N-Body project

Computational Astrophysics, HS24

04.02.2024

Rémy Moll

1 N-body forces and analytical solutions

Objective

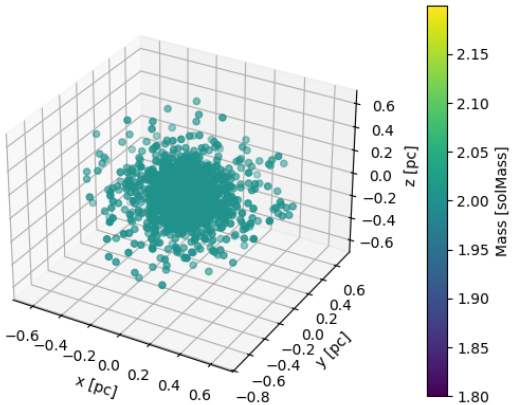
Implement naive N-body force computation and get an intuition of the challenges:

- accuracy
- computation time
- stability

$\Rightarrow$ still useful to compute basic quantities of the system, but too limited for large systems or the dynamical evolution of the system

Overview - the system

Get a feel for the particles and their distribution. [Code at 4.a]




```
### Direct N body force computation and comparison with analytical model
epsilon = utils.mean_interparticle_distance(particles)
```

Overview - the system (ii)

```
epsilon_range = np.logspace(-3, 3, 7)
n_squared_forces = []
for e in epsilon_range:
    n_particles = particles.shape[0]
    cache_file = CACHE_ROOT / f"n_squared_forces__n_{n_particles}
__softening_multiplier_{e:.0f}.npy"
    try:
        f = np.load(cache_file)
    except FileNotFoundError:
        f = utils.n_body_forces(particles, G, e * epsilon)
        np.save(cache_file, f)
        logger.debug(f"Saved forces to {cache_file}")
    n_squared_forces.append(f)

analytical_force = utils.analytical_forces(particles)
```

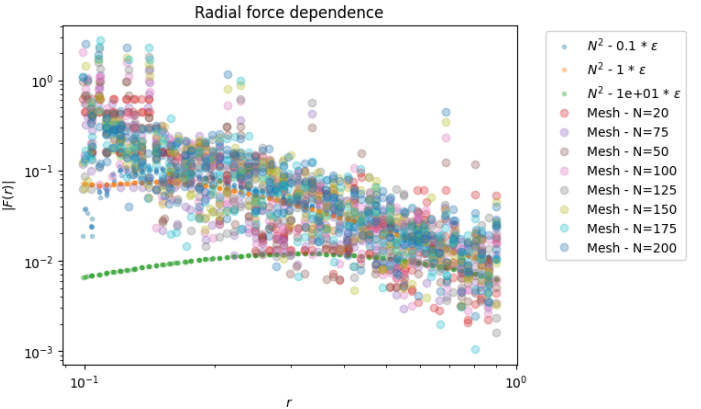
Inspecting the data

```
## Compare the mesh computation with the direct summation
r = np.linalg.norm(particles[:,:3], axis=1)

plt.figure()
plt.title('Radial force dependence')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('$r$')
plt.ylabel('$|F(r)|$')

# many of the particles have the same distance from the origin, so we skip some
of them
SKIP_N = 20

for f, e in zip(n_squared_forces, epsilon_range):
    # remove the black hole:
    plt.plot(r[1::SKIP_N], np.linalg.norm(f, axis=1)[1::SKIP_N], '.',
label=f"$N^2$ - {e:.1g} * $\\epsilon$", alpha=0.3)
for f, s in zip(mesh_forces, mesh_size_range):
    # remove the black hole:
    plt.plot(r[1::SKIP_N], np.linalg.norm(f, axis=1)[1::SKIP_N], 'o',
```

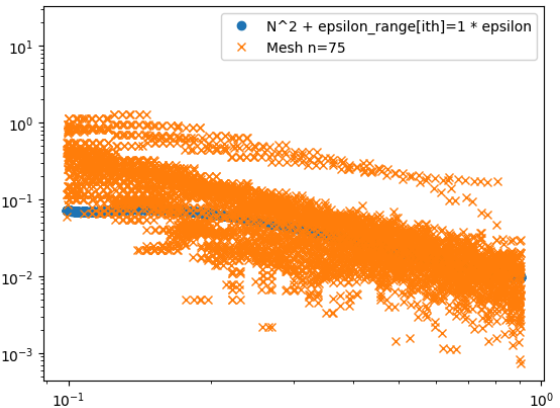
Inspecting the data (ii)

```
label=f"Mesh - N={s}", alpha=0.3)

plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()


# TODO: compare computation time

plt.figure()
plt.xscale('log')
plt.yscale('log')
ith = 1
f = n_squared_forces[ith]
f_val = np.linalg.norm(f, axis=1)
plt.plot(r, f_val, 'o', label=f'N^2 + {epsilon_range[ith]=:.2g} * epsilon')
f = mesh_forces[1]
f_val = np.linalg.norm(f, axis=1)
plt.plot(r, f_val, 'x', label=f'Mesh n={mesh_size_range[1]}')
plt.legend()
plt.show()
```

Inspecting the data (iii)



Inspecting the data (iv)



Density

Some images about the density

N Body and variations

sdsd

Relaxation

sd

2 Default Styling in diatypst

Terms, Code, Lists

diatypst defines some default styling for elements, e.g Terms created with `/ Term: Definition` will look like this

Term
Definition

A code block like this

```
// Example Code
print("Hello World!")
```

Lists have their marker respect the `title-color`

- A
  - AAA
    - B

1. AAA
2. BBB
3. CCC

3 Additional

Inspiration

this template is inspired by slydst, and takes part of the code from it. If you want simpler slides, look here!

4 Appendix - Code

Code

```
# Set G = 1
G = 1


# Since we have a globular cluster, we can use typical values
M_TOT = 1e5 * u.M_sun
R_TOT = 20 * u.pc

# Rescale the units of the particles
M_particles = particles[:,3].sum()
R_particles = np.max(np.linalg.norm(particles[:, :3], axis=1))
logger.info(f"Considering a globular cluster - total mass of particles:
{M_particles}, maximum radius of particles: {R_particles}")
m_scale = M_TOT / M_particles
r_scale = R_TOT / R_particles
utils.seed_scales(r_scale, m_scale)



### Plot again with scales
reduced = utils.remove_outliers(particles.copy())
```

Code (ii)

```
positions = utils.apply_units(reduced[:, :3], "position")
masses = utils.apply_units(reduced[:, 3], "mass")
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
sc = ax.scatter(positions[:,0], positions[:,1], positions[:,2], cmap='viridis',
c=masses)

ax.set_xlabel(f'x [{positions.unit}]')
ax.set_ylabel(f'y [{positions.unit}]')
ax.set_zlabel(f'z [{positions.unit}]')
cbar = plt.colorbar(sc, ax=ax, pad=0.1)
cbar.set_label(f'Mass [{masses.unit}]')

plt.show()
```