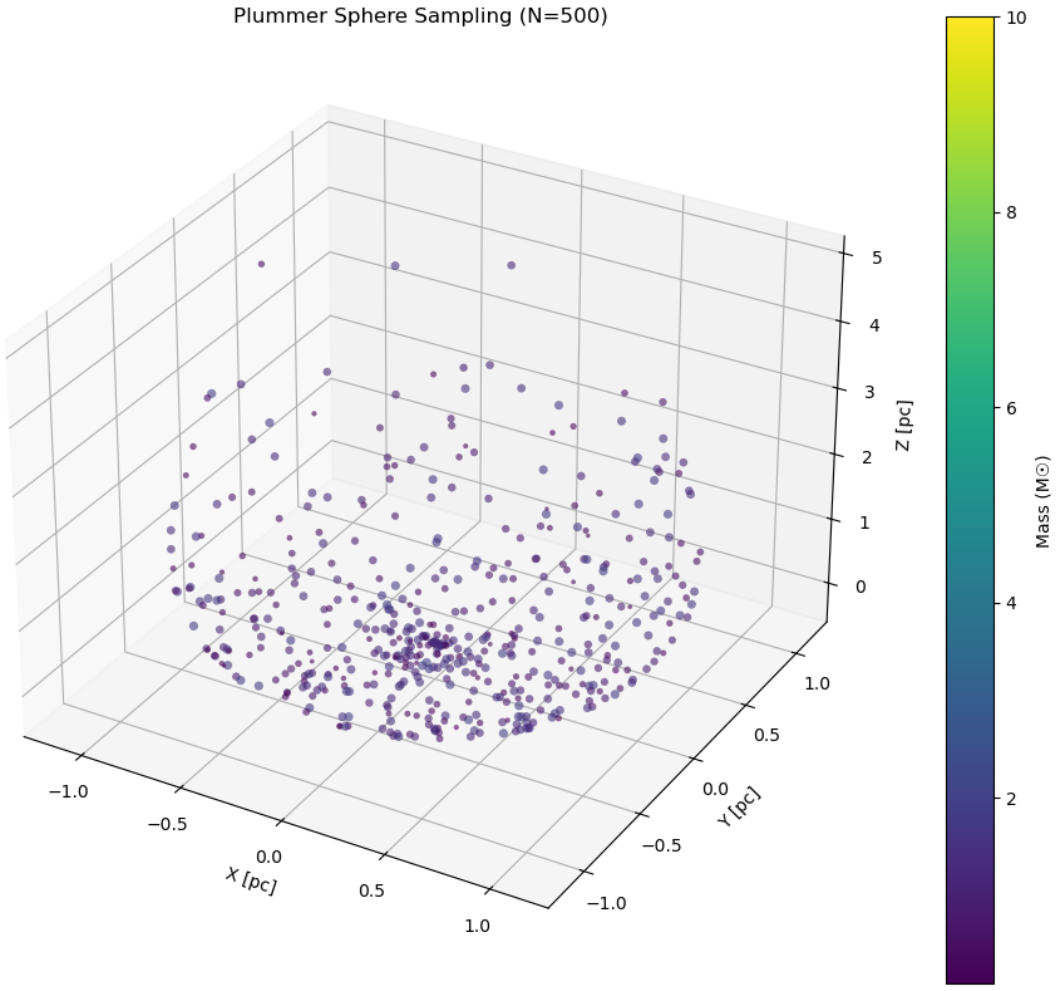

Plummer Sphere Sampling (N=100)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
2
4
6
8
10

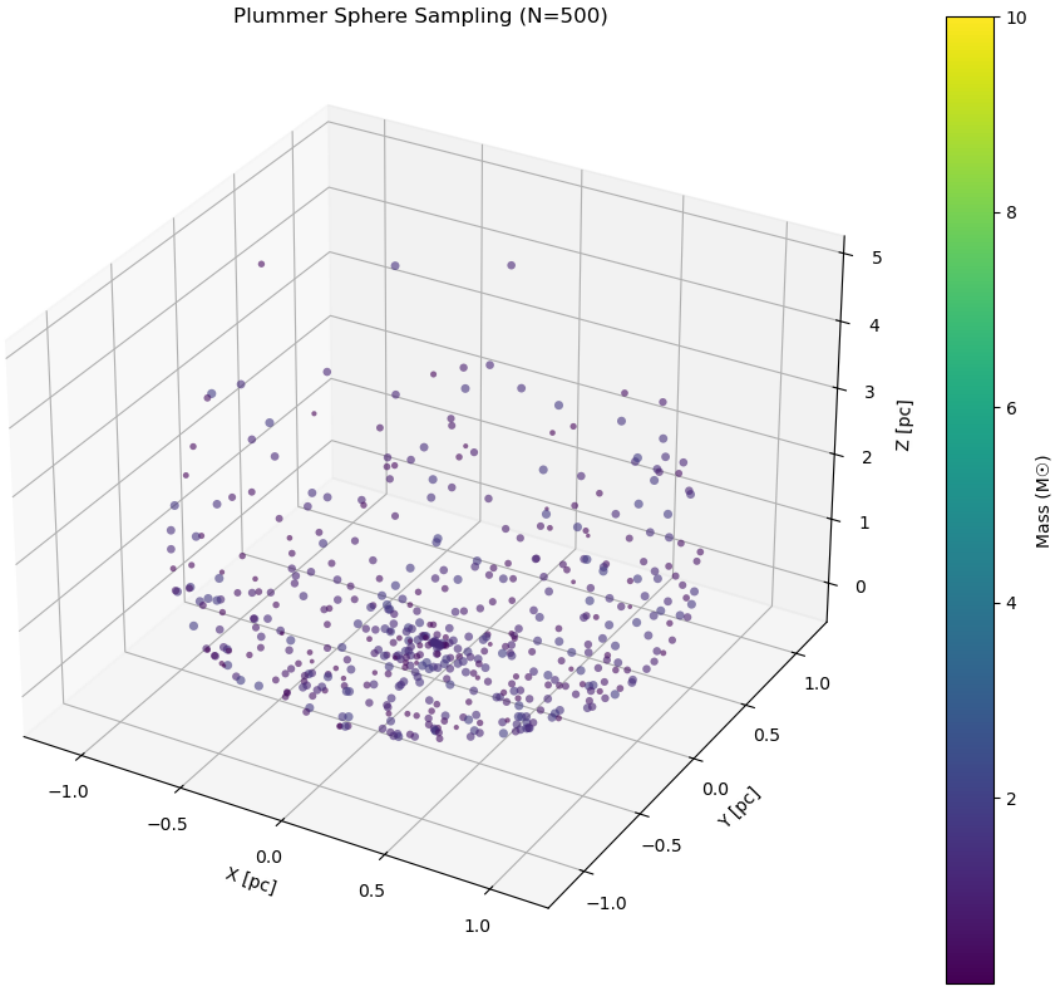


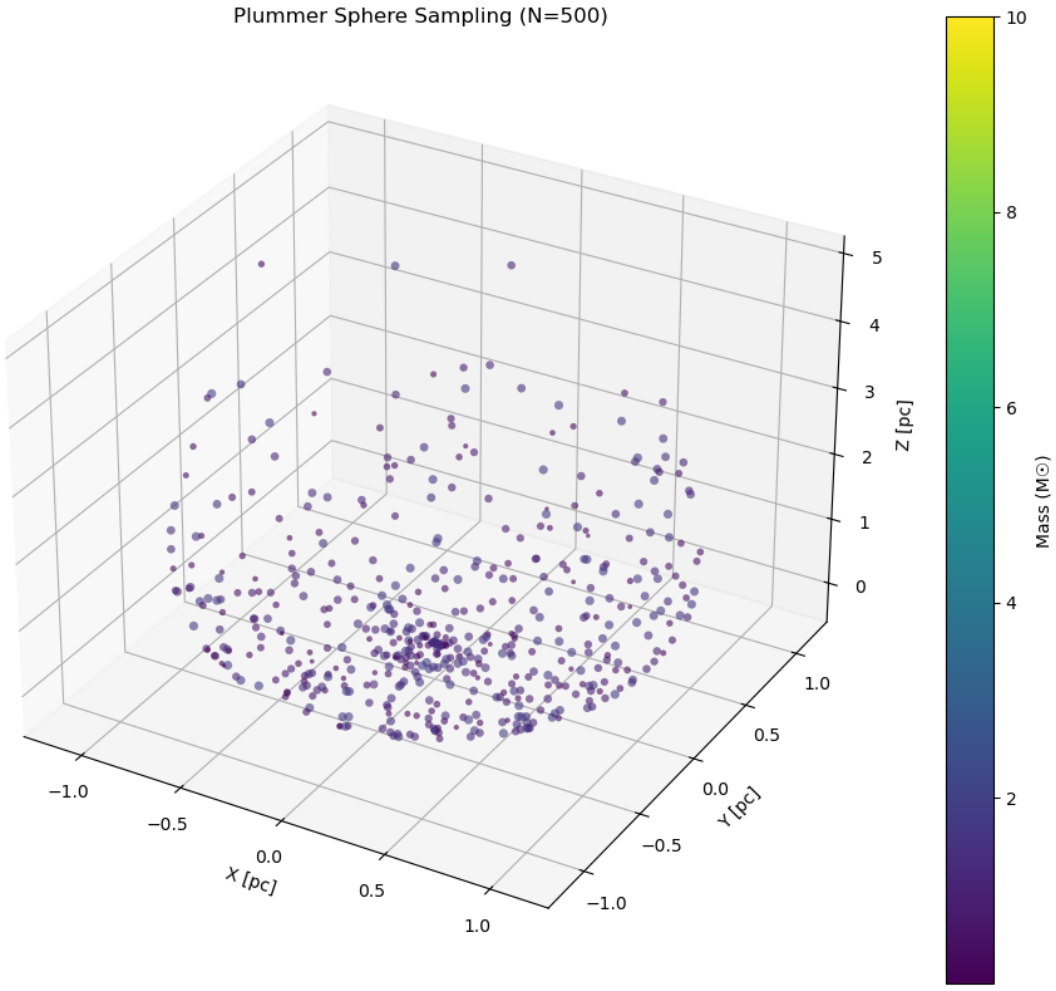

```
1.21 s ± 67.5 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[[ 0.39303572  0.70974042  0.05348446]
 [ 0.10252727 -1.13879842  1.21555116]
 [-0.10020195  0.36210161 -0.23423264]
 …
 [ 0.08572901  0.54627754 -0.17155582]
 [-0.59931562  0.70601495  4.14806288]
 [-0.97094638 -0.27570446  3.23025479]]
```

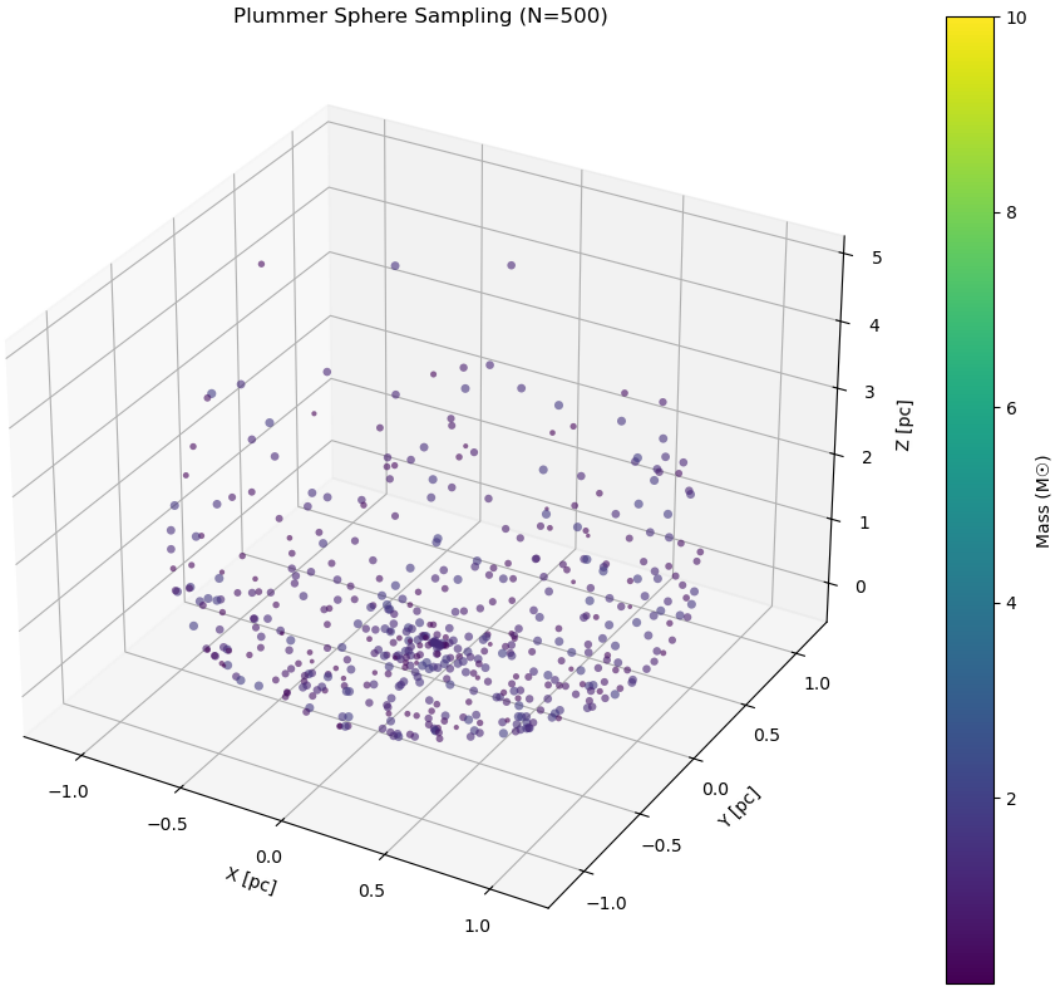

Plummer Sphere Sampling (N=500)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
2
4
6
8
10

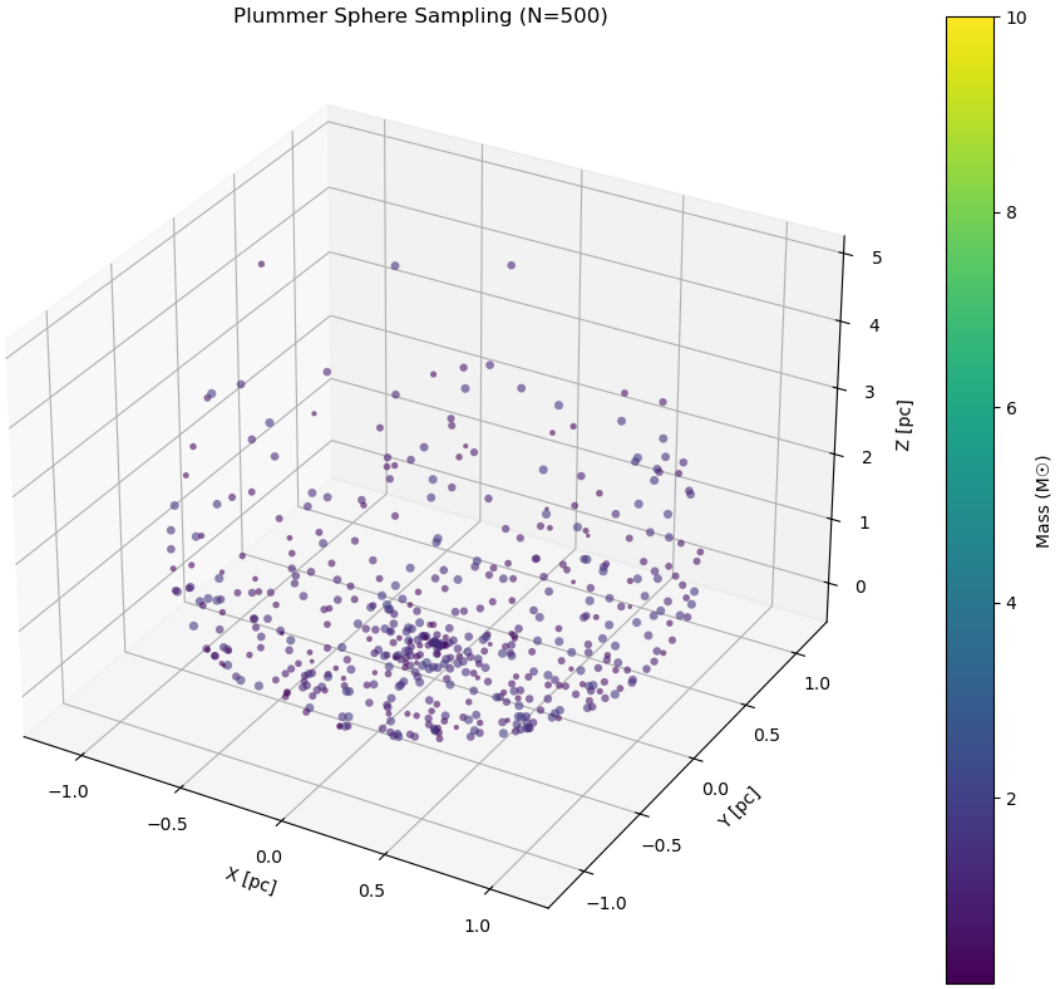

Plummer Sphere Sampling (N=500)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
2
4
6
8
10

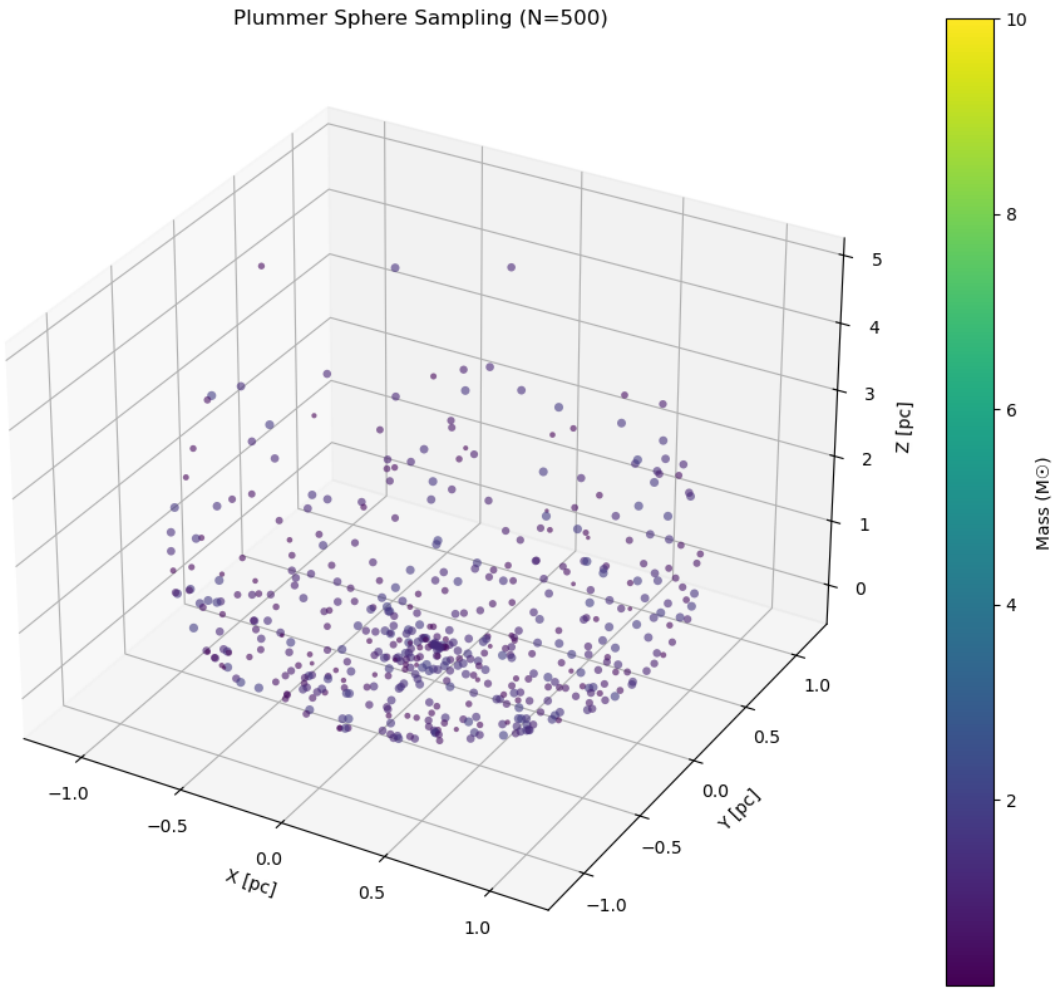

Plummer Sphere Sampling (N=500)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
2
4
6
8
10


Plummer Sphere Sampling (N=500)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
2
4
6
8
10

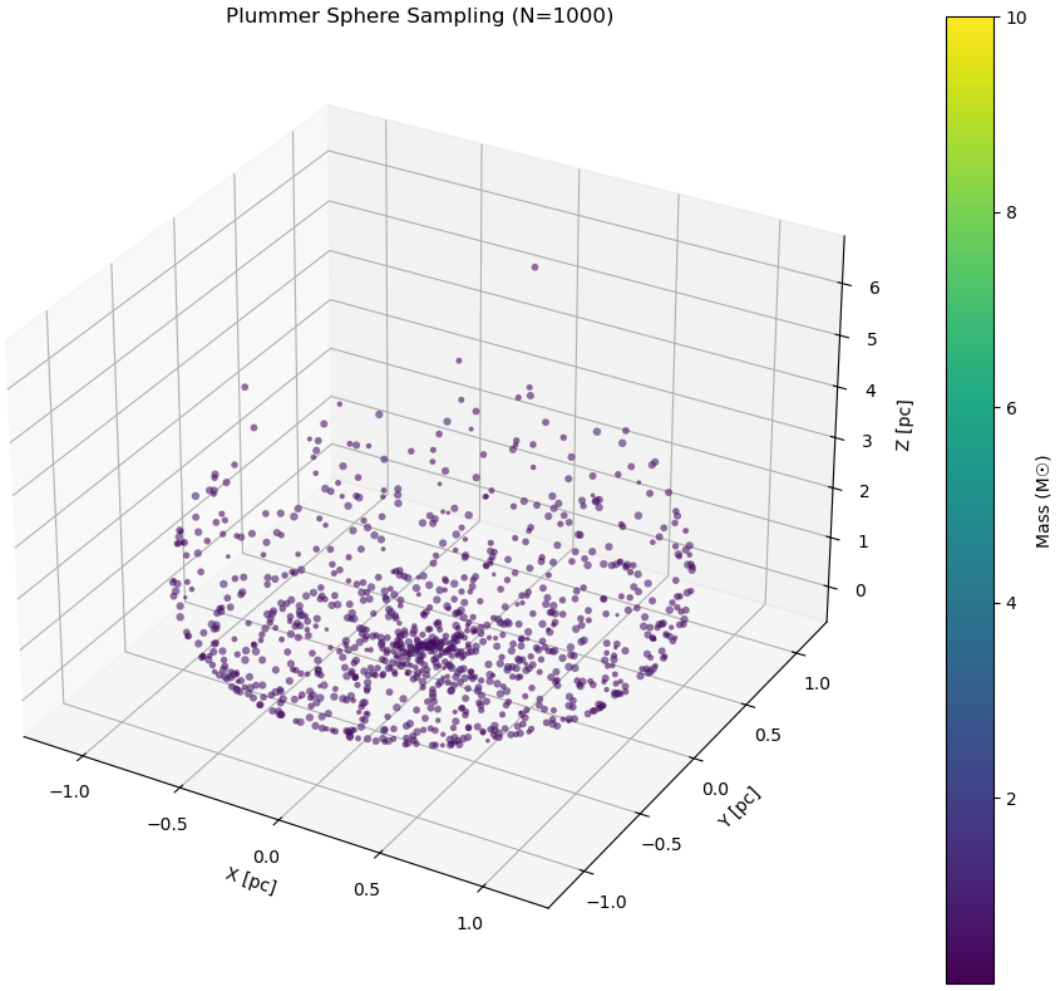

Plummer Sphere Sampling (N=500)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
2
4
6
8
10

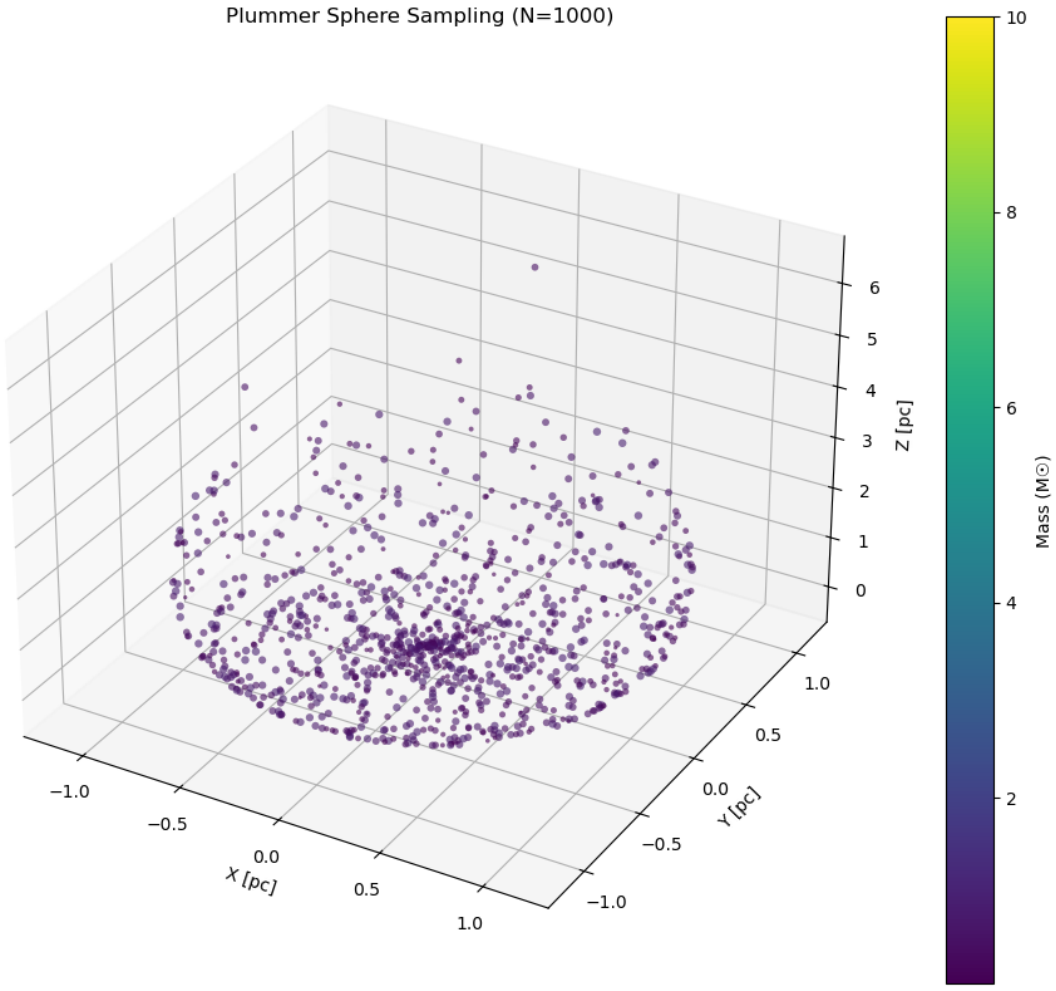

Plummer Sphere Sampling (N=500)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
2
4
6
8
10

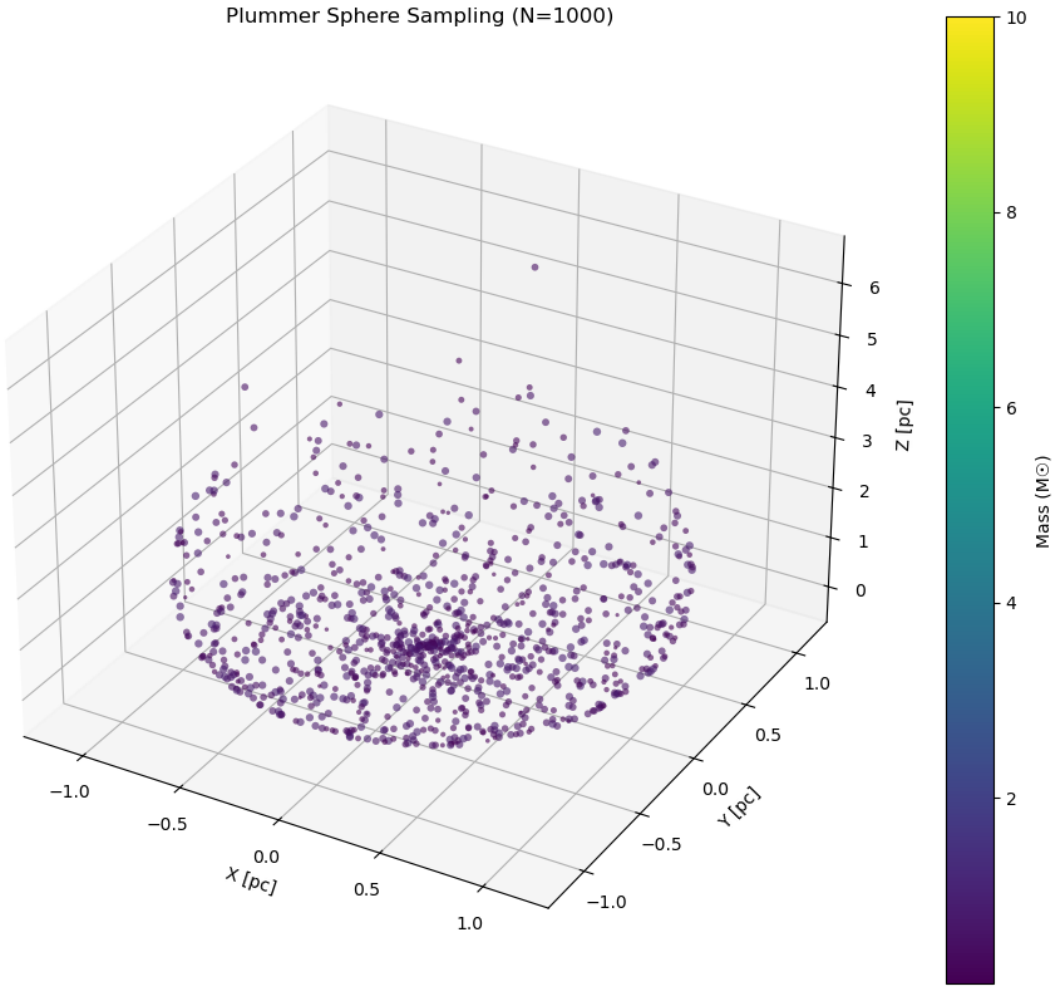

Plummer Sphere Sampling (N=500)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
2
4
6
8
10

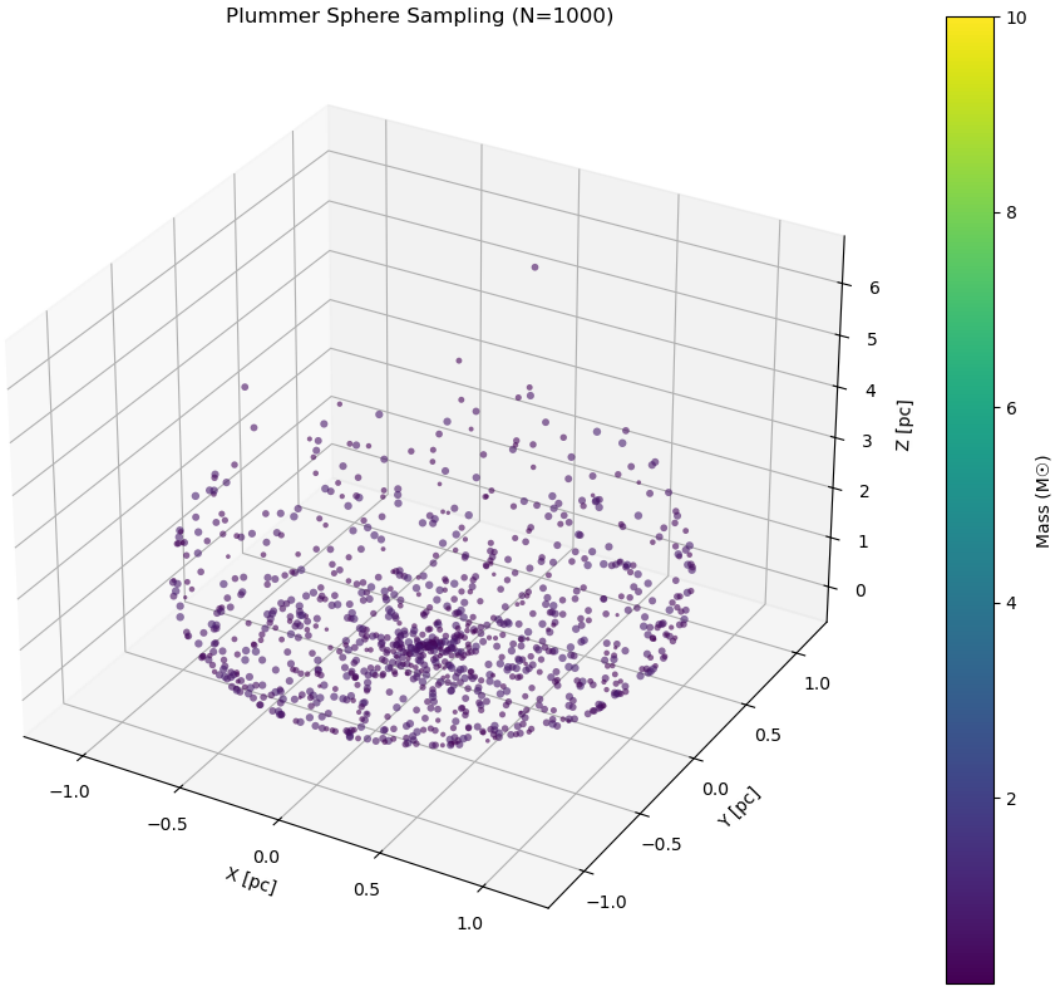


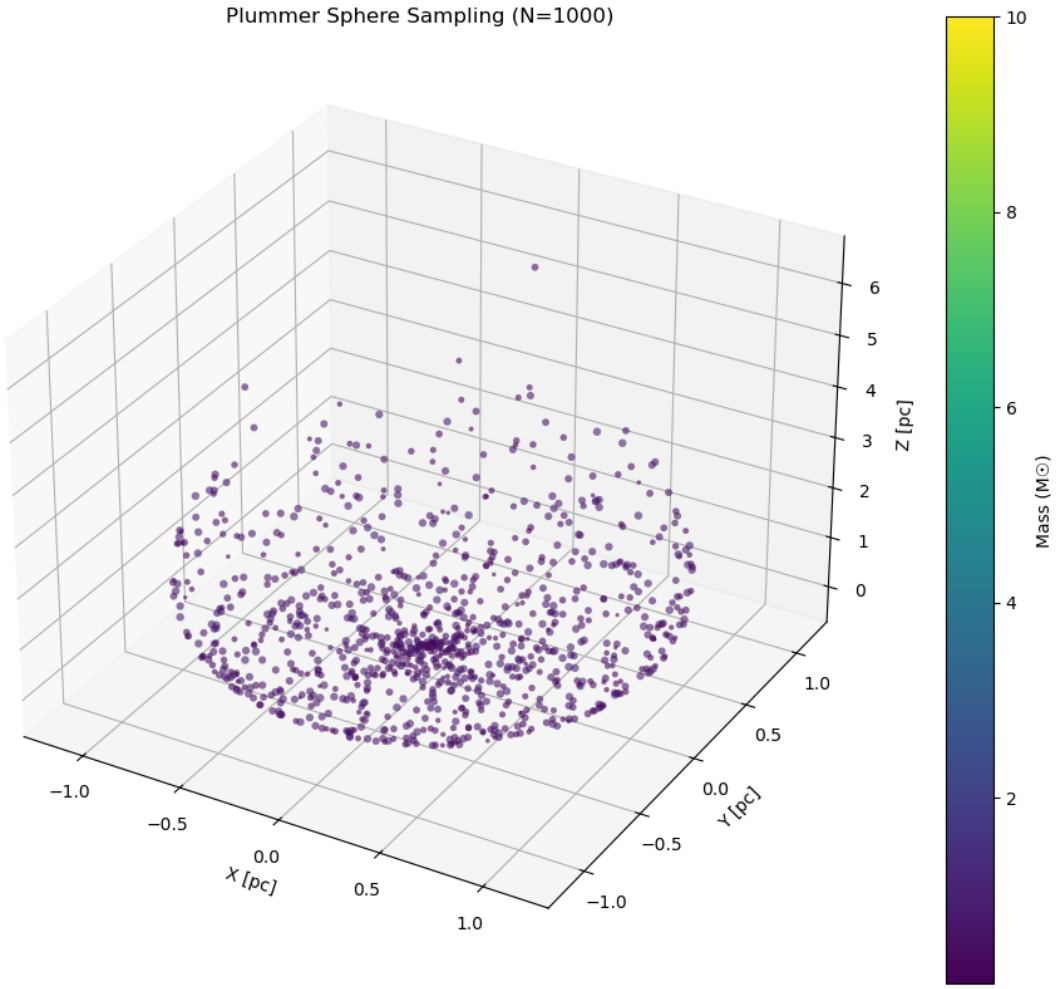

```
1.32 s ± 122 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
[[ 0.67920844 -0.65979879  3.90617023]
 [ 0.19170494 -1.08894849  2.20331072]
 [ 1.02502526  0.48937596  1.76863221]
 …
 [ 0.76391363 -0.75897749  0.65559099]
 [-0.18727914 -0.07791875 -0.22910879]
 [ 0.09259143 -0.87427308  0.15053799]]
```

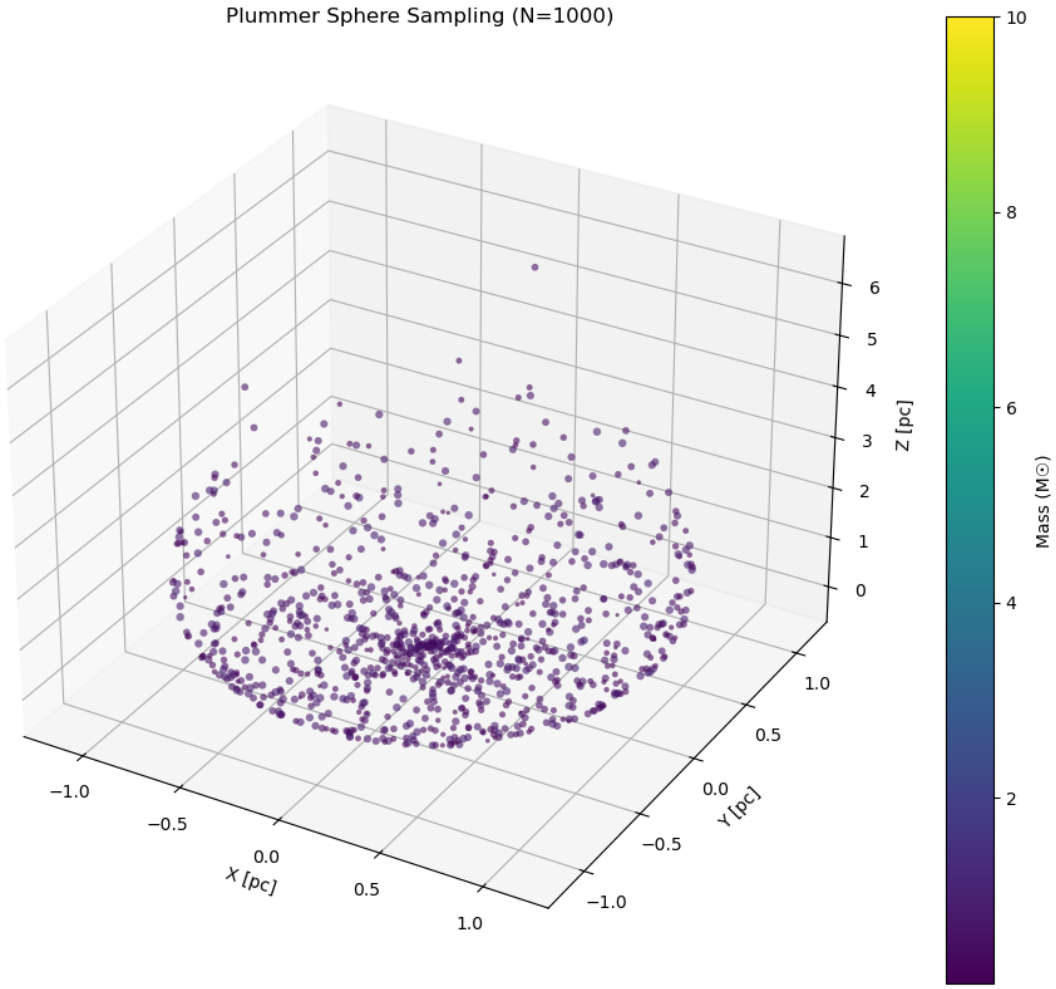

Plummer Sphere Sampling (N=1000)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
6
2
4
6
8
10

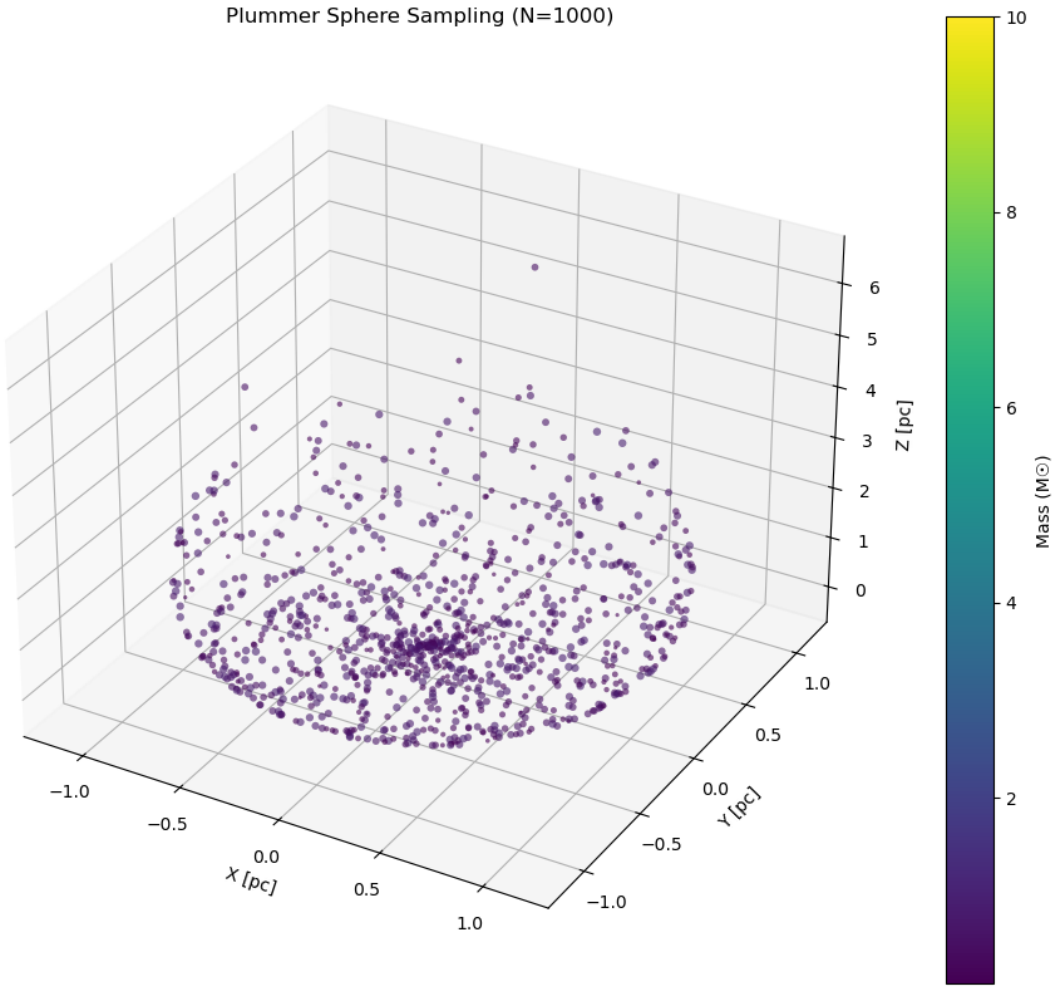

Plummer Sphere Sampling (N=1000)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
6
2
4
6
8
10

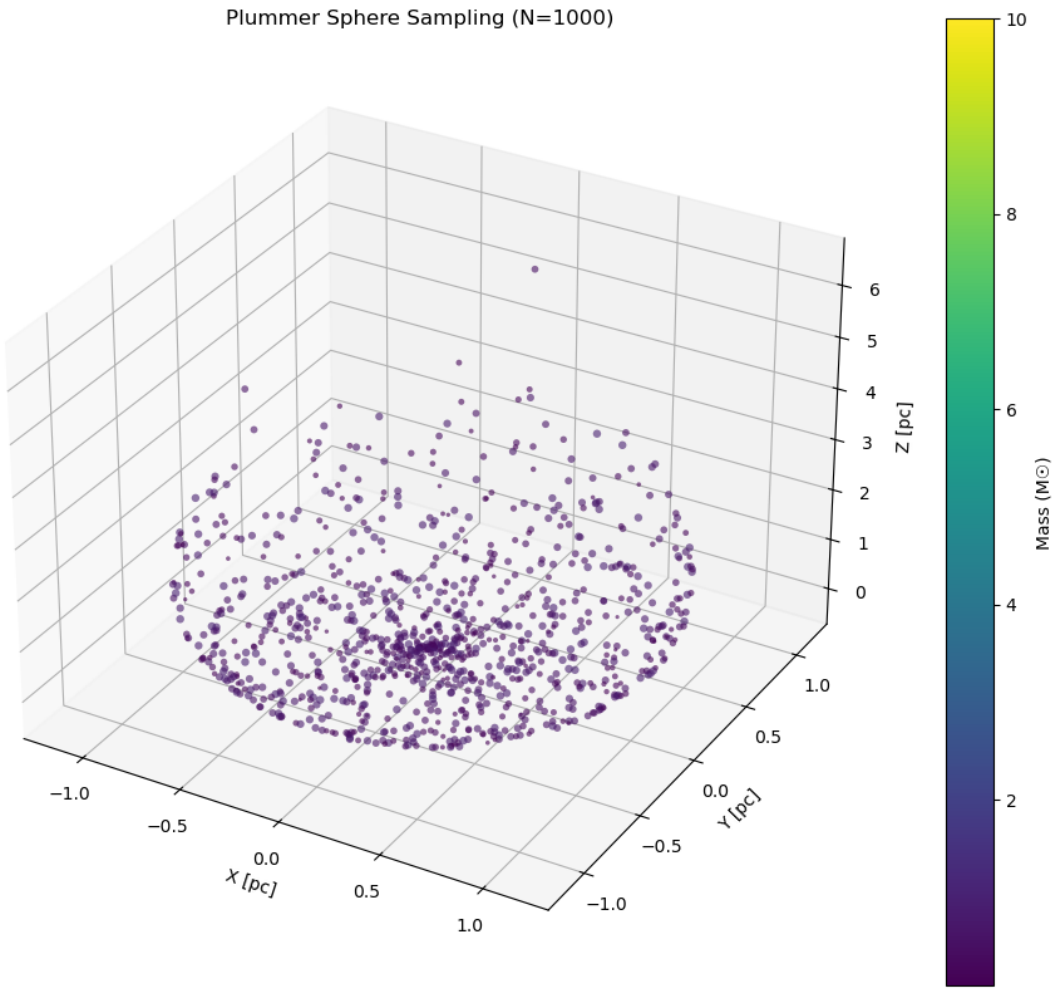

Plummer Sphere Sampling (N=1000)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
6
2
4
6
8
10


Plummer Sphere Sampling (N=1000)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
6
2
4
6
8
10


Plummer Sphere Sampling (N=1000)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
6
2
4
6
8
10


Plummer Sphere Sampling (N=1000)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
6
2
4
6
8
10


Plummer Sphere Sampling (N=1000)
X [pc]
Y [pc]
Z [pc]
Mass (M⊙)
−1.0
−0.5
0.0
0.5
1.0
0
1
2
3
4
5
6
2
4
6
8
10




```
1.38 s ± 63.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

0.2 Simulation & Informal Write Up

1) Your implementation of Monte Carlo sampling from the Plummer model.

I implemented the Monte Carlo Sampling from the Plummer Model by defining a function for n_body, distance, plummer density, kinetic energy, potential energy, total energy, and other equations defined.

I started out (going down from the sample_plummer function) defining the masses 0.1 - 10 as defined in the instructions. From there is needs to be normilized in order to work. Created the u, v dimenisions from 0 -> 1 for n steps.

After that Spherical coordinates need to be defined for radial distance using the U coordiate, then the theta coordinate, and then the phi.

After that I converted positions back from spherical to cartesian and calculated the velcotity values using my sample_velocity function.

For this function I was given pos (in terms of x,y,z) and calculated the radii. I then put it in a loop that was based on position, to update the velocity dispersions at each of the radii.

2)

0.3 Performance Write Up:

1) Execution Time per simulation for LP vs RK-4 Looking at what my execution time gave for LP vs RK-4

   - LP: 471 ms ± 45 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
   - RK-4: 2.07 s ± 115 ms per loop (mean ± std. dev. of 7 runs, 1 loop each) I can see that Leapfrog method took longer than RK-4 this makes sense, because based on (10 simulations) leapfrog is not good for small time scales. We can also see that RK-4 has a bigger standard deviation despite it being faster so we know that there is an issue between it.

2) Monte-Carlo Simulation I did it for a range of N = 10, 50, 100. This was my output (might differ for others)

   - 1.46 s ± 97.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
   - 1.51 s ± 79.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
   - 1.33 s ± 35.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
   - 1.32 s ± 122 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
   - 1.38 s ± 63.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

   I think its interesting to note that there was a decrease is run time for N = 100 I'm not sure why this may have happened…

[ ]: