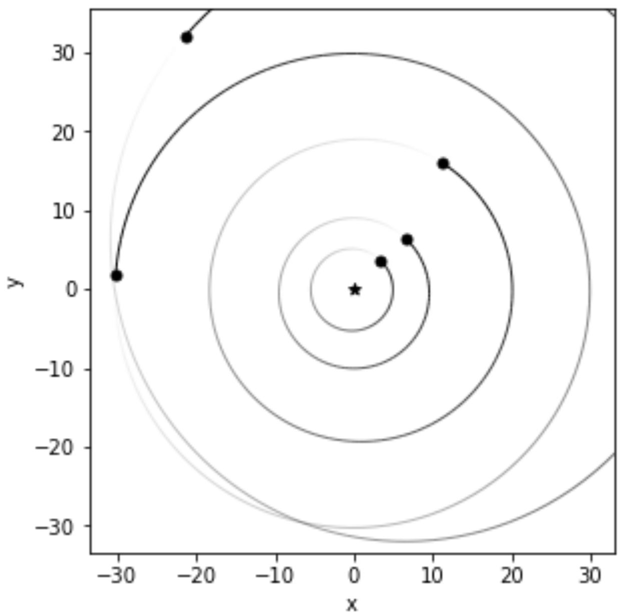
Integrations with REBOUND

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import rebound
```

In [2]:
```
sim = rebound.Simulation()
rebound.data.add_outer_solar_system(sim)
```

In [3]:
```
rebound.OrbitPlot(sim)
```

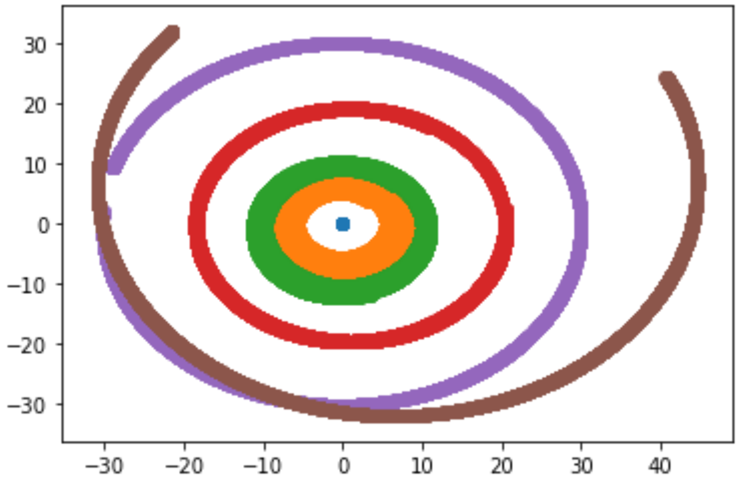
Out[3]:
```
(<Figure size 360x360 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fe4f223e2d0>)
```



In [4]:
```
def eulerstep(sim):
    sim.integrator = "None"
    sim.step() # Calculates the accelerations
    for i in range(sim.N):
        sim.particles[i].x += sim.dt * sim.particles[i].vx
        sim.particles[i].y += sim.dt * sim.particles[i].vy
        sim.particles[i].z += sim.dt * sim.particles[i].vz
        sim.particles[i].vx += sim.dt * sim.particles[i].ax
        sim.particles[i].vy += sim.dt * sim.particles[i].ay
        sim.particles[i].vz += sim.dt * sim.particles[i].az
```

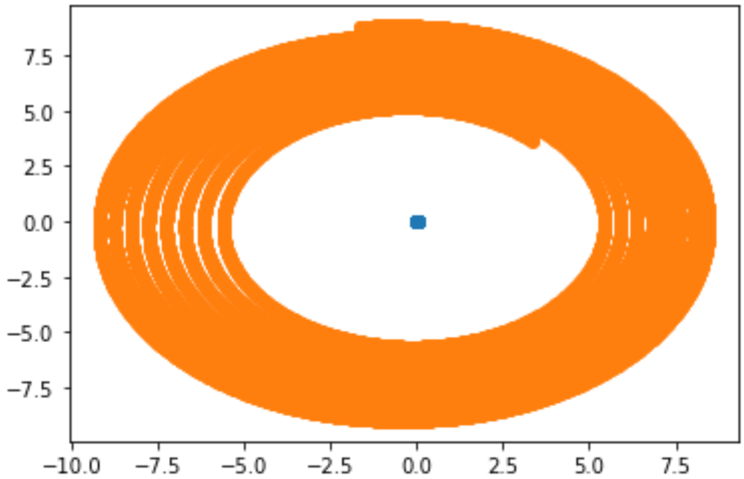
In [5]:
```
sim.dt = 1e-1
Nsteps = int(1000/sim.dt)
xyz = np.zeros((Nsteps, sim.N,3))
for i in range(Nsteps):
    eulerstep(sim)
    for j in range(sim.N):
        xyz[i][j] = sim.particles[j].xyz
```

In [6]:
```python
for i in range(sim.N):
    plt.scatter(xyz[:,i,0], xyz[:,i,1])
```



In [7]:
```python
plt.scatter(xyz[:,0,0], xyz[:,0,1])
plt.scatter(xyz[:,1,0], xyz[:,1,1])
# Euler method is really bad, and you can see that below because th
e orbit is varying too much
```

Out[7]: <matplotlib.collections.PathCollection at 0x7fe4f04f5990>

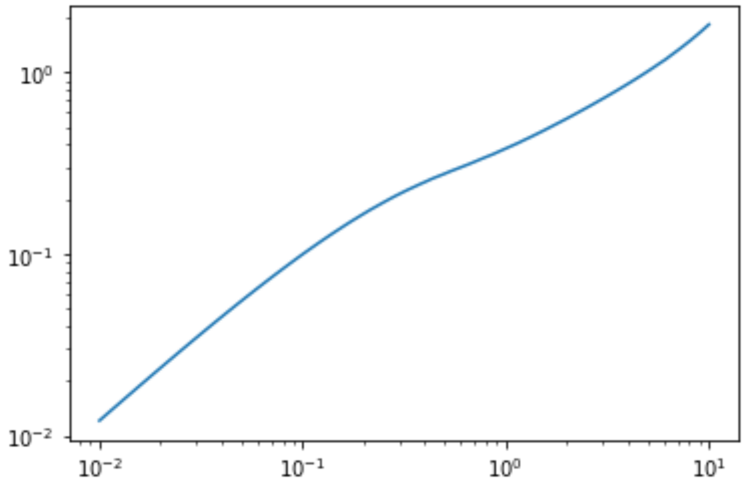


In [8]:
```python
def measureError(dt, step):
    sim = rebound.Simulation()
    rebound.data.add_outer_solar_system(sim)
    sim.dt = dt
    Nsteps = int(100/sim.dt)
    Ei = sim.calculate_energy()
    for i in range(Nsteps):
        step(sim)
    Ef = sim.calculate_energy()
    return np.abs((Ef-Ei)/Ei)
```

In [9]:
```python
dts = np.logspace(-2,1,100)
errors_euler = np.zeros(len(dts))
for i in range(len(dts)):
    errors_euler[i] = measureError(dts[i], eulerstep)
```

In [10]:
```
plt.xscale("log")
plt.yscale("log")
plt.plot(dts, errors_euler)
```

Out[10]: [<matplotlib.lines.Line2D at 0x7fe4f0463950>]

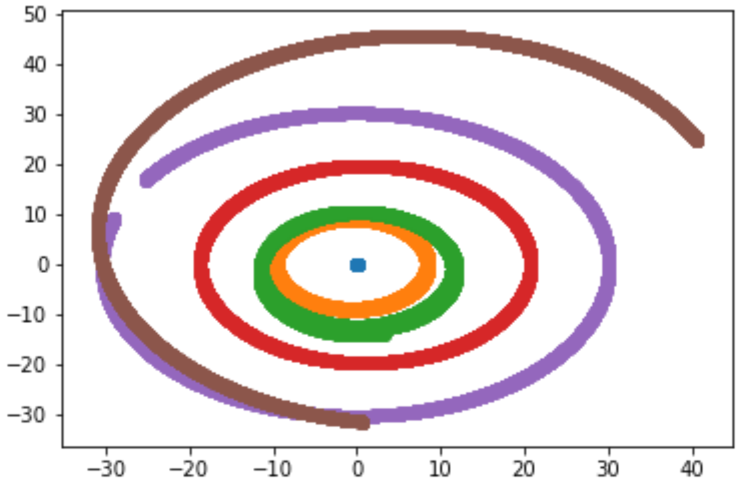


Task 1

In [11]:
```
def leapfrog(sim):
    sim.integrator = "None"
    # sim.step() # Calculates the accelerations
    for i in range(sim.N):
        sim.particles[i].x += 0.5*sim.dt * sim.particles[i].vx
        sim.particles[i].y += 0.5*sim.dt * sim.particles[i].vy
        sim.particles[i].z += 0.5*sim.dt * sim.particles[i].vz
    sim.step()
    for i in range(sim.N):
        sim.particles[i].vx += sim.dt * sim.particles[i].ax
        sim.particles[i].vy += sim.dt * sim.particles[i].ay
        sim.particles[i].vz += sim.dt * sim.particles[i].az
    for i in range(sim.N):
        sim.particles[i].x += 0.5*sim.dt * sim.particles[i].vx
        sim.particles[i].y += 0.5*sim.dt * sim.particles[i].vy
        sim.particles[i].z += 0.5*sim.dt * sim.particles[i].vz
```

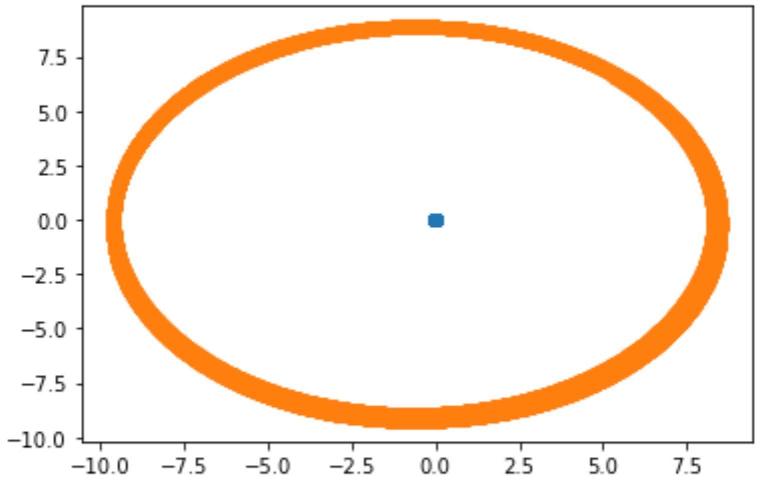
In [12]:
```
# Checking out leapfrog
sim.dt = 1e-1
Nsteps = int(1000/sim.dt)
xyz = np.zeros((Nsteps, sim.N,3))
for i in range(Nsteps):
    leapfrog(sim)
    for j in range(sim.N):
        xyz[i][j] = sim.particles[j].xyz
```

In [13]:
```python
for i in range(sim.N):
    plt.scatter(xyz[:,i,0], xyz[:,i,1])
```



In [14]:
```python
# Checking out the Jupiter orbit
plt.scatter(xyz[:,0,0], xyz[:,0,1])
plt.scatter(xyz[:,1,0], xyz[:,1,1])
```

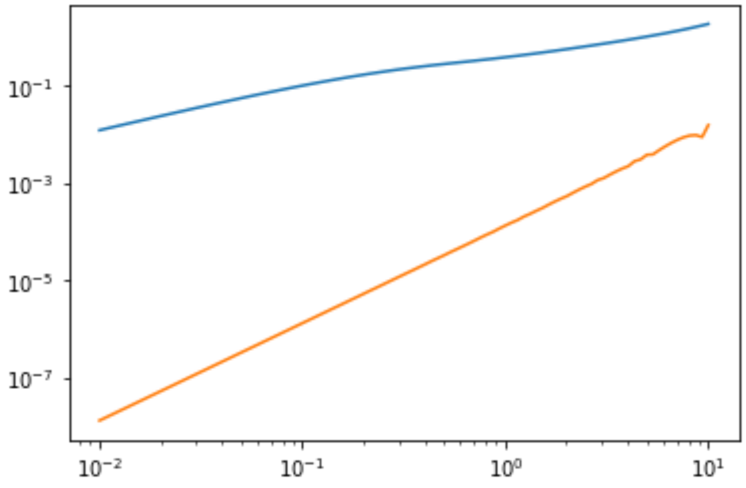
Out[14]:
```
<matplotlib.collections.PathCollection at 0x7fe4d91281d0>
```



In [15]:
```python
dts = np.logspace(-2,1,100)
errors_leapfrog = np.zeros(len(dts))
for i in range(len(dts)):
    errors_leapfrog[i] = measureError(dts[i], leapfrog)
```

```
In [16]: plt.xscale("log")
         plt.yscale("log")
         plt.plot(dts, errors_euler)
         plt.plot(dts, errors_leapfrog)
```

Out[16]: [<matplotlib.lines.Line2D at 0x7fe4d93c1b50>]

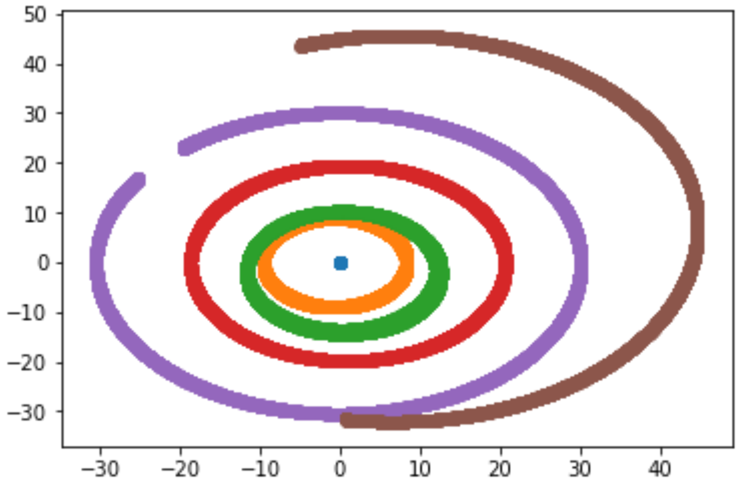


Task 2

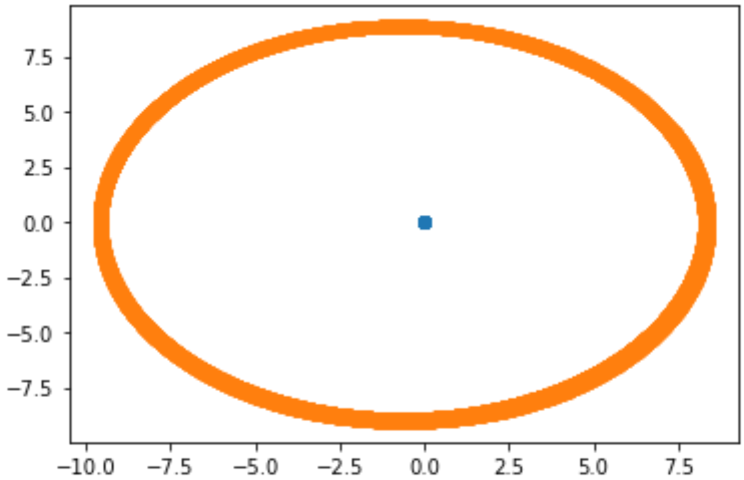
```
In [17]: def whfaststep(sim):
             sim.integrator = "WHfast"
             sim.step()
```

```
In [18]: # Checking out WHfast
         sim.dt = 1e-1
         Nsteps = int(1000/sim.dt)
         xyz = np.zeros((Nsteps, sim.N,3))
         for i in range(Nsteps):
             whfaststep(sim)
             for j in range(sim.N):
                 xyz[i][j] = sim.particles[j].xyz
```

```
In [19]: for i in range(sim.N):
             plt.scatter(xyz[:,i,0], xyz[:,i,1])
```

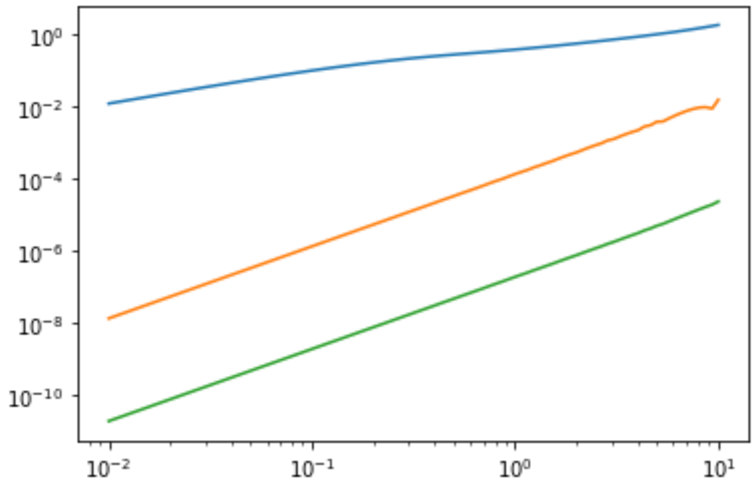


```
In [20]: # Checking out the Jupiter orbit
         plt.scatter(xyz[:,0,0], xyz[:,0,1])
         plt.scatter(xyz[:,1,0], xyz[:,1,1])

Out[20]: <matplotlib.collections.PathCollection at 0x7fe4f0489710>
```



```
In [21]: dts = np.logspace(-2,1,100)
         errors_whfaststep = np.zeros(len(dts))
         for i in range(len(dts)):
             errors_whfaststep[i] = measureError(dts[i], whfaststep)
```

```
In [22]: plt.xscale("log")
         plt.yscale("log")
         plt.plot(dts, errors_euler)
         plt.plot(dts, errors_leapfrog)
         plt.plot(dts, errors_whfaststep)

Out[22]: [<matplotlib.lines.Line2D at 0x7fe4d8e9ea90>]
```



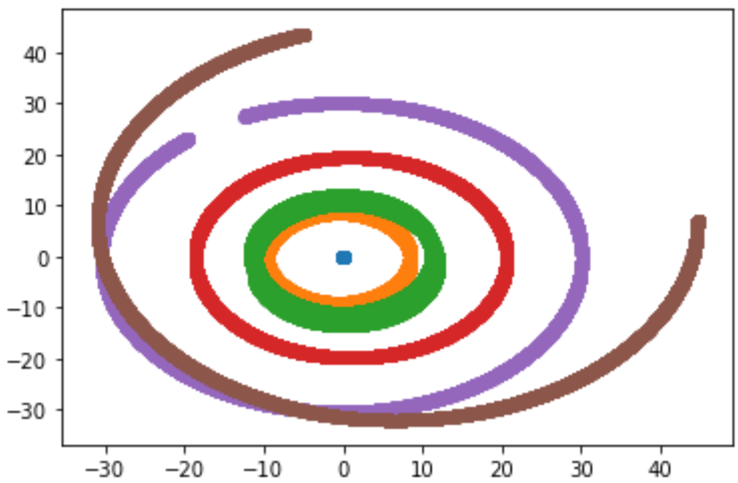
The error in the WHFaststep is significantly lower, and it is of the same order as the leapfrog method, because the slopes are the same.

Bonus Task 1

In [23]:
```
def whckl(sim):
    sim.integrator = "WHCKL"
    sim.step()
```

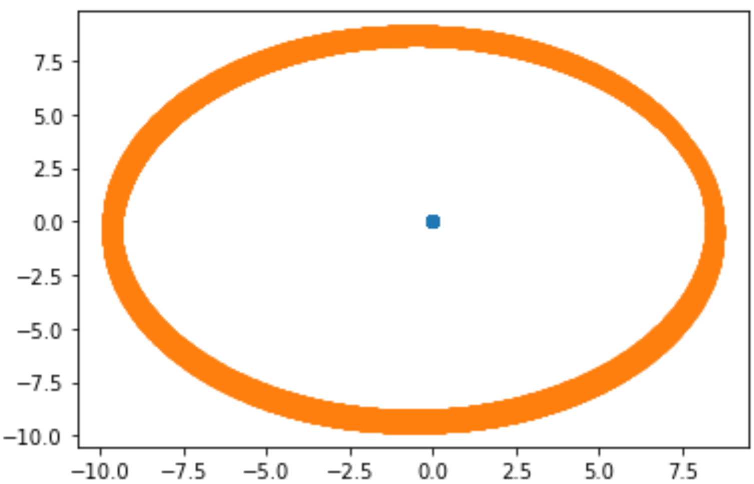
In [24]:
```
# Checking out WHCKL
sim.dt = 1e-1
Nsteps = int(1000/sim.dt)
xyz = np.zeros((Nsteps, sim.N,3))
for i in range(Nsteps):
    whckl(sim)
    for j in range(sim.N):
        xyz[i][j] = sim.particles[j].xyz
```

In [25]:
```
for i in range(sim.N):
    plt.scatter(xyz[:,i,0], xyz[:,i,1])
```



In [26]:
```
# Checking out the Jupiter orbit
plt.scatter(xyz[:,0,0], xyz[:,0,1])
plt.scatter(xyz[:,1,0], xyz[:,1,1])
```

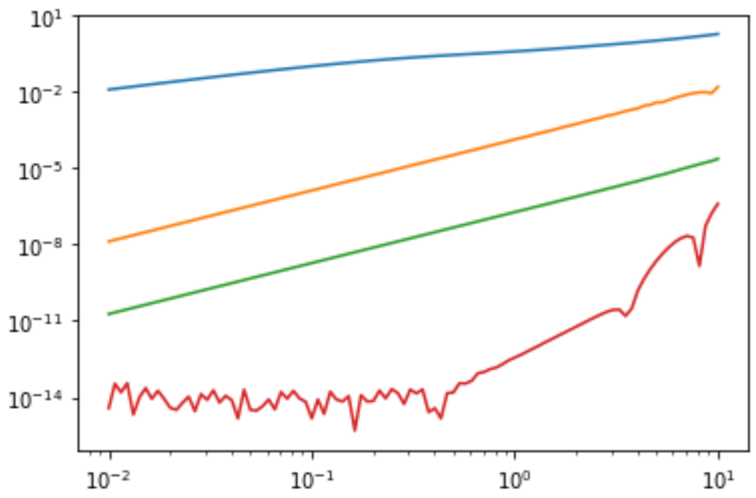
Out[26]:
```
<matplotlib.collections.PathCollection at 0x7fe4d91c41d0>
```



In [27]:
```
dts = np.logspace(-2,1,100)
errors_whckl = np.zeros(len(dts))
for i in range(len(dts)):
    errors_whckl[i] = measureError(dts[i], whckl)
```

In [28]:
```
plt.xscale("log")
plt.yscale("log")
plt.plot(dts, errors_euler)
plt.plot(dts, errors_leapfrog)
plt.plot(dts, errors_whfaststep)
plt.plot(dts, errors_whckl)
```

Out[28]:
```
[<matplotlib.lines.Line2D at 0x7fe4d8da62d0>]
```



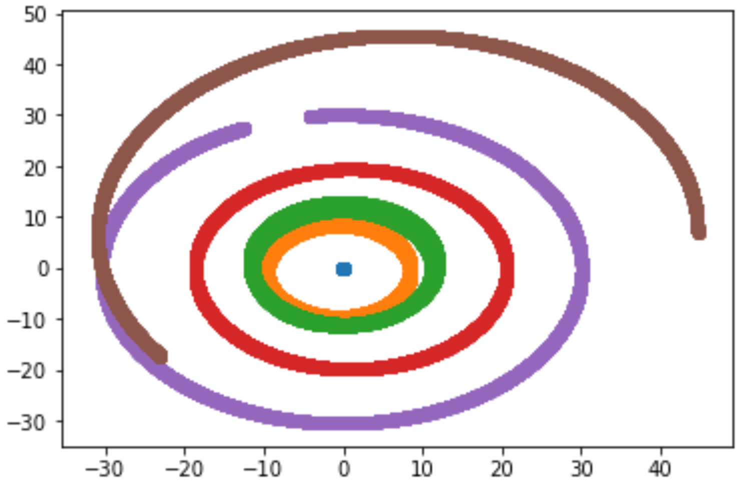
The WHCKL method has a different order as the errors increase significantly as the size of the timestep is increased.

Bonus Task 2

In [29]:
```
def whfaststep_c(sim):
    sim.integrator = "WHfast"
    sim.ri_whfast.corrector = 17
    sim.step()
```

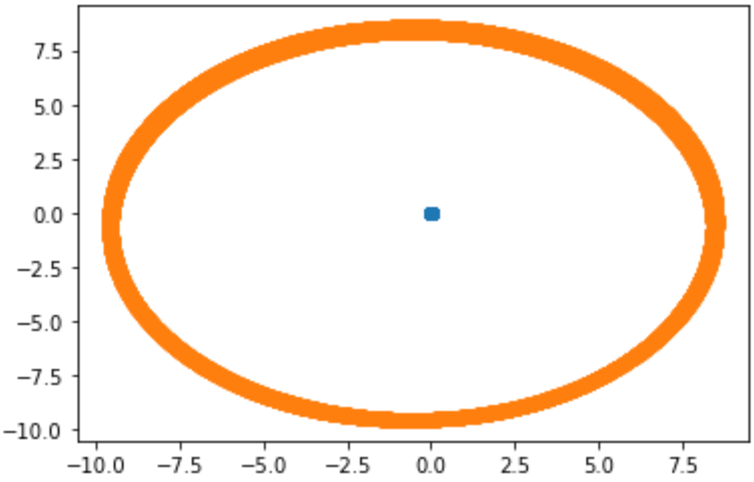
In [30]:
```
# Checking out WH with sympletic correctors
sim.dt = 1e-1
Nsteps = int(1000/sim.dt)
xyz = np.zeros((Nsteps, sim.N,3))
for i in range(Nsteps):
    whfaststep_c(sim)
    for j in range(sim.N):
        xyz[i][j] = sim.particles[j].xyz
```

In [31]:
```
for i in range(sim.N):
    plt.scatter(xyz[:,i,0], xyz[:,i,1])
```



In [32]:
```
# Checking out the Jupiter orbit
plt.scatter(xyz[:,0,0], xyz[:,0,1])
plt.scatter(xyz[:,1,0], xyz[:,1,1])
```

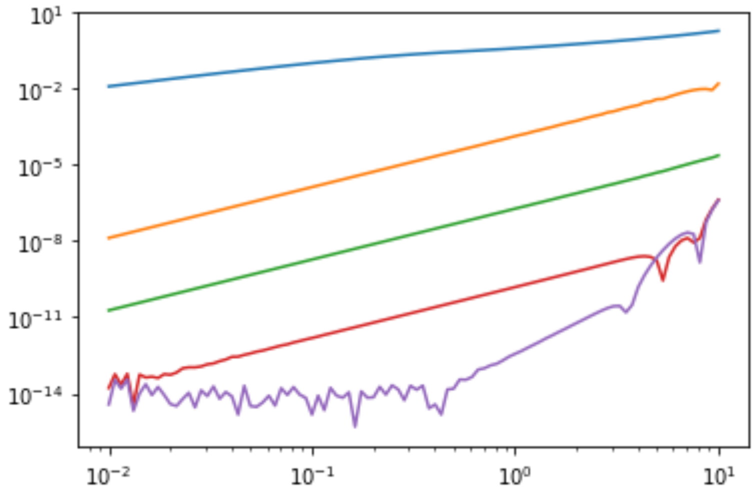
Out[32]: <matplotlib.collections.PathCollection at 0x7fe4d93c8fd0>



In [33]:
```
dts = np.logspace(-2,1,100)
errors_whfaststep_c = np.zeros(len(dts))
for i in range(len(dts)):
    errors_whfaststep_c[i] = measureError(dts[i], whfaststep_c)
```

In [34]:
```
plt.xscale("log")
plt.yscale("log")
plt.plot(dts, errors_euler)
plt.plot(dts, errors_leapfrog)
plt.plot(dts, errors_whfaststep)
plt.plot(dts, errors_whfaststep_c)
plt.plot(dts, errors_whckl)
```

Out[34]:
```
[<matplotlib.lines.Line2D at 0x7fe4d8c533d0>]
```



In [ ]: