Midterm Exam (part 2) - Computational Physics I

Deadline: Tuesday 8 April 2024 (by 19h00)

19/20

Name: Alan Palma Travez

Part 2. (20 points) Two-Body Problem: Black Hole Orbits

This problem consists of developing your own standalone python module to simulate a two-body problem. The module accepts initial parameters from the user and delivers customised simulations of two-body systems where the interaction between them is of gravitational nature, accounting for relativistic effects.

We will assume that the most massive object of mass M is a **black hole** and it is located at the origin of the Cartesian coordinate system (x, y), while the other object is a **planet the size of Earth** with mass $m = m_{\rm earth}$ or smaller, orbiting around the black hole. In this coordinate system, the position of the planet is $\vec{r} = x\hat{x} + y\hat{y}$, which is the vector pointing from the black hole to the planet.

To account for relativistic effects, we need to modify the Newtonian equations of motion. We will use the post-Newtonian approximation, which provides an adequate balance between accuracy and computational efficiency for orbits around the black hole. The **relativistic ODE system** describing the motion of the planet is:

$$\frac{d\vec{r}}{dt} = \vec{v}$$

$$m\frac{d\vec{v}}{dt} = -\frac{G\,m\,M}{r^3}\vec{r}\left(1 + \frac{3\,L^2}{r^2\,c^2}\right)$$

where $L = |\vec{r} \times \vec{v}|$ is the specific angular momentum of the planet, and c is the speed of light. The correction term, $\frac{3\,L^2}{r^2\,c^2}$, accounts for the relativistic precession of the orbit. Note that m cancels out in the above equation.

In addition, **Kepler's third law** for $M \gg m$ states that: $4\,\pi^2\,a^3 \approx G\,M\,T^2$, where a is the semi-major axis of the elliptical relative motion of one object relative to the other and T is the orbital period. Note that in astrophysics we use special units, e.g a is typically in astronomical units ($\rm AU$, where $1\rm AU \equiv$ distance between the Sun and the Earth), T is in yr, and M is in solar masses ($M_{\odot}$, which stands for 1 Solar mass).

At $t = 0$, we will place the planet at **periapsis** (the closest point in its orbit to the black hole). Thus:

$$x_0 = 0$$

$$y_0 = a\,(1 - e)$$

$$v_{x0} = -\sqrt{\frac{G\,M}{a}\,\frac{1+e}{1-e}}$$

$$v_{y0} = 0$$

where e is the eccentricity of the orbit. You can adjust e to control the orbit shape.

The Schwarzschild radius (r_s) of a black hole is the radius of a sphere such that, if all the mass of an object were compressed within that sphere, the escape velocity from the surface of the sphere would equal the speed of light. It is given by:

$$r_s = \frac{2\,G\,M}{c^2}$$

Module design (1 point):

(a) Read the instructions below and clearly outline the directory structure of your module in an **analysis.ipynb** notebook. Follow the class notes on how to structure python packages.

The directory structure is:

```
orbits
├── orbits
│   ├── __init__.py
│   └── orbits.py
└── setup.py
└── test_orbits.py
└── config_generator.py
└── README.md
└── LICENSE.txt
└── config_orbits.ini
└── examples
```

add analysis.ipynb here.

Notes:

Ok, but the INI file would be sufficient.

- The `config_orbits.ini` is the correct format for the INI file to run the simulation and it was generated with `config_generator.py` script.
- `examples` is a directory that contain simulations for Pluto to show the user how the data, images and movie will be generated. ok.

Code development (8 points):

Create a single python script/module **orbits.py**, adequately organised in classes and functions, that:

(b) initialises the two-body problem on a 2D Cartesian grid with an option to save the initial map (if the user wishes to do so). Use the Argparse Library to facilitate user customisation. The grid should be in astronomical units, AU, and a circle denoting the Schwarzschild radius of the black hole should be added. Runs successfully.

(cx2) includes three ODE integration methods: two own-developed methods to carry out the **Trapezoidal Euler** and **Runge-Kutta 3** integrations, and one that uses higher order **SciPy integrators** for initial value problems. Good code structure, fix naming convention.

(dx2) includes a function for the **relativistic** and **classical** slopes given by the above equations of motion. The user should be able to select which slope to use (relativistic or classical).

(e) includes a **run class** to integrate the above system of ODEs for N orbital periods and saves the history of the planet's orbital motion around the black hole into an output file inside an **outputfolder**. **Note:** Both ODEs need to be integrated simultaneously, so you don't need separate functions for the integration of each.

-0.75

Fix pylint complaints. Score is quite low. See below.

(f) includes an **animation class** that reads the planet's orbital history and returns a GIF animation containing the planet position and velocity at different times. The user should be able to turn on a flag at runtime to indicate if the GIF animation is desired. Use the Argparse Library to add this functionality.

Parallelisation worked well -> animate()

(g) accepts as inputs from the user: e, M, a, N, and the numerical method to update the ODE system. Use the Argparse Library to add this functionality. **Note:** Please provide an example of how I should execute your code in the README file.

pylint output:

```
pylint orbits.py
************* Module orbits.orbits
orbits.py:2:18: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:3:31: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:11:21: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:47:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:64:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:78:29: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:89:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:102:55: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:116:29: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:118:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:121:0: C0303: Trailing whitespace (trailing-whitespace)
```

```
orbits.py:143:0: C0301: Line too long (109/100) (line-too-long)
orbits.py:144:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:146:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:150:0: C0301: Line too long (135/100) (line-too-long)
orbits.py:160:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:165:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:170:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:174:0: C0301: Line too long (106/100) (line-too-long)
orbits.py:184:71: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:185:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:188:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:190:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:198:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:199:0: C0301: Line too long (114/100) (line-too-long)
orbits.py:212:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:228:75: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:229:13: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:237:39: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:248:47: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:260:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:272:64: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:280:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:293:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:296:0: C0301: Line too long (104/100) (line-too-long)
orbits.py:320:42: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:334:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:346:33: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:356:60: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:363:60: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:379:68: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:381:17: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:385:0: C0301: Line too long (117/100) (line-too-long)
orbits.py:387:44: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:393:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:401:33: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:420:0: C0301: Line too long (134/100) (line-too-long)
orbits.py:423:0: C0301: Line too long (154/100) (line-too-long)
orbits.py:427:0: C0301: Line too long (102/100) (line-too-long)
orbits.py:440:30: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:441:0: C0301: Line too long (111/100) (line-too-long)
orbits.py:444:0: C0301: Line too long (110/100) (line-too-long)
orbits.py:446:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:450:0: C0301: Line too long (120/100) (line-too-long)
orbits.py:482:55: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:484:18: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:500:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:506:0: C0301: Line too long (105/100) (line-too-long)
orbits.py:520:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:528:0: C0301: Line too long (102/100) (line-too-long)
orbits.py:536:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:539:0: C0301: Line too long (108/100) (line-too-long)
orbits.py:542:0: C0301: Line too long (127/100) (line-too-long)
```

```
orbits.py:548:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:551:0: C0301: Line too long (113/100) (line-too-long)
orbits.py:553:0: C0301: Line too long (134/100) (line-too-long)
orbits.py:556:0: C0301: Line too long (160/100) (line-too-long)
orbits.py:559:103: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:559:0: C0301: Line too long (103/100) (line-too-long)
orbits.py:560:105: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:560:0: C0301: Line too long (105/100) (line-too-long)
orbits.py:564:0: C0301: Line too long (127/100) (line-too-long)
orbits.py:570:30: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:571:0: C0301: Line too long (111/100) (line-too-long)
orbits.py:574:0: C0301: Line too long (110/100) (line-too-long)
orbits.py:576:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:582:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:585:0: C0301: Line too long (108/100) (line-too-long)
orbits.py:601:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:605:0: C0301: Line too long (122/100) (line-too-long)
orbits.py:615:0: C0301: Line too long (112/100) (line-too-long)
orbits.py:619:0: C0301: Line too long (150/100) (line-too-long)
orbits.py:622:105: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:622:0: C0301: Line too long (105/100) (line-too-long)
orbits.py:623:105: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:623:0: C0301: Line too long (105/100) (line-too-long)
orbits.py:625:0: C0301: Line too long (106/100) (line-too-long)
orbits.py:629:31: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:631:0: C0301: Line too long (131/100) (line-too-long)
orbits.py:639:30: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:643:0: C0301: Line too long (111/100) (line-too-long)
orbits.py:646:0: C0301: Line too long (110/100) (line-too-long)
orbits.py:648:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:649:0: C0301: Line too long (107/100) (line-too-long)
orbits.py:652:85: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:653:19: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:654:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:667:60: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:673:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:709:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:710:0: C0301: Line too long (104/100) (line-too-long)
orbits.py:715:62: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:724:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:725:13: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:729:0: C0301: Line too long (132/100) (line-too-long)
orbits.py:736:29: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:737:0: C0301: Line too long (111/100) (line-too-long)
orbits.py:739:29: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:740:0: C0301: Line too long (111/100) (line-too-long)
orbits.py:742:29: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:743:0: C0301: Line too long (111/100) (line-too-long)
orbits.py:745:29: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:746:0: C0301: Line too long (101/100) (line-too-long)
orbits.py:751:81: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:755:0: C0301: Line too long (149/100) (line-too-long)
```

```
orbits.py:756:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:758:123: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:758:0: C0301: Line too long (123/100) (line-too-long)
orbits.py:759:0: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:761:164: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:761:0: C0301: Line too long (164/100) (line-too-long)
orbits.py:764:0: C0301: Line too long (124/100) (line-too-long)
orbits.py:790:24: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:791:0: C0301: Line too long (112/100) (line-too-long)
orbits.py:802:0: C0301: Line too long (112/100) (line-too-long)
orbits.py:804:27: C0303: Trailing whitespace (trailing-whitespace)
orbits.py:805:0: C0301: Line too long (139/100) (line-too-long)
orbits.py:1:0: C0114: Missing module docstring (missing-module-
docstring)
orbits.py:13:0: R0402: Use 'from matplotlib import colors' instead
(consider-using-from-import)
orbits.py:40:8: C0103: Attribute name "G" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:30:26: C0103: Argument name "M" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:43:8: R1720: Unnecessary "else" after "raise", remove the
"else" and de-indent the code inside it (no-else-raise)
orbits.py:30:23: W0613: Unused argument 'a' (unused-argument)
orbits.py:30:26: W0613: Unused argument 'M' (unused-argument)
orbits.py:48:4: C0103: Method name "F_rel" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:61:11: E1101: Instance of 'OrbitsIntegrators' has no
'name_method' member (no-member)
orbits.py:75:8: C0103: Variable name "L" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:79:33: E1101: Instance of 'OrbitsIntegrators' has no 'M'
member; maybe 'G'? (no-member)
orbits.py:85:11: E1101: Instance of 'OrbitsIntegrators' has no
'name_method' member (no-member)
orbits.py:48:20: W0613: Unused argument 't' (unused-argument)
orbits.py:90:4: C0103: Method name "F_classical" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:104:11: E1101: Instance of 'OrbitsIntegrators' has no
'name_method' member (no-member)
orbits.py:117:30: E1101: Instance of 'OrbitsIntegrators' has no 'M'
member; maybe 'G'? (no-member)
orbits.py:123:11: E1101: Instance of 'OrbitsIntegrators' has no
'name_method' member (no-member)
orbits.py:90:26: W0613: Unused argument 't' (unused-argument)
orbits.py:128:4: C0103: Method name "trapezoidal_E" doesn't conform
to snake_case naming style (invalid-name)
orbits.py:128:35: W0621: Redefining name 'sol' from outer scope
(line 807) (redefined-outer-name)
orbits.py:142:37: E1101: Instance of 'OrbitsIntegrators' has no 'F'
member; maybe 'G'? (no-member)
orbits.py:143:38: E1101: Instance of 'OrbitsIntegrators' has no 'F'
member; maybe 'G'? (no-member)
```

```
orbits.py:143:61: E1101: Instance of 'OrbitsIntegrators' has no 'F'
member; maybe 'G'? (no-member)
orbits.py:147:4: C0103: Method name "RK3" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:147:25: W0621: Redefining name 'sol' from outer scope
(line 807) (redefined-outer-name)
orbits.py:162:17: E1101: Instance of 'OrbitsIntegrators' has no 'F'
member; maybe 'G'? (no-member)
orbits.py:163:17: E1101: Instance of 'OrbitsIntegrators' has no 'F'
member; maybe 'G'? (no-member)
orbits.py:164:17: E1101: Instance of 'OrbitsIntegrators' has no 'F'
member; maybe 'G'? (no-member)
orbits.py:171:4: C0103: Method name "DOP853" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:171:28: W0621: Redefining name 'sol' from outer scope
(line 807) (redefined-outer-name)
orbits.py:183:24: E1101: Instance of 'OrbitsIntegrators' has no 'F'
member; maybe 'G'? (no-member)
orbits.py:183:49: E1101: Instance of 'OrbitsIntegrators' has no
'x0' member (no-member)
orbits.py:183:58: E1101: Instance of 'OrbitsIntegrators' has no
'y0' member (no-member)
orbits.py:183:66: E1101: Instance of 'OrbitsIntegrators' has no
'vx0' member (no-member)
orbits.py:183:76: E1101: Instance of 'OrbitsIntegrators' has no
'vy0' member (no-member)
orbits.py:171:21: W0613: Unused argument 'dt' (unused-argument)
orbits.py:171:28: W0613: Unused argument 'sol' (unused-argument)
orbits.py:230:12: C0103: Attribute name "M" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:235:12: C0103: Attribute name "N" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:287:12: C0103: Attribute name "F" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:192:0: R0902: Too many instance attributes (16/7) (too-
many-instance-attributes)
orbits.py:199:4: R0913: Too many arguments (8/5) (too-many-
arguments)
orbits.py:199:4: R0917: Too many positional arguments (8/5) (too-
many-positional-arguments)
orbits.py:217:8: R1720: Unnecessary "else" after "raise", remove
the "else" and de-indent the code inside it (no-else-raise)
orbits.py:222:8: R1720: Unnecessary "else" after "raise", remove
the "else" and de-indent the code inside it (no-else-raise)
orbits.py:227:8: R1720: Unnecessary "else" after "raise", remove
the "else" and de-indent the code inside it (no-else-raise)
orbits.py:232:8: R1720: Unnecessary "else" after "raise", remove
the "else" and de-indent the code inside it (no-else-raise)
orbits.py:294:4: C0103: Method name "solve_ODE" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:315:8: W0621: Redefining name 'time' from outer scope
(line 807) (redefined-outer-name)
```

```
orbits.py:327:8: W0621: Redefining name 'sol' from outer scope
(line 807) (redefined-outer-name)
orbits.py:314:8: C0103: Variable name "T" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:316:8: C0103: Variable name "dT" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:374:12: R1723: Unnecessary "elif" after "break", remove
the leading "el" from "elif" (no-else-break)
orbits.py:374:15: R1714: Consider merging these comparisons with
'in' by using 'val in ('yes', 'Yes')'. Use a set instead if
elements are hashable. (consider-using-in)
orbits.py:377:17: R1714: Consider merging these comparisons with
'in' by using 'val in ('No', 'no')'. Use a set instead if elements
are hashable. (consider-using-in)
orbits.py:432:21: W1309: Using an f-string that does not have any
interpolated variables (f-string-without-interpolation)
orbits.py:412:8: W0612: Unused variable 'fig' (unused-variable)
orbits.py:452:4: E0213: Method 'error_estimate' should have "self"
as first argument (no-self-argument)
orbits.py:478:8: W0612: Unused variable 'n_ref' (unused-variable)
orbits.py:268:12: W0201: Attribute 'method' defined outside
__init__ (attribute-defined-outside-init)
orbits.py:310:8: W0201: Attribute 'name_method' defined outside
__init__ (attribute-defined-outside-init)
orbits.py:330:8: W0201: Attribute 'solution' defined outside
__init__ (attribute-defined-outside-init)
orbits.py:331:8: W0201: Attribute 'time_arr' defined outside
__init__ (attribute-defined-outside-init)
orbits.py:388:8: W0201: Attribute 'filename' defined outside
__init__ (attribute-defined-outside-init)
orbits.py:510:8: C0103: Attribute name "G" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:511:8: C0103: Attribute name "M" doesn't conform to
snake_case naming style (invalid-name)
orbits.py:495:0: R0902: Too many instance attributes (14/7) (too-
many-instance-attributes)
orbits.py:560:19: R1735: Consider using '{"facecolor": 'white',
"alpha": 0.7, "edgecolor": 'black'}' instead of a call to 'dict'.
(use-dict-literal)
orbits.py:547:8: W0612: Unused variable 'fig' (unused-variable)
orbits.py:583:4: R0914: Too many local variables (19/15) (too-many-
locals)
orbits.py:623:19: R1735: Consider using '{"facecolor": 'white',
"alpha": 0.7, "edgecolor": 'black'}' instead of a call to 'dict'.
(use-dict-literal)
orbits.py:652:61: C0209: Formatting a regular string which could be
an f-string (consider-using-f-string)
orbits.py:611:8: W0612: Unused variable 'fig' (unused-variable)
orbits.py:685:15: R1732: Consider using 'with' for resource-
allocating operations (consider-using-with)
orbits.py:680:8: W0201: Attribute 'output_dir' defined outside
__init__ (attribute-defined-outside-init)
```

```
orbits.py:9:0: C0411: standard import "os" should be placed before
third party imports "numpy", "matplotlib.pyplot", "scipy" (...)
"scipy.interpolate.interp1d", "pandas", "scienceplots" (wrong-
import-order)
orbits.py:10:0: C0411: standard import "glob" should be placed
before third party imports "numpy", "matplotlib.pyplot", "scipy"
(...) "scipy.interpolate.interp1d", "pandas", "scienceplots"
(wrong-import-order)
orbits.py:14:0: C0411: standard import "argparse" should be placed
before third party imports "numpy", "matplotlib.pyplot", "scipy"
(...) "PIL.Image", "matplotlib.colormaps", "matplotlib.colors"
(wrong-import-order)
orbits.py:16:0: C0411: standard import "configparser" should be
placed before third party imports "numpy", "matplotlib.pyplot",
"scipy" (...) "matplotlib.colormaps", "matplotlib.colors", "pytest"
(wrong-import-order)
orbits.py:17:0: C0411: standard import "multiprocessing" should be
placed before third party imports "numpy", "matplotlib.pyplot",
"scipy" (...) "matplotlib.colormaps", "matplotlib.colors", "pytest"
(wrong-import-order)
orbits.py:18:0: C0412: Imports from package matplotlib are not
grouped (ungrouped-imports)
orbits.py:4:0: W0611: Unused scipy imported as sp (unused-import)
orbits.py:8:0: W0611: Unused import scienceplots (unused-import)
orbits.py:15:0: W0611: Unused import pytest (unused-import)
orbits.py:18:0: W0611: Unused Circle imported from
matplotlib.patches (unused-import)
orbits.py:18:0: W0611: Unused Patch imported from
matplotlib.patches (unused-import)

------------------------------------------------------------------
Your code has been rated at 0.87/10 (previous run: 0.87/10, +0.00)
```

Unit tests (2 points):

(h) Create a **test_orbits.py** file containing `pytest` unit tests. Provide 3 examples of pytest unit tests that could verify: a) correct input values from the user, b) handling of invalid input methods, and c) whether different inputs actually lead to different outputs.

Fix pylint complaints. See below.

Functions are missing return lines.
a) It works. All methods return Value Error.
b) Correct ValueError for undefined method.
c) Good.

-0.25

pylint output:

```
pylint test_orbits.py
************* Module test_orbits
test_orbits.py:9:0: C0301: Line too long (106/100) (line-too-long)
test_orbits.py:16:11: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:18:82: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:20:82: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:30:26: C0303: Trailing whitespace (trailing-
```

```
whitespace)
test_orbits.py:43:36: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:51:70: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:53:0: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:54:0: C0301: Line too long (111/100) (line-too-long)
test_orbits.py:55:71: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:57:0: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:59:71: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:61:0: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:63:71: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:67:71: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:68:56: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:69:0: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:77:53: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:82:0: C0301: Line too long (117/100) (line-too-long)
test_orbits.py:83:0: C0301: Line too long (117/100) (line-too-long)
test_orbits.py:85:49: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:93:52: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:95:81: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:112:0: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:127:0: C0301: Line too long (120/100) (line-too-
long)
test_orbits.py:134:0: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:136:29: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:137:29: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:142:29: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:143:29: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:148:0: C0301: Line too long (118/100) (line-too-
long)
test_orbits.py:149:0: C0301: Line too long (128/100) (line-too-
long)
```

```
test_orbits.py:151:0: C0303: Trailing whitespace (trailing-
whitespace)
test_orbits.py:153:0: C0301: Line too long (130/100) (line-too-
long)
test_orbits.py:155:0: C0301: Line too long (121/100) (line-too-
long)
test_orbits.py:156:0: C0305: Trailing newlines (trailing-newlines)
test_orbits.py:1:0: C0114: Missing module docstring (missing-
module-docstring)
test_orbits.py:23:8: W0107: Unnecessary pass statement
(unnecessary-pass)
test_orbits.py:34:8: W0107: Unnecessary pass statement
(unnecessary-pass)
test_orbits.py:44:8: C0103: Variable name "M" doesn't conform to
snake_case naming style (invalid-name)
test_orbits.py:47:8: C0103: Variable name "N" doesn't conform to
snake_case naming style (invalid-name)
test_orbits.py:3:0: C0411: third party import "pytest" should be
placed before first party import "orbits.orbits"  (wrong-import-
order)
test_orbits.py:4:0: C0411: third party import "numpy" should be
placed before first party import "orbits.orbits"  (wrong-import-
order)

------------------------------------------------------------------
Your code has been rated at 3.58/10 (previous run: 3.58/10, +0.00)
```

pytest output:

```
pytest test_orbits.py
==================================================================
test session starts
===================================================================
platform darwin -- Python 3.9.18, pytest-8.3.4, pluggy-1.5.0
rootdir:
/Users/wbandabarragan/Library/CloudStorage/Dropbox/Yachay_Tech/Semestre
plugins: anyio-4.7.0
collected 5 items

test_orbits.py .....
[100%]

===================================================================
5 passed in 0.85s
====================================================================
```

Relativistic versus classical mechanics (3 points):

Within your python notebook **analysis.ipynb**, add the following:

(i) Use your module/script to run and show two simulations: one relativistic and one classical for this set of initial conditions. It may be helpful to compare the orbital history in a single plot.

Parameter	Description	Units
e	Eccentricity of the orbit	0
M	Mass of the central black hole	$5 \times 10^6\ \mathrm{M}_\odot$
a	Semi-major axis of the orbit	$1\ \mathrm{AU}$
N	Number of orbital periods to simulate	2
Method	Numerical method for ODE integration	RK3

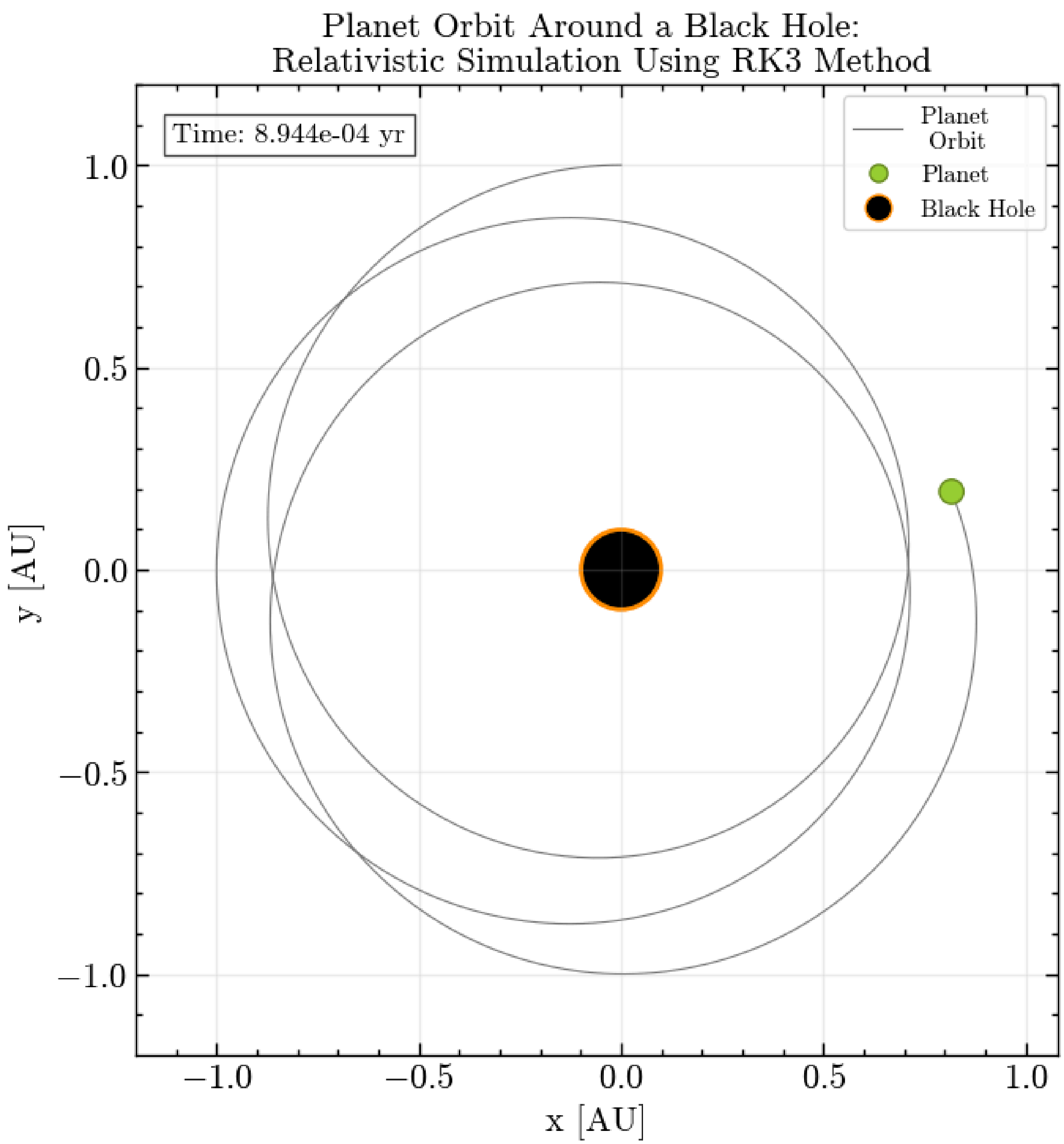
In [25]:
```python
# Import our package after installing
from orbits.orbits import RunOrbits
from orbits.orbits import AnimateOrbits

# Third party libraries
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle, Patch
from matplotlib.lines import Line2D
```

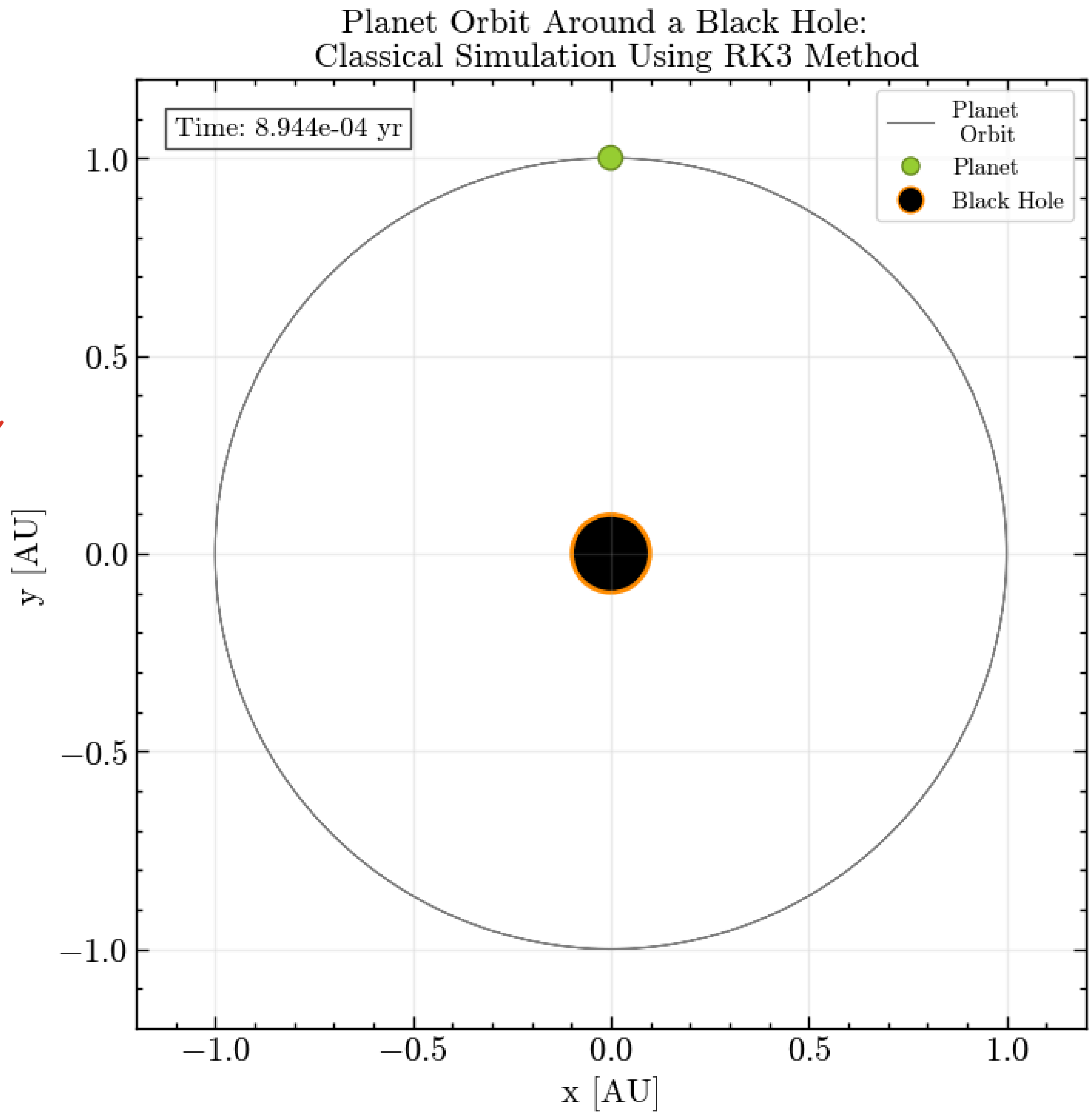
In [26]:
```python
# Instanciate the classes
orbit_re = RunOrbits(M = 5.e6, e = 0.0, a = 1.0, N = 2.0, n = 500,\
                     simulation = "Relativistic")
orbit_cls = RunOrbits(M = 5.e6, e = 0.0, a = 1.0, N = 2.0, n = 500,\
                     simulation = "Classical")

# Select the method and solve the ODE
time_re, sol_re = orbit_re.solve_ODE("RK3")
time_cls, sol_cls = orbit_cls.solve_ODE("RK3")

# Save both simulations
fname_re = orbit_re.save_solution("01sim")
fname_cls = orbit_cls.save_solution("02sim")
```

```
Directory 'outputfolder' has been created.
Solution have been saved in 'outputfolder' as: 01sim-Rel.out
Directory 'outputfolder' already exists.
Solution have been saved in 'outputfolder' as: 02sim-Cla.out
```

In [27]:

```
# Use module for ploting the orbital history

plot_orbit_re = AnimateOrbits(orbit_re)
plot_orbit_re.plot_simulation()

plot_orbit_cls = AnimateOrbits(orbit_cls)
plot_orbit_cls.plot_simulation()
```

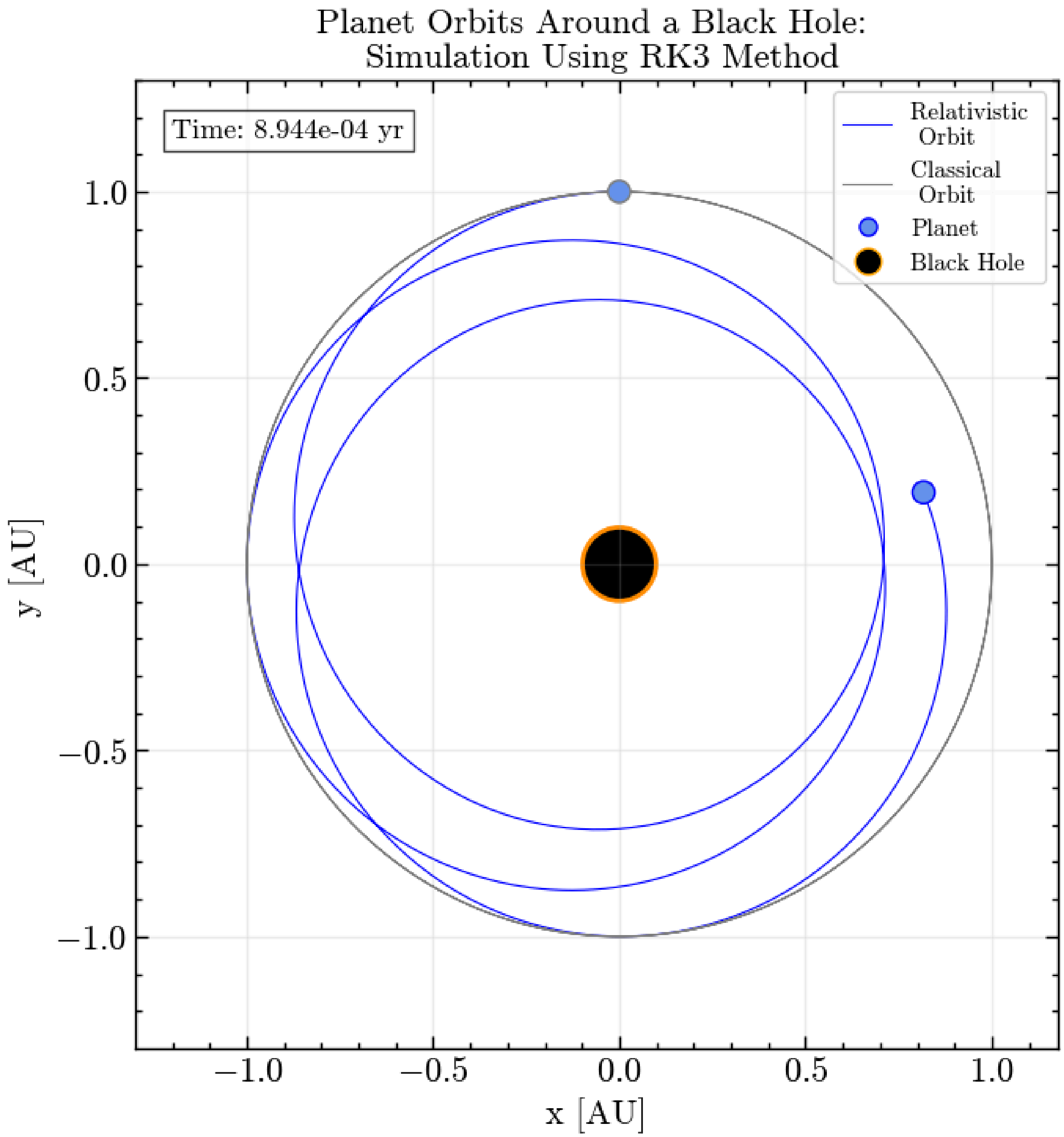




In [28]:

```
# Ploting the orbits together

# Plot the orbital history together
fig, ax = plt.subplots(figsize = (10, 8))

# Planet orbit: Relativistic
orb1, = ax.plot(sol_re[:,0, 0], sol_re[:,0, 1], color = "blue", linestyle =
# Planet orbit: Classical
orb2, = ax.plot(sol_cls[:,0, 0], sol_cls[:,0, 1], color = "gray", linestyle

# Black hole
black_hole = plt.Circle((0 ,0), orbit_re.rs, facecolor = "black", edgecolor
ax.add_patch(black_hole)
# Planet: Earth
planet = plt.Circle((sol_re[:,0,0][-1], sol_re[:,0,1][-1]), 0.03 , facecolor
ax.add_patch(planet)

# Planet: Earth
```

```
planet = plt.Circle((sol_cls[:,0,0][-1], sol_cls[:,0,1][-1]), 0.03 , facecol
ax.add_patch(planet)

# Time stamp
ax.text(0.04, 0.96, f"Time: {time_cls[-1]:.3e} yr", ha ='left', va = 'top',
    bbox = dict(facecolor = 'white', alpha = 0.7, edgecolor = 'black'), tran

ax.set_xlabel("x [AU]")
ax.set_ylabel("y [AU]")
ax.set_title(f"Planet Orbits Around a Black Hole: \n Simulation Using {orbit

ax.grid(alpha = 0.2)
ax.set_xlim(np.min(sol_re[:,0,0]) - 0.3, np.max(sol_re[:,0,0]) + 0.3)
ax.set_ylim(np.min(sol_re[:,0,1])-0.3, np.max(sol_re[:,0,1])+0.3)


# Create custom legend
legend_e = Line2D([0], [0], marker = "o", color = "w", label= "Planet", mark
                   markeredgecolor = "blue", markersize = 8)

legend_bh = Line2D([0], [0], marker = "o", color = "w", label= "Black Hole",
                   markeredgecolor = "orange", markersize = 12)

ax.set_aspect('equal')  # Ensures circles stay circular
ax.legend(frameon = True, handles=[orb1, orb2, legend_e, legend_bh], fontsiz

plt.show()
```



(j) Use the orbital history of both simulations to design a method that quantifies their differences and evaluates the importance of using the relativistic approach for massive objects. Do we need to worry about the relativistic corrections if we replace the black hole with our Sun?

For comparing the differences in orbits trajectories, let's use the distance between orbits as a metric:

$$D(t) = \sqrt{(x_c - x_r)^2 + (y_c - y_r)^2},$$

where subindex c stands for classical and subindex r for relativistic.

In [29]:
```
def metric(xc, yc, xr, yr):
    """
```

```
    Compute error metrics between a computed trajectory (xc, yc) and a refer

    Inputs:
        xc (array): x-coordinates of the approximate solution.
        yc (array): y-coordinates of the approximate solution.
        xr (array): x-coordinates of the reference solution.
        yr (array): y-coordinates of the reference solution.

    Output:
        diff (array): Point-wise Euclidean distance (error) between the comp
        mean_diff (float): Mean of the point-wise differences, representing
        max_diff (float): Maximum of the point-wise differences, representir
    """
    # Compute the difference
    diff = np.sqrt((xc - xr)**2 + (yc - yr)**2)

    # Get the average
    mean_diff = np.mean(diff)

    # Maximum difference
    max_diff = np.max(diff)

    return diff, mean_diff, max_diff
```

In [30]:
```
diff_1, mean_d1, max_d1 = metric(sol_cls[:,0, 0], sol_cls[:,0, 1], sol_re[:,

print(f"The mean difference is: {mean_d1:.3f} AU")
print(f"The max difference is: {max_d1:.3f} AU")
```

```
The mean difference is: 1.281 AU
The max difference is: 1.852 AU
```

To have a point of comparison let's compute the solution for an object that has the mass on the sun.

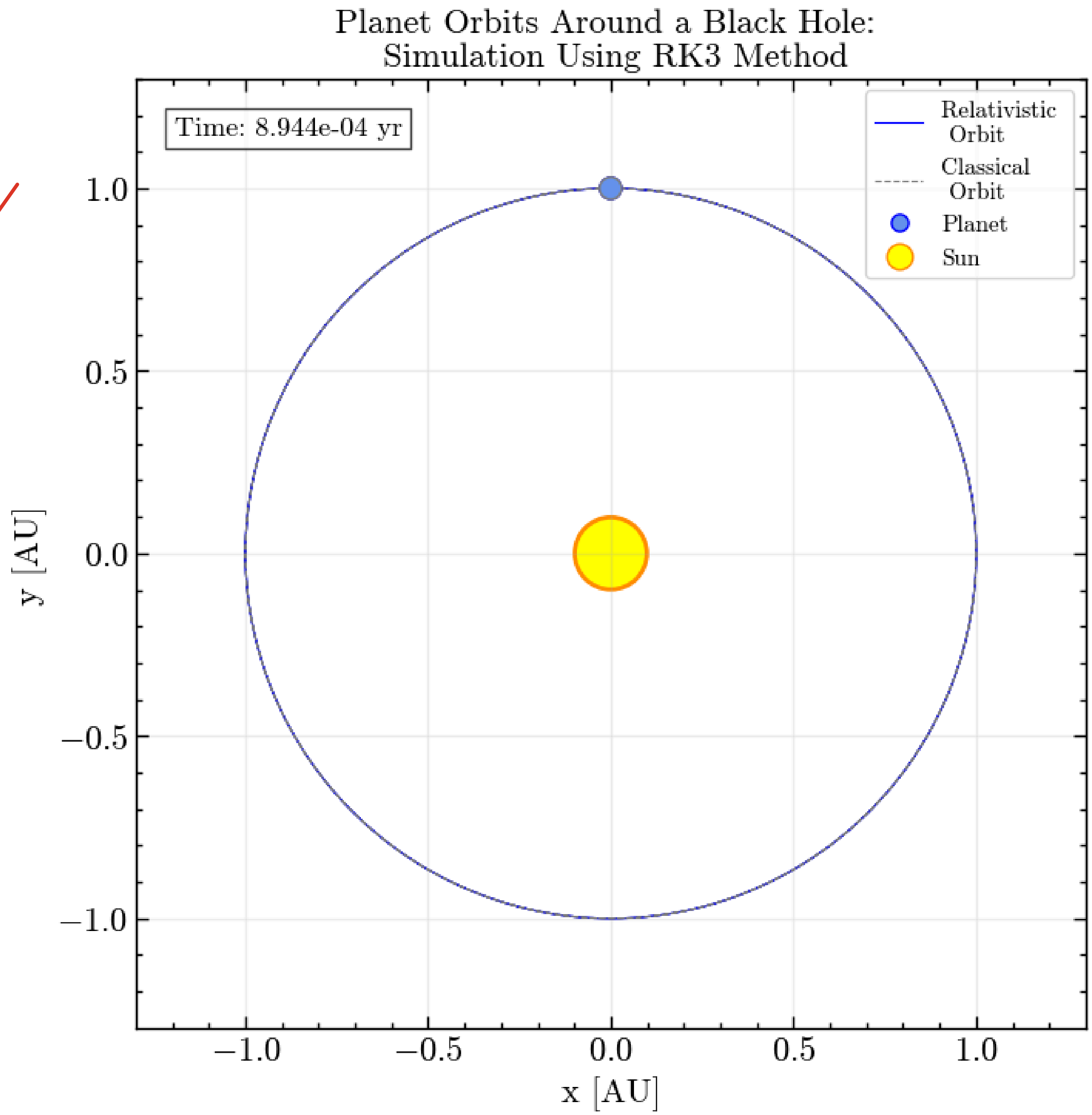
Sun

In [31]:
```
# Instanciate the classes
orbit_re_sun = RunOrbits(M = 1., e = 0.0, a = 1.0, N = 2.0, n = 500,\
                   simulation = "Relativistic")
orbit_cls_sun = RunOrbits(M = 1., e = 0.0, a = 1.0, N = 2.0, n = 500,\
                   simulation = "Classical")

# Select the method and solve the ODE
time_re_sun, sol_re_sun = orbit_re_sun.solve_ODE("RK3")
time_cls_sun, sol_cls_sun = orbit_cls_sun.solve_ODE("RK3")

# Save both simulations
fname_re = orbit_re_sun.save_solution("01sim_sun")
fname_cls = orbit_cls_sun.save_solution("02sim_sun")
```

```
Directory 'outputfolder' already exists.
Solution have been saved in 'outputfolder' as: 01sim_sun-Rel.out
Directory 'outputfolder' already exists.
Solution have been saved in 'outputfolder' as: 02sim_sun-Cla.out
```

In [32]:
```
# Ploting the orbits together
```

```
# Plot the orbital history together
fig, ax = plt.subplots(figsize = (10, 8))

# Planet orbit: Relativistic
orb1, = ax.plot(sol_re_sun[:,0, 0], sol_re_sun[:,0, 1], color = "blue", line
# Planet orbit: Classical
orb2, = ax.plot(sol_cls_sun[:,0, 0], sol_cls_sun[:,0, 1], color = "gray", li

# Sun
sun = plt.Circle((0 ,0), orbit_re.rs, facecolor = "yellow", edgecolor = "dar
ax.add_patch(sun)
# Planet: Earth
planet = plt.Circle((sol_re_sun[:,0,0][-1], sol_re_sun[:,0,1][-1]), 0.03 , f
ax.add_patch(planet)

# Planet: Earth
planet = plt.Circle((sol_cls_sun[:,0,0][-1], sol_cls_sun[:,0,1][-1]), 0.03 ,
ax.add_patch(planet)

# Time stamp
ax.text(0.04, 0.96, f"Time: {time_cls[-1]:.3e} yr", ha ='left', va = 'top',
    bbox = dict(facecolor = 'white', alpha = 0.7, edgecolor = 'black'), tran

ax.set_xlabel("x [AU]")
ax.set_ylabel("y [AU]")
ax.set_title(f"Planet Orbits Around a Black Hole: \n Simulation Using {orbit

ax.grid(alpha = 0.2)
ax.set_xlim(np.min(sol_re_sun[:,0,0]) - 0.3, np.max(sol_re_sun[:,0,0]) + 0.3
ax.set_ylim(np.min(sol_re_sun[:,0,1])-0.3, np.max(sol_re_sun[:,0,1])+0.3)


# Create custom legend
legend_e = Line2D([0], [0], marker = "o", color = "w", label= "Planet", mark
                   markeredgecolor = "blue", markersize = 8)

legend_bh = Line2D([0], [0], marker = "o", color = "w", label= "Sun", marker
                   markeredgecolor = "darkorange", markersize = 12)

ax.set_aspect('equal')  # Ensures circles stay circular
ax.legend(frameon = True, handles=[orb1, orb2, legend_e, legend_bh], fontsiz

plt.show()
```



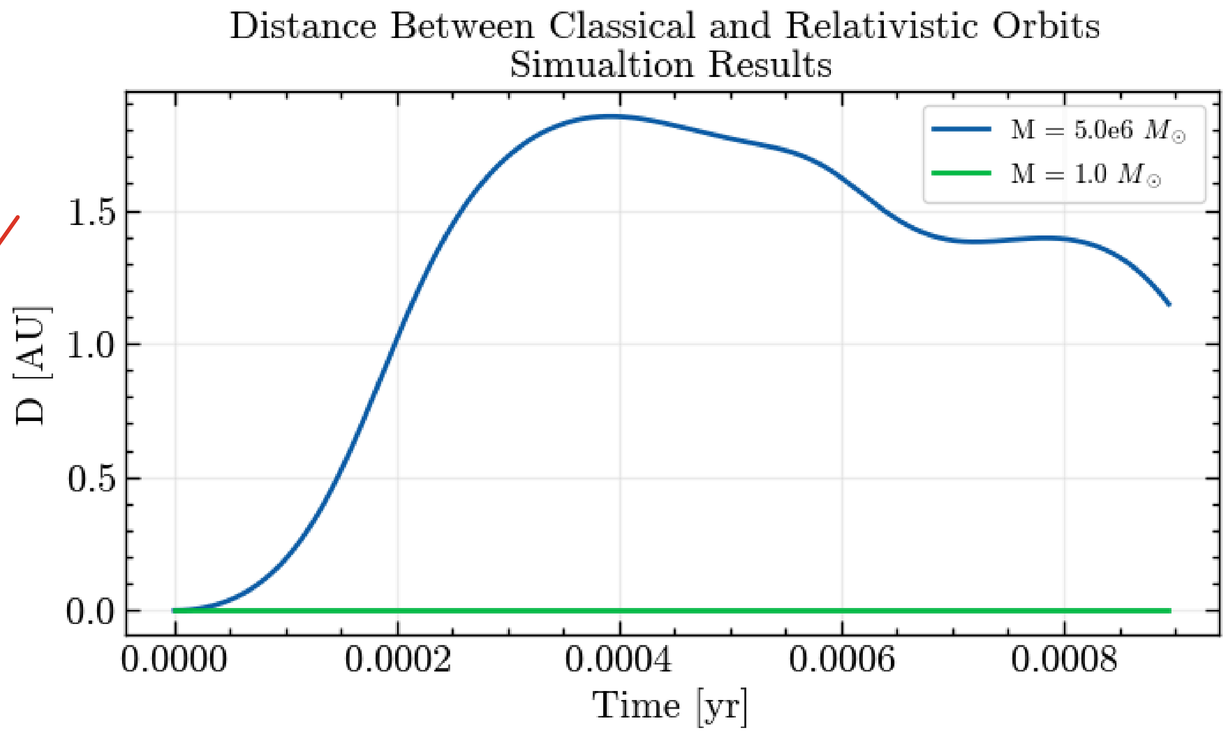
Using the method created before let's quantify the differences.

In [33]:
```
diff_2, mean_d2, max_d2 = metric(sol_cls_sun[:,0, 0], sol_cls_sun[:,0, 1], s

print(f"The mean difference is: {mean_d2:.3e} AU")
print(f"The max difference is: {max_d2:.3e} AU")
```

```
The mean difference is: 3.760e-07 AU
The max difference is: 7.453e-07 AU
```

In [34]:
```
plt.figure(figsize=(8, 4))

plt.plot(time_cls, diff_1, label = r"M = 5.0e6 $M_\odot$ ")
plt.plot(time_cls, diff_2, label = r"M = 1.0 $M_\odot$")

plt.xlabel("Time [yr]")
plt.ylabel("D [AU]")
plt.title("Distance Between Classical and Relativistic Orbits \n" \
```

```
              "Simualtion Results")

plt.grid(alpha = 0.2)
plt.legend(frameon = True, fontsize = 11)

plt.show()
```



It was found that the mean difference between classical and relativistic orbits is 1.281 AU for the black hole case. However, in the simulation considering a solar mass, the difference is only 3.760×10^{-7} AU. Therefore, relativistic corrections are crucial for black holes but likely negligible for objects like the Sun, which have similar mass and orbits. In the figure above, a graphical comparison of the differences between the orbits is shown more clearly.

The role of eccentricity (3 points):

(k) Use your module/script to run and show three relativistic simulations for objects with different eccentricities, e, and assuming the same M, a, N as above. It may be helpful to compare the orbital history for all values of e in a single plot throughout time.

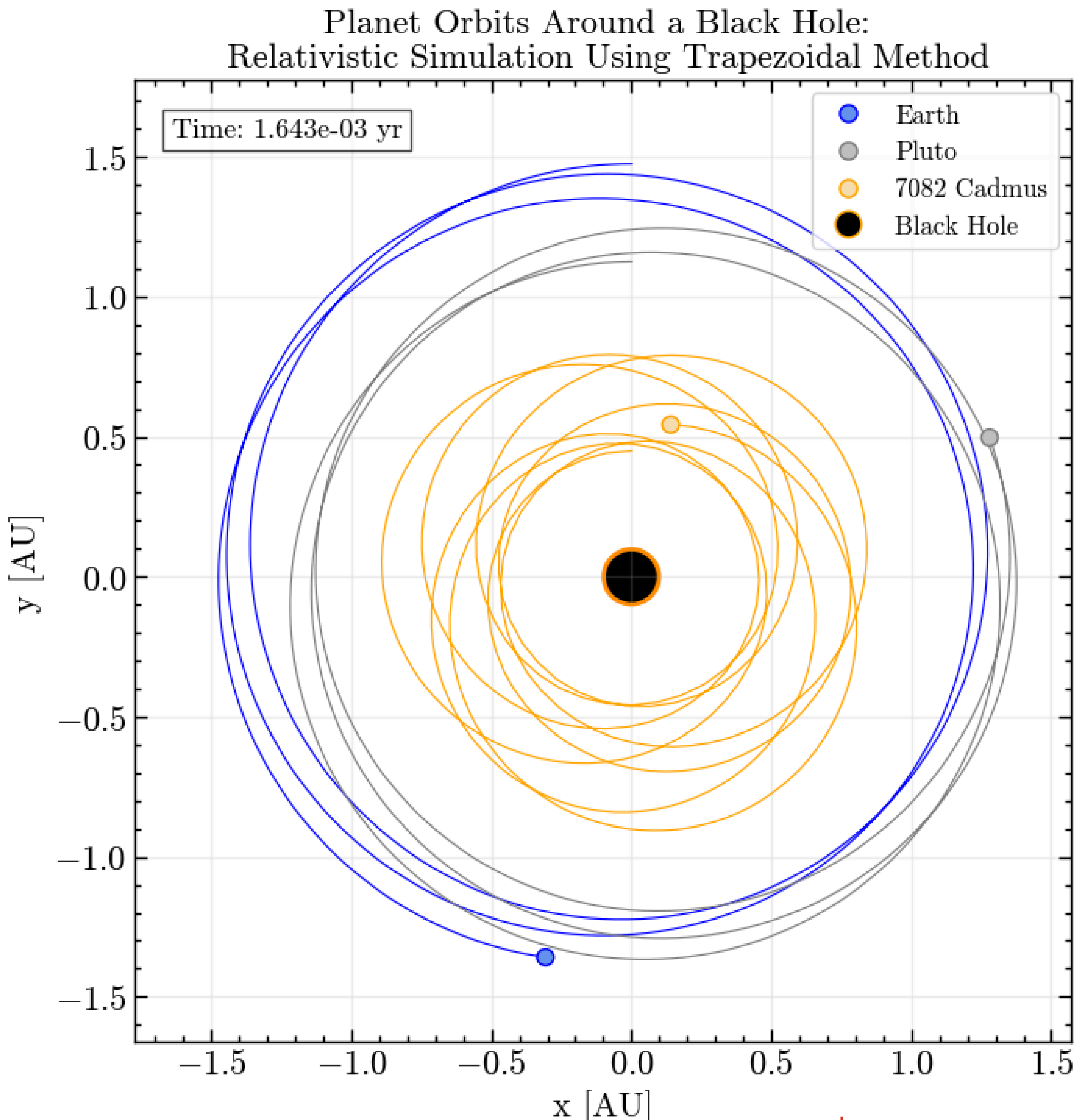
Object	Eccentricity (e)	Integration Method
Earth	0.01671	Trapezoidal
Pluto	0.25	Trapezoidal
7092 Cadmus	0.70	Trapezoidal

(l) Describe the differences in the orbits of the above objects. What happens to objects with high eccentricities?

In [35]:
```
# Instanciate the classes
orbit_e = RunOrbits(M = 5.e6, e = 0.01671, a = 1.5, N = 2.0, n = 500,\
                   simulation = "Relativistic")
orbit_p = RunOrbits(M = 5.e6, e = 0.25, a = 1.5, N = 2.0, n = 500,\
                   simulation = "Relativistic")
orbit_cad = RunOrbits(M = 5.e6, e = 0.70, a = 1.5, N = 2.0, n = 500,\
                   simulation = "Relativistic")

# Select the method and solve the ODE
time, sol_e = orbit_e.solve_ODE("Trapezoidal")
_, sol_p = orbit_p.solve_ODE("Trapezoidal")
_, sol_cad = orbit_cad.solve_ODE("Trapezoidal")
```

In [36]:
```
# Plot the orbital history together
fig, ax = plt.subplots(figsize = (10, 8))

# Planet orbit: Earth
ax.plot(sol_e[:,0, 0], sol_e[:,0, 1], color = "blue", linestyle = "-", linew
# Planet orbit: Pluto
ax.plot(sol_p[:,0, 0], sol_p[:,0, 1], color = "gray", linestyle = "-", linew
# Planet orbit: 7082 Cadmus
ax.plot(sol_cad[:,0, 0], sol_cad[:,0, 1], color = "orange", linestyle = "-",
# Black hole
black_hole = plt.Circle((0 ,0), orbit_e.rs, facecolor = "black", edgecolor =
ax.add_patch(black_hole)
# Planet: Earth
planet = plt.Circle((sol_e[:,0,0][-1], sol_e[:,0,1][-1]), 0.03 , facecolor =
ax.add_patch(planet)
# Planet: Pluto
planet = plt.Circle((sol_p[:,0,0][-1], sol_p[:,0,1][-1]), 0.03 , facecolor =
ax.add_patch(planet)
# Planet: 7082 Cadmus
planet = plt.Circle((sol_cad[:,0,0][-1], sol_cad[:,0,1][-1]), 0.03 , facecol
ax.add_patch(planet)
# Time stamp
ax.text(0.04, 0.96, f"Time: {time[-1]:.3e} yr", ha ='left', va = 'top', font
```

```
    bbox = dict(facecolor = 'white', alpha = 0.7, edgecolor = 'black'), tran

ax.set_xlabel("x [AU]")
ax.set_ylabel("y [AU]")
ax.set_title(f"Planet Orbits Around a Black Hole: \n {orbit_e.simulation_typ

ax.grid(alpha = 0.2)
ax.set_xlim(np.min(sol_e[:,0,0]) - 0.3, np.max(sol_e[:,0,0]) + 0.3)
ax.set_ylim(np.min(sol_e[:,0,1])-0.3, np.max(sol_e[:,0,1])+0.3)


# Create custom legend
legend_e = Line2D([0], [0], marker = "o", color = "w", label= "Earth", marke
                   markeredgecolor = "blue", markersize = 8)
legend_p = Line2D([0], [0], marker = "o", color = "w", label= "Pluto", marke
                   markeredgecolor = "gray", markersize = 8)
legend_cad = Line2D([0], [0], marker = "o", color = "w", label= "7082 Cadmus
                   markeredgecolor = "orange", markersize = 8)
legend_bh = Line2D([0], [0], marker = "o", color = "w", label= "Black Hole",
                   markeredgecolor = "orange", markersize = 12)

ax.set_aspect('equal')  # Ensures circles stay circular
ax.legend(frameon = True, handles=[legend_e, legend_p, legend_cad, legend_bh

plt.show()
```



ok.

I just slightly increased the semi-major axis to better visualize the orbits. The eccentricity makes the orbits more elongated,the higher the eccentricity, the more oval-shaped the orbits become.

Numerical convergence (3 points):

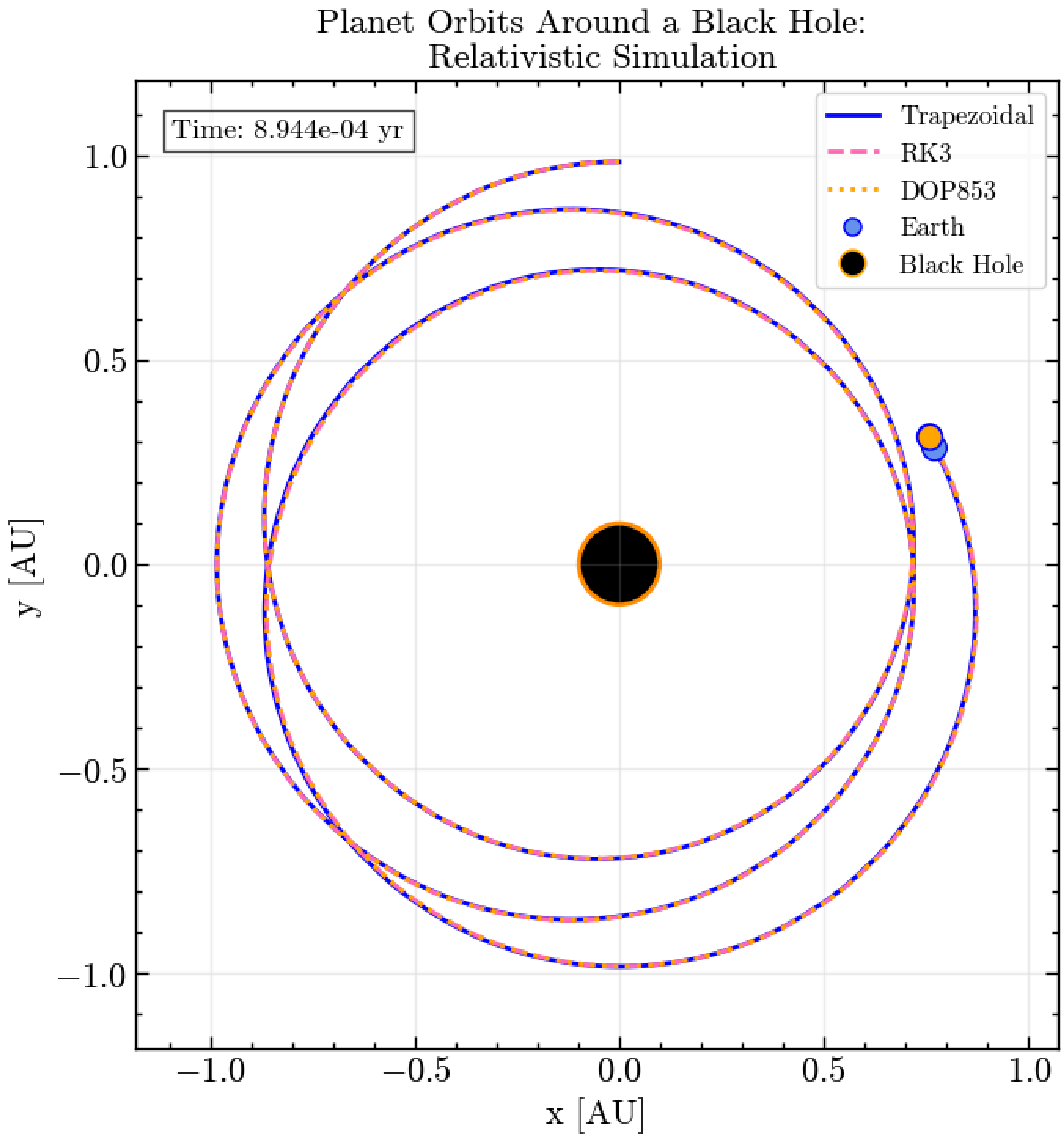
(m) Use your script to generate additional simulations with the same initial conditions as before, but only for $e = 0.01671$ (Earth's eccentricity) with RK3, the Trapezoidal method and the higher-order SciPy integrator. Compare the orbital history for all methods in a single plot throughout time.

In [37]:

```
# Instanciate a class for computing the orbits
orbit_comp = RunOrbits(M = 5.e6, e = 0.01671, a = 1.0, N = 2.0, n = 600,\
                       simulation = "Relativistic")
```

```
# Select the method and solve the ODE
time_tra, sol_tra = orbit_comp.solve_ODE("Trapezoidal")
_, sol_RK3 = orbit_comp.solve_ODE("RK3")
_, sol_DOP853 = orbit_comp.solve_ODE("DOP853")
```

In [38]:
```
# Plot the orbital history together
fig, ax = plt.subplots(figsize = (10, 8))

# Planet orbit: Trapezoidal
orb_tra, = ax.plot(sol_tra[:,0, 0], sol_tra[:,0, 1], color = "blue", linesty
# Planet orbit: RK3
orb_RK3, = ax.plot(sol_RK3[:,0, 0], sol_RK3[:,0, 1], color = "hotpink", line
# Planet orbit: DOP853
orb_DOP583, = ax.plot(sol_DOP853[:,0, 0], sol_DOP853[:,0, 1], color = "orang
# Black hole
black_hole = plt.Circle((0 ,0), orbit_comp.rs, facecolor = "black", edgecolo
ax.add_patch(black_hole)
# Planet: Earth
planet = plt.Circle((sol_tra[:,0,0][-1], sol_tra[:,0,1][-1]), 0.03 , facecol
ax.add_patch(planet)
# Planet: Earth
planet = plt.Circle((sol_RK3[:,0,0][-1], sol_RK3[:,0,1][-1]), 0.03 , facecol
ax.add_patch(planet)
# Planet: EarthDOP853
planet = plt.Circle((sol_DOP853[:,0,0][-1], sol_DOP853[:,0,1][-1]), 0.03 , f
ax.add_patch(planet)
# Time stamp
ax.text(0.04, 0.96, f"Time: {time_tra[-1]:.3e} yr", ha ='left', va = 'top',
    bbox = dict(facecolor = 'white', alpha = 0.7, edgecolor = 'black'), tran

ax.set_xlabel("x [AU]")
ax.set_ylabel("y [AU]")
ax.set_title(f"Planet Orbits Around a Black Hole: \n {orbit_comp.simulation_

ax.grid(alpha = 0.2)
ax.set_xlim(np.min(sol_RK3[:,0,0]) - 0.2, np.max(sol_RK3[:,0,0]) + 0.2)
ax.set_ylim(np.min(sol_RK3[:,0,1])-0.2, np.max(sol_RK3[:,0,1]) + 0.2)



# Create custom legend
legend_e = Line2D([0], [0], marker = "o", color = "w", label= "Earth", marke
                   markeredgecolor = "blue", markersize = 8)
legend_bh = Line2D([0], [0], marker = "o", color = "w", label= "Black Hole",
                   markeredgecolor = "orange", markersize = 12)

ax.set_aspect('equal')  # Ensures circles stay circular
ax.legend(frameon = True, handles=[orb_tra, orb_RK3, orb_DOP583, legend_e, l

plt.show()
```



As seen, the three methods gave very similar results, although there are still slight differences between them. The point of reference should be the DOP853 method, as it is the higher-order method. It is also observed that, depending on the accuracy of each method, the final position of the planet within the period varies noticeably. The final position using the SciPy integrator is slightly ahead of the other two.

(n) Measure convergence of the simulations with RK3 and Trapezoidal method for $e = 0.01671$ by integrating at a number of different time steps. To analyse convergence, you need to define some measure for the error with respect to the higher order method, and then plot it against different time steps for both methods. Thus, you may add additional functions for this to your code in **orbits.py**.

I am going to use interpolation for adjusting the shape arrays of the reference solution to the aproximate solution. The error will ve computed as the mean of distance between the two solutions over the number of time steps: Ok.

$$\langle D(t)/n \rangle,$$

error_estimate() added in orbits.py

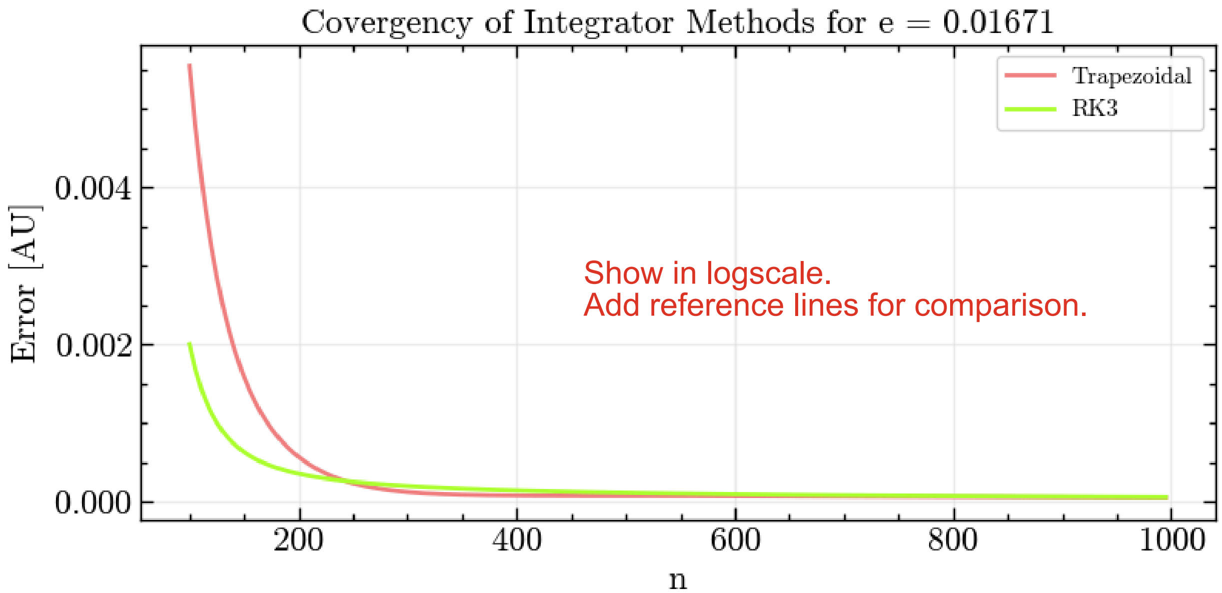
where $D(t)$ is the distance over time and n the number of time steps.

In [39]:
```
# Define the time steps that will be used
n_min = 100
n_max = 1000

# Compute exact solution (reference)

# Instanciate a class for computing the orbits
orbit_comp = RunOrbits(M = 5.e6, e = 0.01671, a = 1.0, N = 2.0, n = n_max,\
                       simulation = "Relativistic")

# Select the method and solve the ODE
time_ref, sol_ref = orbit_re.solve_ODE("DOP853")
```

In [40]:
```
# Create a step array for interating
n_arr = np.arange(n_min, n_max, 5)

# Empty list to save error
error_tra = []
error_RK3 = []

for n_i in n_arr:
    # Instanciate a class for computing the orbits
    orbit_err = RunOrbits(M = 5.e6, e = 0.01671, a = 1.0, N = 2.0, n = n_i,\
                          simulation = "Relativistic")

    # Select the method and solve the ODE
    time_err, sol_tra_err = orbit_err.solve_ODE("Trapezoidal")
    _, sol_RK3_err = orbit_err.solve_ODE("RK3")

    # Compute the error
    err_tra_i = RunOrbits.error_estimate(time_err, sol_tra_err, time_ref, sc
    err_RK3_i = RunOrbits.error_estimate(time_err, sol_RK3_err, time_ref, sc

    # Append the results
    error_tra.append(err_tra_i)
    error_RK3.append(err_RK3_i)
```

In [41]:
```
# Plot the solutions

plt.figure(figsize = (9, 4))

plt.plot(n_arr, error_tra, color = "lightcoral", label = "Trapezoidal")
plt.plot(n_arr, error_RK3, color = "greenyellow", label = "RK3")

plt.xlabel("n")
plt.ylabel("Error [AU]")
```

```
plt.title(f"Covergency of Integrator Methods for e = {orbit_err.e}")

plt.grid(alpha = 0.2)
plt.legend(frameon = True, fontsize = 11)

plt.show()
```



Show in logscale.
Add reference lines for comparison.

Both methods reach good convergence around the 500-step point. However, it is observed that the trapezoidal integrator converges first, requiring fewer steps. Additionally, the RK3 method is much more accurate at the beginning when only 100 steps are used.

Note: Please include all your simulation outputs in the **outputfolder** for a reference.

Exam submission:

- Send your code **in a single .tar ball file via email** to wbanda@yachaytech.edu.ec by the deadline.