Unit 1: Classes & Copilot

Computer Modelling

1 Aims

In this unit you will:

- learn how to interact with the AI assistant.
- learn how to make and use a class with attributes and methods.

Read the sections of the course notes on Object-oriented Programming before starting this, or you won't understand it.

2 Preparation

Create a new directory for this unit, and open it in VSCode by selecting "Open Folder" from the "File" menu and navigating to it. Download `unit1.py` from Learn and save it in your new folder.

Open the file in the explorer on the left. If the directory contents are not visible, click the top icon on the left of the window, which looks like two pieces of paper.

Check you can run the file by clicking the "play" icon at the top right of your code window; it should open a terminal at the bottom and give you the message "NameError: name 'Cosmology' is not defined" because you have not yet completed the unit.

3 Background

The physics background here in the first two sub-sections will not be examined in the semester 1 exam - it is just to help you understand the problem you are working on. The third sub-section is about the AI assistant CoPilot, and you should read it carefully.

3.1 Redshift and distance

When we see a distant object in the sky like a galaxy, we can try to measure to important quantities about it: its *redshift*, and its *distance*.

Distant galaxies are so far away that it takes that light from them millions to billions of years to reach us. This means that when we look at a galaxy, we are seeing it as it was in the past. This also means that during that time, the Universe will have expanded detectably, so the galaxy will have moved further away from us.

We measure the expansion of the Universe using a quantity called *redshift*. This is a measure of how

much the Universe has stretched out since the light from the galaxy was emitted. The redshift is given the symbol z, and $z = 0$ means the present day, and redshift increases as we look further back in time. $z = \infty$ means the Big Bang. The most distant galaxies are at about $z = 10$, but most galaxies we see are at $z < 1$.

Because of the expansion of the Universe, the *distance* D to a galaxy is ambiguous. We will focus here on one particlar type of distance measurement, but the details are not important here.

It is relatively easy to measure the redshift z of a galaxy by looking at its spectrum, but it is much harder to measure the distance to it, though it can be done in some cases.

3.2 Mathematical model

The distance D to a galaxy is related to the redshift of the galxay z by this equation:

$$D(z) = \frac{c}{\mathrm{H}_0} \int_0^z \left[\Omega_m(1+z')^3 + \Omega_k(1+z')^2 + \Omega_\Lambda\right]^{-\frac{1}{2}} \, \mathrm{d}z'$$

The speed of light c is a well-measured constant, but the other parameters in the equation are not precisely known. We can use this equation, together with separate measurements of both D and z, to try to determine the values of these quantities. This is what we will do next semester using your class.

The parameters are:

- H_0 is the Hubble constant, representing the expansion rate of the Universe today, measured in kilometres per second per MegaParsec.
- Ω_m is the fraction of matter in the Universe today. It is around 0.3.
- Ω_Λ is the fraction of dark energy in the Universe today, an unknown substance accelerating the universe. It is around 0.7.
- Ω_k is the fraction that Einstein's curvature of space contributes to cosmic density. It seems to be close to zero; if it is exactly zero then we call that a *flat* Universe.

Three of the parameters are related via $\Omega_k + \Omega_\Lambda + \Omega_m = 1$.

We call a specific set of the cosmological parameters, with an understanding of their meaning, a *cosmological model*.

3.3 Prompting CoPilot

Copilot is an AI assistant that can help you write code. It is not a substitute for actually understanding, and you won't get any partial credit for answers in this course with mistakes in because you didn't check its output.

CoPilot will now run automatically in your files, suggesting what you might want to type next. You can accept its suggestions by pressing `Tab`. Its suggestions depend on what you've typed previously, so you may need to type a little before it starts to help. You can push it in the right direction by typing some comments or code that you want it to follow, for example, typing this:

```
# function to compute the hypotenuse from the two sides of a triangle
```

will usually prompt it to write a function that does that. Similarly you can start writing some code and it will try to complete it, for example, writing:

```
def save_array(filename, array):
```

will prompt it to write a function that saves an array to a file.

Note that it provides the most common or popular solutions, not necessarily the best. When I ran this example it suggested a function that works for 1D arrays, but nothing else.

You can also ask it to write code directly, by pressing `Ctrl-I` and typing a prompt. Or you can open a chat terminal to talk in more detail, by clicking the "GitHub CoPilot Chat" icon in the bottom left of the left hand-pane of the window. There you can ask it to explain things generally, or if you have a file open you can ask it about things in that code. You can also ask it to write code here.

4 Tasks

4.1 Creating a class

We will write a `Cosmology` python class to represent cosmological models.

It will store the values of a model's parameters, so that we can create instances of it from the values of H_0, Ω_m, and Ω_Λ.

- Write a class called Cosmology with the three attributes in your file `unit1.py`. The more specific you are in your prompt the better it will work. Ensure you understand everything it has written, removed things not needed here, and made sure everything else works.
- Ask CoPilot to explain what the `__init__` method does, and write that explanation in comments in your code, *in your own words.*
- Ask it to explain what `self` means in the code it writes, and write that explanation *in your own words* in comments too.
- Upgrade the code to calculate and store Ω_k from the other three.

4.2 Adding methods

Methods are functions that are attached to a class. Add these methods to your class to:

1. Compute the integrand of the distance formula:
$$\left[\Omega_m(1+z')^3 + \Omega_k(1+z')^2 + \Omega_\Lambda\right]^{-\frac{1}{2}}$$
2. Return whether the Universe is flat.
3. Set Ω_m and modify Ω_λ accordingly (keeping the curvature of the Universe the same).
4. Set Ω_λ and modify Ω_m accordingly (also keeping the curvature of the Universe the same).

5. Return the quantity $\Omega_m h^2$ where $h = \mathrm{H}_0/100\mathrm{km/s/Mpc}$.

6. Return a string describing the model, e.g. `<Cosmology with H0=72.0, Omega_m=0.3, Omega_lambda=0.72, Omega_k=0.02>`. This method must be called ___str___. Ask CoPilot to explain what this method is for, and write that explanation in comments in your code, in your own words.

In each case you will need to check that the AI has written the code correctly, and if not, correct it.

Write a simple function you can run that shows the use of these methods, and that they are working.

Note that in the first you don't have to integrate to actually get D yet; we will do that in unit 3.

4.3 Adding a tester function

1. Make a plot of the integrand between $z = 0$ to $z = 1$ at fixed parameters of your choice.

2. Make a second plot showing how varying Ω_m changes the integrand behaviour. Explain your findings in the comments.

3. Make a third plot like the last one, but using your "setter" method from part 3 of the previous section. Explain how this differs from the previous plot in the comments.

4. Print out various different `Cosmology` objects and explain in the comments how this is related to your work in the previous section.

5 Submitting

Submit your `unit1.py` file to Learn. The marker should be able to run it and have it correctly execute your main function without crashing, running all the tests above.

5.1 Mark Scheme

This unit is marked out of 10, and counts in total for 5% of your final grade for this course. You get 5 marks for the class and methods, and 5 marks for the tester function. We're not marking you on code quality and style for this exercise, but it's still good practice for you to write clean, readable code.