Unit 4: Likelihoods & Optimization

Computer Modelling

1 Aims

- Compute Gaussian likelihoods
- Compute the likelihood of a sample of supernova measurements.
- Find optimal parameters for our model using your likelihood

2 Preparation

Make a new directory for this unit, and download the provided data file `pantheon_data.txt` into it.

Copy your cosmology class from Unit 3 into the directory too, in a file called `cosmology.py`. Make sure that you can import that file without it doing any tests or creating any plots; you can delete the unit 3 main function, as we will only need the cosmology class from now on.

Make a new file called `unit4.py` where you will write the code for this unit.

Create a results document in word or google docs for your plots and values for this unit.

Go through your feedback from unit 3, and make changes to fix any problems they identify with your distance modulus calculation and the methods it depends on. We will be using this code extensively in semester 2. Here are some reference values that you could check against:

- $\mu(z = 0.5) = 42.61580$ for $\Omega_m = 0.25$, $\Omega_\Lambda = 0.7$, $H_0 = 60.0$
- $\mu(z = 1.0) = 44.10024$ for $\Omega_m = 0.3$, $\Omega_\Lambda = 0.7$, $H_0 = 70.0$
- $\mu(z = 1.5) = 44.76602$ for $\Omega_m = 0.4$, $\Omega_\Lambda = 0.65$, $H_0 = 80.0$

If you get different values or can't find your problem then speak with a TA or post on Piazza.

3 Background

3.1 Likelihoods

A **likelihood** is a measure of how well a model fits some data. It is the probability $P(d|\theta)$ where d is some observed data and θ is the model parameters we are testing. Typically, to compute a likelihood we first compute some theoretical prediction for what the data would look like if the model were correct, and then compare this to the observed data.

The specific likelihood we should use depends on the kind of noise or other scatter in the data, but a particularly common form is the *Gaussian likelihood*. In this case, for a single data point, the likeihood is:

$$P(d|\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(d - m(\theta))^2}{2\sigma^2}\right)$$

where d is the observed data, m is the model's prediction for d, which depends on the parameters θ, and σ is the standard deviation of the noise in the data. This likelihood is a function of the difference between the data and the model, and is largest when the model is closest to the data.

We often make use of the (natural) log-likelihood function, which for Gaussians is, discarding a constant factor:

$$\log P(d|\theta) = -\frac{1}{2}\frac{(d - m(\theta))^2}{\sigma^2}$$

Because independent probabilities multiply, the likelihood of a set of data points is the product of the likelihoods of the individual points. Because $\log ab = \log a + \log b$, this means that the log-likelihood of a set of data points is the sum of the log-likelihoods of the individual points. So for a set of data points d_i with standard deviations σ_i and model predictions $m_i(\theta)$, the log-likelihood is:

$$\log P(d|\theta) = -\frac{1}{2}\sum_i \frac{(d_i - m_i(\theta))^2}{\sigma_i^2}$$

Note that the errors for the different data points can be different, and the likelihood is still valid. This is a very common situation in real data analysis.

Also note that the instructions for unit 1 were missing the minus sign in this formula!

3.2 Maximium likelihood optimization

It is usually of interest to find which model parameters give us the best fit to the data. In some cases this my be found analytically, by computing the derivative of the likelihood with respect to the parameters. In other cases, including ours here, that is either impossible or not really worth the effort, and we will use a numerical optimization method.

The `scipy` library includes a minimization function that can optimize a function. For example if you have a function $f(\mathbf{x})$ where $\mathbf{x}$ is a vector of parameters, you can use `scipy.optimize.minimize` to find the values of the components of $\mathbf{x}$ that give the smallest possible value of f.

In general minimization is quite untrustworthy - it can work very well in one scenario but very badly in another. It tends to work better if you can provide sensible initial guesses for the parameters, and if you can provide bounds on the parameters. It is also a good idea to try the optimization multiple times starting from different places, to show that the result is robust.

It is easier to maximize the log-likelihood, rather than the likelihood itself. This should give the same result, but is often more stable numerically.

3.3 Distance moduli and supernovae

3.3.1 Distance modulus theory

In unit 3 you wrote a method that computes a theoretical distance modulus to a given redshift. We can relate this to the *magnitude* m of an object like a supernova, which is a dimensionless measure of its brightness. The relationship is:

$$m_i = \mu(z_i) + M$$

where $\mu(z_i)$ is the distance modulus at the redshift z_i of the supervnoa, and M is a fixed *absolute magnitude* of the supernova, an estimate of its intrinsic brightness, and is the same for all supernovae in our collection.

Note that as the distance modulus increases, the magnitude increases too. Higher magnitude values correspond to dimmer, and so more distant, objects.

We can fix $M = -19.3$ for now, though you may vary it later in the semester, so keep this in mind. The modulus μ depends on the cosmological parameters Ω_m, Ω_Λ, and H_0 as well as the known redshifts z_i.

These m_i values correspond to the modellled $m_i(\theta)$ in the likelihood formula.

3.3.2 Distance modulus data

The data file `pantheon_data.txt` contains a set of measurements of the observed magnitudes of some supernovae at different redshifts.

Its three columns are

1. the redshift z_i of the supernova
2. a (noisy) measurement `mobs` of its magnitude
3. an estimate of the error `sigma` on the measurement

Since it contains measured data values, `mobs` corresponds to the d_i values in the likelihood formula. The `sigma` column contains the σ_i values in the likelihood formula. The redshifts z_i are used to calculate the theoretical prediction from the previous section.

3.4 Calling classes

We have already used several special python methods in our classes, `__init__` and `__str__`. Here you will use another, `__call__`. This method makes your objects *callable*, so that you can use instances of it it like a function. For example:

```python
class MyClass:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __call__(self, x):
        return self.a * x + self.b
```

```
m = MyClass(2, 3)
x = m(5)  # call the class like a function. This runs its __call__ method.
print(x)  # print 2 * 5 + 3 = 13
```

You've actually already used a class with this feature - the scipy interpolator that you used last unit works like this.

3.5 Keywords

Python functions and methods can take optional arguments which, if you do not specify them, take on a default value. You have already used these in matplotlib plotting, for example. where e.g. the `label` option is not required and left blank if unset.

Here is an example:

```
def f(x, n=1000):
    return x * n

print("Without keyword", f(2))     # prints 2000
print("With keyword", f(2, 10))    # prints 20
```

This tool is especially useful when you would like some optional behaviour that a functions sometimes include, but doesn't have to, or in cases where there is a sensible default value for a parameter and most of the time the user will want to leave it as that value.

You can use keywords in methods exactly the same as in functions.

4 Tasks

4.1 Likelihoods

- Write a `Likelihood` class that loads the provided data file when initializated, stores the columns in the object, and then when called like a function with a numpy vector of parameters $\theta = \{\Omega_m, \Omega_\Lambda, H_0\}$, computes the log of the Gaussian likelihood for all the supernovae using an instance of your cosmology class. The value of the likelihood for $\Omega_m = 0.3$, $\Omega_\Lambda = 0.7$, $H_0 = 70.0$ should be around -476.

Hint: It will be useful later to have a separate method that computes the theoretical model $m(z)$ for a given z vector. You can call this from inside your `__call__` method, and also use it to make plots later.

- Test the convergence of your likelihood calculation with respect to the number of integration/interpolation points N. For convergence we want the change in the log-likelihood to be small compared to 1.0. Show a convergence plot in your results document, and also record there what N value you will use in the rest of the unit and why.

4.2 Optimization

- Use the function `scipy.optimize.minimize` to find the best-fitting parameters for this model. You will need to:

  - provide an initial guess for the parameters, which you can choose yourself.

  - use the special `bounds` keyword to restrict the values of the parameters to being positive, and with a reasonable maximum. Have a look at the documentation for the function.
  - remember that you're trying to maximize the likelihood, but the function minimizes things. You need to deal with this somehow.
  - explore the object that is returned by the optimization function.

- Print the best-fitting parameters and the value of the likelihood at that point. Record the values in your results document.

- Make a plot showing the data, with error bars, and see if the best-fit model points go through it. Save it in your results document.

- Make a plot of the *residuals* of the data points, which are the differences between the data and the best-fit model predictions, divided by the errors:

$$r_i = \frac{d_i - m(z_i, \theta)}{\sigma_i}$$

This is a good way to see if the model is a good fit to the data. If everything is working the r_i should be distributed around zero with a standard deviation of somewhere near 1. (In this case they will be a bit below 1 because we have neglected some things in our analysis.) Save the plot and record the mean and standard deviation of the residuals in your results document.

4.3 Changing the model

- Add a *keyword* to your call function that allows you to choose between two different models. The first model should be the standard one you have been using so far. The second should be the same, but with Ω_Λ fixed to be zero (so varying only two parameters, Ω_m and H_0).

- Use your new keyword option to repeat your optimization process to find the best fit model with $\Omega_\Lambda = 0$. Record the best-fitting parameters and the likelihood in your results document, with a note on which model fits the data better. Coment on the parameter values you find for this model in your document.

5 Submitting

Submit three files through learn, `unit4.py`, `cosmology.py`, and a PDF of your results document.

5.1 Marking

You will be marked out of 20 for this unit, and the submission is worth 10% of the final course grade. The marks are broken down as follows:

- 5 marks for the results document.

- 10 marks for code functionality, including sensible use of numpy and OOP.
- 5 marks for code quality and style, following the conventions we discussed in the Style mini-lecture. The slides for this are on Learn.