particles

September 18, 2023

1 AMUSE tutorial on particle sets

AMUSE particle sets are a handy tool for storing data

[419]:
```
#Load in the amuse units module
from amuse.units import units
from amuse.lab import Particles
```

[420]:
```
# Declare a single particle
sun_and_earth = Particles(2)
sun = sun_and_earth[0]
sun.mass = 1 | units.MSun
sun.position = (0,0,0) | units.au
sun.velocity = (0,0,0) | units.kms
print("Sun=", sun)
```

```
Sun= Particle(4058686528702482234, set=<140208969954304>
    , mass=1.0 MSun
    , vx=0.0 kms
    , vy=0.0 kms
    , vz=0.0 kms
    , x=0.0 au
    , y=0.0 au
    , z=0.0 au)
```

[421]:
```
# Now declare the Earth
from amuse.units.constants import G
earth = sun_and_earth[1]
earth.mass = 1 | units.MEarth
earth.position = (1, 0, 0) | units.au
def relative_orbital_velocity(mass, distance):
    return (G*mass/distance).sqrt()
vorb = relative_orbital_velocity(sun_and_earth.mass.sum(),
                                earth.position.sum())
earth.velocity = (0, 1, 0) * vorb
print("Earth=", earth)
```

```
Earth= Particle(6659848503659424852, set=<140208969954304>
```

```
    , mass=3.00273515275e-06 MSun
    , vx=0.0 kms
    , vy=29.7885123292 kms
    , vz=0.0 kms
    , x=1.0 au
    , y=0.0 au
    , z=0.0 au)
```

[422]:
```
sun_and_earth.move_to_center()
print(sun_and_earth)
```

```
                 key         mass          vx          vy          vz
x           y           z
                   -         MSun         kms         kms         kms
au          au          au
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========
 4058686528702482234    1.000e+00    0.000e+00   -8.945e-05    0.000e+00
-3.003e-06    0.000e+00    0.000e+00
 6659848503659424852    3.003e-06    0.000e+00    2.979e+01    0.000e+00
1.000e+00    0.000e+00    0.000e+00
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========
```

As you see, the particles have all the essential properties to define their orbit.

Now, let's give the particles a specific name (or other attribute)

[423]:
```
setattr(sun_and_earth, "name", "")
sun_and_earth.name = ["sun", "earth"]
```

How we have declared the particles and moved them to the center of mass. We can also search for a specific particle. For example, the one with the "sun" in the attribute "name".

[424]:
```
earth = sun_and_earth[sun_and_earth.name=="earth"]
print("Sun=", sun)
```

```
Sun= Particle(4058686528702482234, set=<140208969954304>
    , mass=1.0 MSun
    , name=sun
    , vx=0.0 kms
    , vy=-8.9446744534e-05 kms
    , vz=0.0 kms
    , x=-3.00272613635e-06 au
    , y=0.0 au
    , z=0.0 au)
```

We can add a moon in orbit around the earth

```
[425]: moon = Particles(1)
moon.name = "moon"
moon.mass = 7.34767309e+22 | units.kg
moon.position = (384400, 0, 0) | units.km
vorb = relative_orbital_velocity(earth.mass + moon.mass,
                                 moon.position.sum())
moon.velocity = (0, 1, 0) * vorb
print("moon=", moon)
```

```
moon=                    key         mass         name           vx           vy
vz            x            y            z
                   -          kg         none  44597309335878.1 * m * s**-1
44597309335878.1 * m * s**-1  44597309335878.1 * m * s**-1          km
km           km
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========  ===========
 4569898869789617676    7.348e+22         moon    0.000e+00    2.297e-11
0.000e+00    3.844e+05    0.000e+00    0.000e+00
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========  ===========
```

The moon, however, is not somewhere inside the Sun with zero velocity which is not good. We will have to replace the moon to make it orbit around the Earth. We do that by simply adding the positions and velocity of Earth to the moon's.

```
[426]: moon.position += earth.position
moon.velocity += earth.velocity
```

And we can add the moon to the Sun and Earth system

```
[427]: sun_and_earth.add_particle(moon)
```

```
[427]: <amuse.datamodel.particles.Particle at 0x7f84f1c25a80>
```

Note the use of the singular here, because we only add a single particle to the particle set sun_and_earth. It is probably better to rename the sun_and_earth now.

```
[428]: solarsystem = sun_and_earth
```

It is important now to recenter the entire system, because by adding the moon we shifted the center of mass.

```
[429]: solarsystem.move_to_center()
print("Solar system:", solarsystem)
```

```
Solar system:                    key         mass         name           vx
vy           vz            x            y            z
                   -        MSun         none          kms          kms
kms           au           au           au
```

```
==================== =========== =========== =========== ===========
=========== =========== =========== ===========
 4058686528702482234    1.000e+00         sun    0.000e+00   -9.059e-05
0.000e+00   -3.040e-06    0.000e+00    0.000e+00
 6659848503659424852    3.003e-06       earth    0.000e+00    2.979e+01
0.000e+00    1.000e+00    0.000e+00    0.000e+00
 4569898869789617676    3.694e-08        moon    0.000e+00    3.081e+01
0.000e+00    1.003e+00    0.000e+00    0.000e+00
==================== =========== =========== =========== ===========
=========== =========== =========== ===========
```

We can now manipulate the planetary system, or query it. for example by querying the masses.

[430]:
```
print("mass=", solarsystem.mass.in_(units.MEarth))
```

```
mass= [333029.704297, 1.0, 0.0123031263019] MEarth
```

This gives us a list of the masses of all objects, in units the the earth's mass. In fact, each of the particle's attributes is a simple numpy array: it can be assigned and manipulated as such.

Another way to ecquire the same information could be done as follows:

[431]:
```
print("mass=", solarsystem.mass/solarsystem[1].mass)
```

```
mass= [  3.33029704e+05   1.00000000e+00   1.23031263e-02]
```

In some (hopefully rare) cases you may want to use the particle set or its attributes as simple numpy arrays, without the units. This is easily achieved by stripping the unit from the array. This can be realied by explicitely querying the selected parameter with that specific unit.

[432]:
```
solarsystem.position.value_in(units.parsec)
```

[432]:
```
array([[ -1.47371911e-11,   0.00000000e+00,   0.00000000e+00],
       [  4.84812207e-06,   0.00000000e+00,   0.00000000e+00],
       [  4.86057963e-06,   0.00000000e+00,   0.00000000e+00]])
```

which, in this case, gives you a 2-dimensional *numpy.array* of the positions of star, planet and moon in units of a parsec.

Now, you may want to query the particle set solarsystem. for example by asking what are all its attributes. this can be done as follows:

Or get some general help on the underlying particle class

You have performed some rudimentary operations on a particle set.

It is now time to experiment a little for yourself.

1.1 Assignments and questions:

1.1.1 Assignment 1:

Add the planet Jupiter (see Wikipedia) to your small planetary system.

1.1.2 Assignment 2:

Your planetary system is notoriously planar, and initialy the earth and moon are positioned along the Cartesian x-axis with the velocity vector in the Cartesian y-direction.

Make the Sun-Earth-Moon system more realistic by introducing a small inclination to the Earth's and Moon's orbits and by giving them a random mean anomaly.

The Orbital element module of AMUSE could come in handy.

1.1.3 Assignment 3:

Calculate the total gravitational binding energy of solarsystem.

Now displace the entire particle set by 100 parsec and give it a linear velocity of 100km/s in the z-direction. Then calculate the binding energy of the system again.

Did the binding energy of the Solar system change by this translation?

1.1.4 Question 1:

Particle sets have the attribute *get_binaries()*. If you use this function to check the binaries in your system you will find that (without Jupiter) you have 3 binaries. Explain why the Sun is in a binary with the Moon. You may want to take a look at the source code.

1.1.5 Assignment 4:

Generate another particle set with a 2 solar-mass star and two planets of 10 and 100 Earth masses in cirular orbits at 0.1 and 0.6 au. Place this second planetary system at apocenter around your Solar system (true anomaly of 180 degrees) at a semimajor axis of 60 au with an eccentricity of 0.6. Then move the entire system to the center of mass.

1.1.6 Question 2:

Which of the orbits of the binary star with planets from *Assignment 4* has the highest binding energy?

1.1.7 Assignment 1

[433]:
```python
jup = Particles(1)
jup.name = "Jupiter"
jup.mass = 1.89813e27 | units.kg
jup.position = (5.2, 0, 0) | units.AU
vorb = relative_orbital_velocity(sun.mass + jup.mass,
                                 jup.position.sum())
jup.velocity = (0, 1, 0) * vorb
print("jup=", jup)
```

```
jup=                  key         mass         name             vx             vy
vz             x             y             z
                    -          kg         none   3646245880.357221 * m * s**-1
3646245880.357221 * m * s**-1  3646245880.357221 * m * s**-1           AU
```

```
AU          AU
==================== =========== =========== =========== ===========
=========== =========== =========== ===========
17467802250472390238    1.898e+27     Jupiter    0.000e+00    3.584e-06
0.000e+00    5.200e+00    0.000e+00    0.000e+00
==================== =========== =========== =========== ===========
=========== =========== =========== ===========
```

[434]:

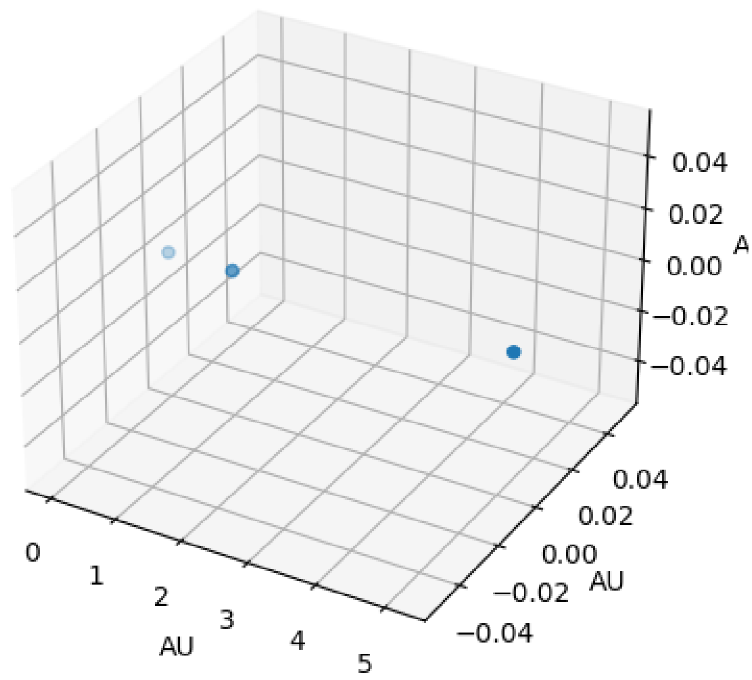
```
jup.position += sun.position
jup.velocity += sun.velocity
solarsystem = solarsystem
solarsystem.add_particle(jup)
solarsystem.move_to_center()
print("Solar system:", solarsystem)
```

```
Solar system:                   key         mass         name            vx
vy            vz            x            y            z
                   -          MSun         none          kms           kms
kms            au            au            au
==================== =========== =========== =========== ===========
=========== =========== =========== ===========
 4058686528702482234    1.000e+00          sun    0.000e+00   -1.255e-02
0.000e+00   -4.961e-03    0.000e+00    0.000e+00
 6659848503659424852    3.003e-06        earth    0.000e+00    2.978e+01
0.000e+00    9.950e-01    0.000e+00    0.000e+00
 4569898869789617676    3.694e-08         moon    0.000e+00    3.080e+01
0.000e+00    9.976e-01    0.000e+00    0.000e+00
17467802250472390238    9.544e-04      Jupiter    0.000e+00    1.306e+01
0.000e+00    5.195e+00    0.000e+00    0.000e+00
==================== =========== =========== =========== ===========
=========== =========== =========== ===========
```

[435]:

```
#Check if correct

import matplotlib.pyplot as plt

ax = plt.axes(projection='3d')

xdata = solarsystem.position.value_in(units.AU)[:,0]
ydata = solarsystem.position.value_in(units.AU)[:,1]
zdata = solarsystem.position.value_in(units.AU)[:,2]
ax.scatter3D(xdata, ydata, zdata);
ax.set_xlabel('AU')
ax.set_ylabel('AU')
ax.set_zlabel('AU')
```

[435]: Text(0.5, 0, 'AU')



[436]:
```
solarsystem.remove_particle(jup)
solarsystem.move_to_center()
```

1.1.8 Assignment 2

[437]:
```
import numpy as np
from amuse.ext import orbital_elements as oe

sun_i, earth_i = oe.generate_binaries(
        1 | units.MSun,
        1 | units.MEarth,
        1 | units.AU,
        0.0167,
        2*np.pi*np.random.rand() | units.rad,
        23.5 | units.deg,
        )

_, moon_i = oe.generate_binaries(
        1 | units.MEarth,
        1/81 | units.MEarth,
        0.0025695 | units.AU,
        0.0549,
        2*np.pi*np.random.rand() | units.rad,
```

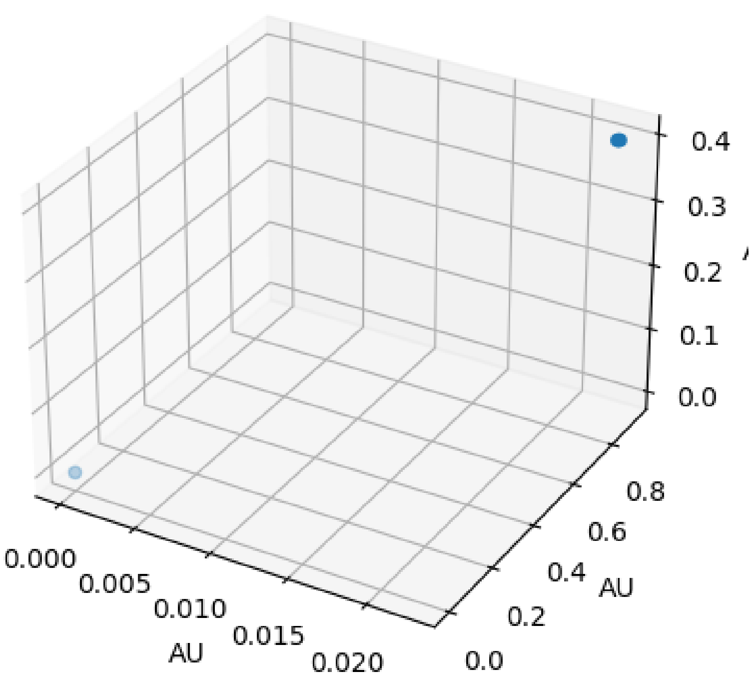
```
        5.1 | units.deg,
        )
```

[438]:
```
moon_i.position += earth_i.position
moon_i.velocity += earth_i.velocity
```

[439]:
```
solarsystem_i = Particles(3, particles=[sun_i,earth_i,moon_i])
solarsystem_i.move_to_center()
```

[440]:
```
#Check if correct

import matplotlib.pyplot as plt

ax = plt.axes(projection='3d')

xdata = solarsystem_i.position.value_in(units.AU)[:,0]
ydata = solarsystem_i.position.value_in(units.AU)[:,1]
zdata = solarsystem_i.position.value_in(units.AU)[:,2]
ax.scatter3D(xdata, ydata, zdata);
ax.set_xlabel('AU')
ax.set_ylabel('AU')
ax.set_zlabel('AU')
```

[440]: Text(0.5, 0, 'AU')



1.1.9 Assignment 3

[441]:
```
def Ekin(mass, vel):
    return 0.5*mass*(vel.length())**2

def Egrav(mass1, mass2, x1, x2, y1, y2, z1, z2):
    r12 = ((x1-x2)**2+(y1-y2)**2+(z1-z2)**2).sqrt()
    return (G*mass1*mass2)/r12



def E_grav_binding(system):
    E_bind = 0 | units.J
    for i in range(len(system)):
        for j in range(len(system)):
            if i == j:
                E = Ekin(system.mass[j], system.velocity[j])
                E_bind += E
            else:
                #the following line is not the prettiest but I find this the␣
 ↪easiest to not make mistakes and understand what is happening!
                E = Egrav(system.mass[i], system.mass[j], system.
 ↪position[i][0], system.position[j][0]
                          , system.position[i][1], system.position[j][1],␣
 ↪system.position[i][2], system.position[j][2])
                E_bind -= E
    return E_bind

E_bind = E_grav_binding(solarsystem_i)


print("The bindingsenergy is", E_bind.in_(units.J), "so the system is␣
 ↪gravitationally bound")
```

```
The bindingsenergy is -8.05348921255e+33 J so the system is gravitationally
bound
```

Now we translate the system

[442]:
```
solarsystem_i.position += 100 | units.parsec
```

[443]:
```
solarsystem_i.velocity.in_(units.kms)
```

[443]:
```
quantity<[[9.05029634168e-05, -3.26499157332e-06, -1.41946983273e-06],
[-29.784963164, 1.0738991089, 0.466944622199], [-28.7721704295, 1.08864561251,
0.468260712584]] kms>
```

[444]:
```
solarsystem_i.velocity += (0,0,100) | units.kms
```

We do not move the system to the center as this will get rid of the translation

[445]:
```
E_bind_trans = E_grav_binding(solarsystem_i)

print("The bindingsenergy after translation is", E_bind_trans.in_(units.J), "so␣
↪the system is not gravitationally bound anymore")
```

```
The bindingsenergy after translation is 9.94462217617e+39 J so the system is not
gravitationally bound anymore
```

1.1.10 Question 1

[446]:
```
#Translate back
solarsystem_i.position -= 100 | units.parsec
solarsystem_i.velocity -= (0,0,100) | units.kms
```

[447]:
```
from amuse.datamodel import particle_attributes as pa

print(solarsystem_i)

result = pa.get_binaries(solarsystem_i)

print(len(result))
```

```
                 key         mass            vx            vy            vz
x            y            z
                   -   1.98892e+30 * kg  3646245880.3572216 * m * s**-1
3646245880.3572216 * m * s**-1  3646245880.3572216 * m * s**-1  149597870691.0 *
m  149597870691.0 * m  149597870691.0 * m
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========
 8799327417738180604    1.000e+00    2.482e-11   -8.954e-13   -3.893e-13
-6.706e-08   -2.787e-06   -1.211e-06
 2345422846309053240    3.003e-06   -8.169e-06    2.945e-07    1.281e-07
2.259e-02    9.162e-01    3.984e-01
 7401319078134742752    3.707e-08   -7.891e-06    2.986e-07    1.284e-07
2.249e-02    9.137e-01    3.982e-01
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========
3
```

The Sun and Moon also have a small impact on the orbit of each other, so they are a binary. Although the impact is small and negligible

1.1.11 Assignment 4

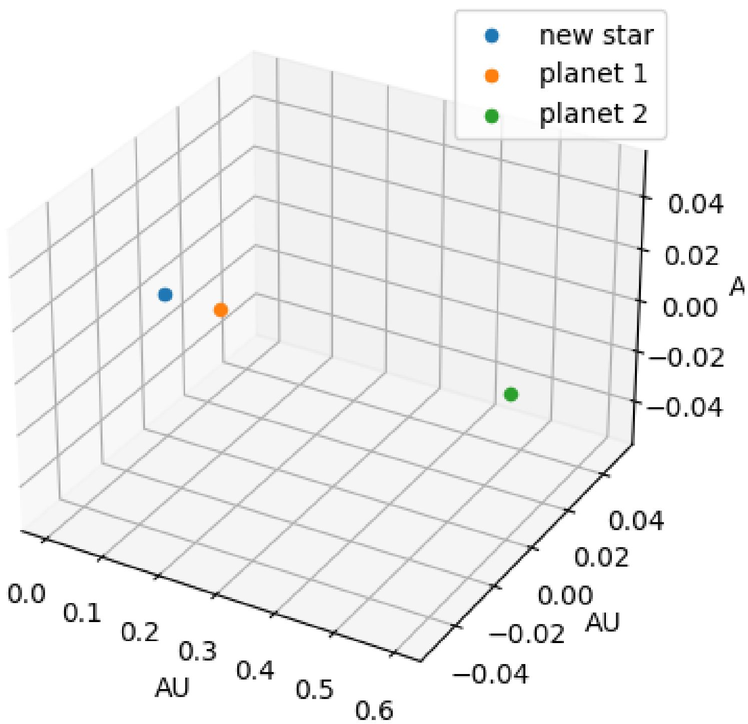
[448]:
```
new_sun, new_star = oe.generate_binaries(
        1 | units.MSun,
        2 | units.MSun,
        60 | units.AU,
        0.6,
        180 | units.deg,
        )

_, planet_1 = oe.generate_binaries(
        2 | units.MSun,
        10 | units.MEarth,
        0.1 | units.AU,
        )

_, planet_2 = oe.generate_binaries(
        2 | units.MSun,
        100 | units.MEarth,
        0.6 | units.AU,
        )
```

[449]:
```
planet_1.position += new_star.position
planet_1.velocity += new_star.velocity

planet_2.position += new_star.position
planet_2.velocity += new_star.velocity
```

[450]:
```
#Check if new planetary system is correct

new_plan_system = Particles(3, particles=[new_star,planet_1,planet_2],␣
 ↪name=["new star", "planet 1", "planet 2"])
new_plan_system.move_to_center()

ax = plt.axes(projection='3d')

for p in new_plan_system: ax.scatter(*p.position.value_in(units.AU), label=p.
 ↪name)
ax.set_xlabel('AU')
ax.set_ylabel('AU')
ax.set_zlabel('AU')
ax.legend()

print(new_plan_system)
```

```
                 key         mass         name             vx             vy
vz             x             y            z
                   -         MSun         none   3646245880.357221 * m * s**-1
```

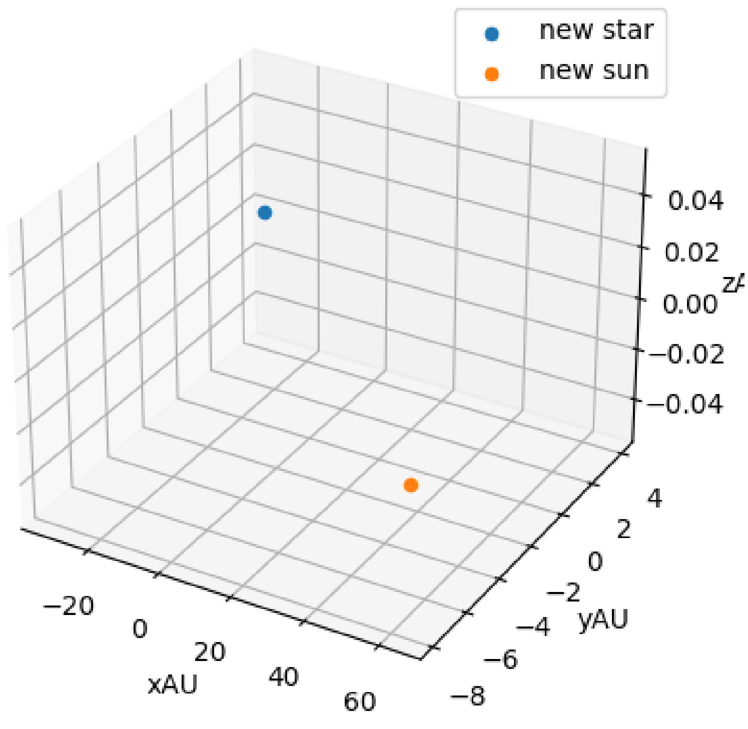
```
3646245880.357221 * m * s**-1  3646245880.357221 * m * s**-1  149597870691.0 * m
149597870691.0 * m  149597870691.0 * m
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========  ===========
13842791319784104811    2.000e+00     new star    2.351e-38   -2.787e-09
0.000e+00   -9.155e-05    0.000e+00    0.000e+00
 5183797904297149354    3.003e-05     planet 1    2.351e-38    3.653e-05
0.000e+00    9.991e-02    0.000e+00    0.000e+00
 1811948128892497646    3.003e-04     planet 2    2.351e-38    1.491e-05
0.000e+00    5.998e-01    0.000e+00    0.000e+00
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========  ===========
```



[451]:
```python
#Check if binary star system is correct

new_bi_system = Particles(2, particles=[new_star,new_sun], name=["new star",␣
 ↪"new sun"])
new_bi_system.move_to_center()

ax = plt.axes(projection='3d')

for p in new_bi_system: ax.scatter(*p.position.value_in(units.AU), label=p.name)
ax.set_xlabel('xAU')
```

```
ax.set_ylabel('yAU')
ax.set_zlabel('zAU')
ax.legend()

print(new_bi_system)
```

```
                 key          mass          name            vx            vy
vz            x            y            z
                   -          MSun          none  3646245880.357221 * m * s**-1
3646245880.357221 * m * s**-1  3646245880.357221 * m * s**-1  149597870691.0 * m
149597870691.0 * m  149597870691.0 * m
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========  ===========
13842791319784104811    2.000e+00     new star   -9.322e-23   -3.045e-07
0.000e+00   -3.200e+01    3.919e-15    0.000e+00
 5225630803776064165    1.000e+00      new sun    1.864e-22    6.089e-07
0.000e+00    6.400e+01   -7.838e-15   -0.000e+00
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========  ===========
```



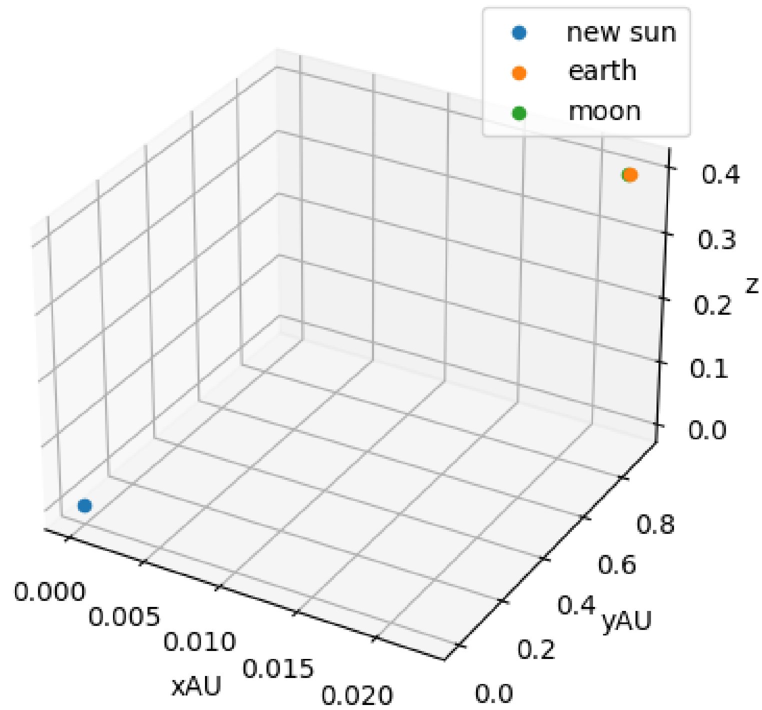
[452]:
```
earth_i.position += new_sun.position
earth_i.velocity += new_sun.velocity
```

```
moon_i.position += new_sun.position
moon_i.velocity += new_sun.velocity
```

[453]:

```
#Check if original solar system is correct

new_sol_system = Particles(3, particles=[new_sun,earth_i,moon_i], name=["new␣
 ↪sun", "earth", "moon"])
new_sol_system.move_to_center()

ax = plt.axes(projection='3d')

for p in new_sol_system: ax.scatter(*p.position.value_in(units.AU), label=p.
 ↪name)
ax.set_xlabel('xAU')
ax.set_ylabel('yAU')
ax.set_zlabel('zAU')
ax.legend()

print(new_sol_system)
```

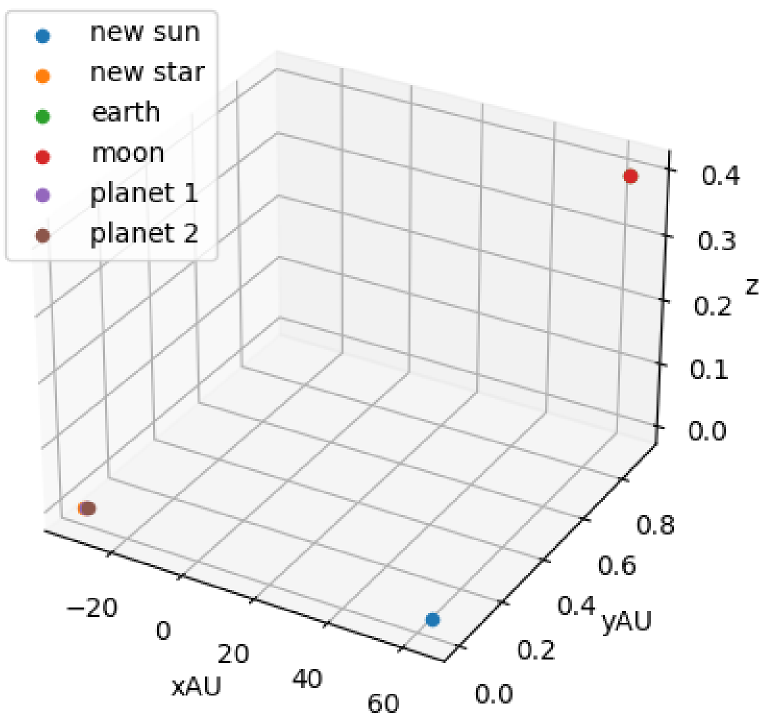
```
                 key         mass         name           vx           vy
vz            x            y            z
                   -  1.98892e+30 * kg         none  3646245880.3572216 * m *
s**-1  3646245880.3572216 * m * s**-1  3646245880.3572216 * m * s**-1
149597870691.0 * m  149597870691.0 * m  149597870691.0 * m
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========  ===========
 5225630803776064165    1.000e+00      new sun    2.482e-11   -8.954e-13
-3.893e-13   -6.867e-08   -2.785e-06   -1.211e-06
 2345422846309053240    3.003e-06        earth   -8.169e-06    2.945e-07
1.281e-07    2.259e-02    9.162e-01    3.984e-01
 7401319078134742752    3.707e-08         moon   -7.891e-06    2.986e-07
1.284e-07    2.249e-02    9.137e-01    3.982e-01
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========  ===========
```



[454]:
```
#Check if everything is correct

new_system = Particles(5,␣
↪particles=[new_sun,new_star,earth_i,moon_i,planet_1,planet_2], name=["new␣
↪sun","new star","earth", "moon","planet 1","planet 2"])
new_system.move_to_center()

ax = plt.axes(projection='3d')

for p in new_system: ax.scatter(*p.position.value_in(units.AU), label=p.name)
ax.set_xlabel('xAU')
ax.set_ylabel('yAU')
ax.set_zlabel('zAU')
ax.legend()

print(new_system)
```

```
                 key          mass          name             vx             vy
vz             x              y             z
                   -   1.98892e+30 * kg          none   3646245880.3572216 * m *
s**-1  3646245880.3572216 * m * s**-1  3646245880.3572216 * m * s**-1
149597870691.0 * m  149597870691.0 * m  149597870691.0 * m
====================  ===========  ===========  ===========  ===========
```

```
===========  ===========  ===========  ===========
 5225630803776064165   1.000e+00    new sun   8.273e-12   6.071e-07
-1.298e-13   6.400e+01   -9.282e-07   -4.036e-07
1384279131978410481 1   2.000e+00    new star   8.273e-12   -3.063e-07
-1.298e-13   -3.200e+01   -9.282e-07   -4.036e-07
 2345422846309053240   3.003e-06     earth   -8.169e-06   9.016e-07
1.281e-07   6.403e+01   9.162e-01   3.984e-01
 7401319078134742752   3.707e-08      moon   -7.891e-06   9.057e-07
1.284e-07   6.403e+01   9.137e-01   3.982e-01
 5183797904297149354   3.003e-05    planet 1   8.273e-12   3.623e-05
-1.298e-13   -3.190e+01   -9.282e-07   -4.036e-07
 1811948128892497646   3.003e-04    planet 2   8.273e-12   1.461e-05
-1.298e-13   -3.140e+01   -9.282e-07   -4.036e-07
====================  ===========  ===========  ===========  ===========
===========  ===========  ===========  ===========
```



It looks wonky but is correct I think. The axis are not in proportion. Due to the 180 degrees difference, the planes in which the planets orbit are perpendicular

1.1.12 Question 2

[457]:
```
for binary in pa.get_binaries(new_system):
    print(binary.name, E_grav_binding(binary))
```

```
['new star' 'planet 1'] 8.8197622154e+35 J
['new star' 'planet 2'] -2.05587858922e+35 J
['new star' 'new sun'] -6.61826328737e+37 J
['new sun' 'moon'] 4.87297385926e+36 J
['new sun' 'earth'] 4.86515031598e+36 J
```

So the new sun and moon, there are the two most “opposite” bodies