Comparing models with data

3.1 Learning outcomes

Mathematics and Physics

- Fit model parameters to data.
- Least square method

Computing and Python

- Use `curve_fit` python function
- Modify a provided program.
- Use the Monte Carlo method.

Preparation for assignment

- Write code to fit a model to a data set.

3.2 Simple Data Fitting

There is no much point in developing a model if we do not compare it with real data. We must thus find methods to do determine the value of the model parameters which best match the data. This is called data fitting.

In general, we have a collection of data D_i, $i = 1 \dots n$ measured at the specific time t_i. We also have a model with a function $F(t, p_1, \dots, p_n)$, which is expected to describe the data, where the p_j are a set of parameters. The problem consists in finding the values for each of the parameters $\tilde{p}_j$ such that the $F(t_i, \tilde{p}_1, \dots, \tilde{p}_n)$ are as close to the D_i as possible. By *as close as possible* we mean to minimise the function

$$S = \sum_{i=1}^{n} (D_i - F(t_i, p_1, \dots, p_n))^2. \tag{3.1}$$

This is called the least square method.

Let us consider a simple example: the elevation of a road is measured approximately every 50 meters and one has the following values

x	h
0	1.2
49	5.1
102	9.3
151	13.1
197	17.9

The road has a constant slope so we should be able to fit the date to a function of the form

$$h(x) = a + bx \tag{3.2}$$

where a and b are 2 parameters which we would like to determine.

This can be easily done using the `curve_fit` function of `scipy.optimize` python module. It does perform the fitting using the least square method and it is relatively easy to use. One must first store all the values of x and the corresponding values of D in 2 separate arrays. (Line 6 and 7 in the code below). One must must then define the function we want to fit. This is done line 10-11. Notice that `func` takes 3 arguments. The first one is the value of x and the next two are the two parameters a and b. There are only two parameters in this example, but one can have any number.

file: `linear_fit.py`

```
1  import math
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import scipy.optimize
5
6  X = np.array([0.0, 49, 102, 151, 197])
7  D = np.array([1.2, 5.1, 9.3, 13.1, 17.9])
8
9  # The function we want to fit
10 def func(x, a, b):
11     f = a + b*x
12     return f
13
14 # Fit the data X, D with the function, starting from a=0 b=0.1
15 popt,pcov = scipy.optimize.curve_fit(func, X, D, p0=[0,0.1])
16
17 print("a=", popt[0], "b=", popt[1])
18
19 # Plot the data and the fit
20 plt.xlabel("X", fontsize=22)
21 plt.ylabel("h", fontsize=22)
22 plt.xticks(fontsize=20)              # Makes x labels digits larger
23 plt.yticks(fontsize=20)              # Makes y labels digits larger
24 plt.tight_layout(rect=[0, 0, 1, 1]) # Ensure the full figure is visible
25
26 plt.plot(X, D, "b*")                # Plot data
27 plt.plot(X, func(X, *popt), 'r--', label="fit to a+bx")
28 plt.show()
```

curve_fit

The python function `curve_fit(func, X, D, p0=[0,0.1])` is then used to perform the fitting. The first argument is the name of the function to fit, `func` in this example. The next two are the data x and D as arrays. The last argument is the initial value to use for the parameters a and b as a list. The number of parameters in the list much match the number of parameters used by `func`. It is usually important to get a reasonable values for a and b otherwise the fitting does not work well (a weakness of the method used by `curve_fit`). The end of the code just displays the results. Notice that we add labels to the graph axis (line 20 and 21). We also use a font size larger than the default (lines 22-24) to make the figure easier to read.

Programming tip: we set the font sizes to be larger than the default value. This generates nicer figures. The `tight_layout` ensures that everything appears on the figure. Without it some of the labels and axis values might be only partially visible because they use a larger font.

Problem 3.1:

Run the program above. Is the fit reasonable? What is the meaning of the parameter `b`.

3.3 Beetles Population Model

We can now revert back to our population model. The file `BeetlesData.txt` in the `data` folder contains some experimental value of a beetles population over a few weeks, starting from a single couple.

We will start by trying to fit the logistic model to the data. As the population will not be normalised we have to use a saturation parameter K and the equation is then

$$N_{n+1} = RN_n(1 - N_n/K). \tag{3.3}$$

As before this states that the population at time $n+1$ is the population at time n multiplied by a coefficient R but attenuated by $1 - N_n/K$. This means that the population is saturated when $N = K$. As N increases and becomes closer to K, the factor becomes smaller and the population is prevented from exploding.

The first problem is that to fit data we need an analytic expression for $N(t)$. We can't solve the discrete logistic equation in closed form but, for this model, we can obtain a good approximation as follows: we write N as a continuous function of time t and use the Taylor expansion of $N(t + dt)$:

$$N(t + dt) \approx N(t) + dt\frac{dN}{dt} = RN(t)\left(1 - \frac{N(t)}{K}\right), \tag{3.4}$$

where we have used (3.3). We can easily rewrite this by dividing the equation by dt and moving $N(t)$ from the left to the right:

$$\begin{aligned}
\frac{dN}{dt} &= -\frac{N(t)}{dt} + \frac{R}{dt}N(t)(1 - \frac{N(t)}{K}) \\
&= \frac{R-1}{dt}N(t) - \frac{R}{dt\,K}N(t)^2 \\
&= \frac{R-1}{dt}N(t)\left(1 - \frac{R}{(R-1)\,K}N(t)\right) \\
&= rN(t)(1 - N(t)/k)
\end{aligned} \tag{3.5}$$

where $r = (R-1)/dt$ and $k = K(R-1)/R$. Equation (3.5) can be integrated in closed form to get

$$N(t) = \frac{k}{kbe^{-rt} + 1}. \tag{3.6}$$

where b is an integration constant (this can be checked by direct substitution but you do not need to check this now).

We will now fit the data with this expression. We have 3 parameters to fit, k, r and b.

Problem 3.2:

- Make a copy of `linear_fit.py` and call it `beetles_fit_log_sol.py`.
- Add the following lines at the beginning of the program:

```
1 data = np.genfromtxt("data/BeetlesData.txt", delimiter=" ")
2 week, beetles = zip(*data)
3 week = np.array(week) # convert tuple to array
4 beetles = np.array(beetles) # convert tuple to array
```

This replaces line 6 and 7 in `linear_fit.py`.

genfromtxt

The numpy function `genfromtxt` reads a file and convert each line to an array of numbers, themselves returned as an array. So a file containing the lines

```
1   1 10
2   2 20
3   3 30
```

is returned as the array `[[1,10],[2,20],[3,30]]`.

The `zip` function takes a number of lists as argument so that

```
1     zip([1,10], [2,20], [3,30])
```

zip

returns

```
1     [1,2,3], [10,20,30]
```

or in other words, it zips the lists together.

Now, if `D = [[1,10],[2,20],[3,30]]` , `zip(*D)` (notice the asterisk), is equivalent to `zip([1,10], [2,20], [3,30])`.

- Modify the function `func` so that it corresponds to the function (3.6). You must also replace `X` and `D` by `week` and `beetles` everywhere and update the figure labels.
- There are now 3 parameters to fit: `r`, `k` and `b` (keep them in that order). So you must change `func` and the `p0` argument of `curve_fit` accordingly. As an initial value take `r=1.0`, `k=1000.` and `b=0.01` .

Check Point 3.1

3.4 Fitting Exponential Data

There are several ways to fit data. We can fit the data as they are provided or fit the data after a monotonic function is applied to the data. For example, suppose we have a set of data of the form $t, d(t)$: `(0,d_0), (1,d_1), ... (n,d_n)` which we want to fit with the exponential function $D(t) = A\exp(k\,t)$ for some parameters A and k. We can fit the data as provided directly with the exponential, or we can take the logarithm of the data and fit them with the logarithm of the function $\ln(A\exp(k\,t)) = \ln(A) + k\,t$.

Is there any difference between the two? Fitting a linear curve is easier to perform, so, in this case, there is an advantage in fitting the logarithm of the data. Will the values obtained for A and k be the same? The answer is no and the reason is that the data fitting algorithm work by minimising the quantity

$$E = \sum_{i=0}^{n} \left[(d_i - D(i))^2\right]. \tag{3.7}$$

When we use the data as provided, the data corresponding to the largest value will have effectively a larger impact on the minimisation as small deviation will lead to larger difference between the data and the fitting function. If the data span several orders of magnitudes, the smaller data point hardly contribute at all the the fitting. When we fit the logarithm of the data on the other hand, we make all the data much closer in value and so make the impact of all data value more similar.

Deciding which method is best depends on what we try to determine. There will be system where small values genuinely carries less weight and other when they do matter just as much as larger values.

As an example, we will consider the Covid 19 data between 1st June and 20 July 2021 in the United Kingdom. Just before the restriction were lifted (on July 19th).

The program `read_covid.py` defines the function `get_data` which reads the specified column of a `csv` file and returns it as an array. When one runs the program it reads the file `covid_JuneJuly2021.csv` displays the data (the new infections during the period covered) on a logarithmic plot. The 0^{th} column contains the date of observation, but for simplicity we will use the index of each element as the number of days since 1st June 2021. Run the program to see the observed curve of infection on a logarithmic plot. It is a more or less linear curve implying the number of infections has the form $A\exp(kd)$ where d is the number of days since 1st June 2021. Interpreting the parameter k is not easy. Instead we can easily compute the number of days τ that it takes to double the number of infections. We have $A\exp(k(t+\tau)) = 2A\exp(k\,t)$. From this we derive

$$\tau = \frac{\ln(2)}{k}. \tag{3.8}$$

We will now write a program to determine τ from the data.

Problem 3.3:

Write a program, called `fit_covid_data.py` which fits the number of new infections, available as the first column of the data file `covid_JuneJuly2021.csv`, as an exponential of the form $A\exp(kd)$ where d is the number of days since 1st June 2021.

In your program, you must start by importing the modules `numpy`, `scipy.optimize` as well as `matplotlib.pyplot`.

You must then proceed as follows:

- Use the numpy function `genfromtxt("covid_JuneJuly2021.csv", delimiter=',', comments="#")` to read the data into a 2 dimensional array. Each row of the array corresponds to a line in the file. The first line is skipped because it is a comment. genfromtxt
- Extract the second column of data (remember it has index 1) from the array to obtain a vector with the total infection values. Store it in a variable called `data`.
- We then need to generate an array containing the day number since the starting date of the data: using `days = np.arange(len(data))` will do the trick.
- Define a function `func(d, A, k)` to evaluate the function to fit $f(d) = A\exp(kd)$.
- Use the `scipy.optimize.curve_fit` fit function to fit the data. Take `1000` for the initial value of A and `0.01` for k.
- Print the values of A and k obtained from the fit as well as the doubling time τ derived from k.
- Plot on the same logarithmic graph the data and the fitted function. (use the `plt.semilogy` function for this).

Check Point 3.2

Problem 3.4:

Make a copy of the program `fit_covid_data.py` and call it `fit_covid_log.py`. Then modify it so that it fits the logarithm of the data to the logarithm of the function $A\exp(kd)$ *i.e.* to the function $B + k\,d$, where $B = \log(A)$. The program should still plot the actual data and the function $A\exp(kd)$ on a logarithmic plot.

Compare the 2 graphs and the 2 values of τ. Are they different? If not, is the difference very big?

In the previous example, the difference between the two methods was not very big. We will now analyse the data of Covid 19 in Italy during the first few months of the epidemics.

Check Point 3.3

We will now look at the data from Italy during the onset of the epidemics. Because few test were available at the time, we must use the total number of fatalities instead (they are closely related but delayed in time).

Problem 3.5:

Write a program called `fit_covid19_italy.py` similar to the previous two which reads the number of total deaths in Italy from the file `Covid19_Italy_2020.csv`. The total number of fatalities are in the 5th column of the file.

For this example, the data file contains some null values and we can't take the logarithm of these. To still be able to fit the data, all we have to do is to remove the zeros and the corresponding days when they occur. Import the module `read_covid` to be able to use the function `rem_zeros()` which does exactly that. It takes an array of values `array([v0,v1,v2,... vn])` and returns a list of the form `[[d1,v1],[d2,v2],.. [dn,v]]` where all the null values have been removed and where the `di` are the days corresponding to the value `vi`. One can the use the `zip` function to extract the list of days and the list of corresponding values and then convert them to arrays:

```
days, data = zip(*rc.rem_zeros(data))
days = np.array(days)
data = np.array(data)
```

You can then fit the data with the function $f(d) = A\exp(kd)$ (taking 1 as the initial value for A and 0.01 for k). In the same program fit the logarithm of the data to the function $f(d) = B + k\,d$ (starting from $B = 1$, $k = 0.01$).

The program must generate on the same graph a logarithmic plot of the data as well as the 2 fitted curves. It must also print the value of A, K and τ obtained from both fit, so that they can be compared.

What do you conclude from the 2 types of fitting? Which one is best? Why did we not have the same problem when fitting the UK data in 2021?

Check Point 3.4

3.5 Extra problems

We will now look at the global warming problem and fit data of the recorded Earth average temperature during the last few decades. The file AT.txt— contains the difference between the recorded average temperature of the Earth and the reference temperature of $13.9°$. This corresponds to the Earth temperature around the middle of the 20th century. The data start in 1880, but we will only be interested in finding out what happened since 1960, the year when the Earth started to warm up more significantly.

We will fit the data with the linear function

$$\Delta T = a + b\,Y \tag{3.9}$$

where Y stands for the year.

Problem 3.6:

Write a program that will determine the estimated year when the Earth temperature will have increased by $1.5°$.

- Invert (3.9) to obtain the year when $\Delta T = 1.5°$ if we know a and b.
- Make a copy of one of the previous programs and call it `fit_EarthTemp_data.py`. Then use `gdata = np.genfromtxt("AT.txt", delimiter=',', comments="#")` to read the file. `gdata[:,0]` and `gdata[:,1]` will provide arrays containing respectively the years and the ΔT.
- Determine the index of the year 1960 in the array of years and remove all the years before that years from the array. Then remove the corresponding values of ΔT from the data. (Hint use slices).
- The program must then fit the data with the function, using the data from 1960. (3.9).
- It must then print the values of a and b on the same line.
- The program must then print `"1.5 degree will be reached in"` followed by the year, as an integer, when it is likely to happen.
- The program must then generate a figure of ΔT as a function of the year with one curve for the data and 1 curve for the fitted function.

In a separate file answer the following questions:

- When is ΔT likely to reach $1.5°$?
- What is the meaning of the parameter a in (3.9)?
- What is the meaning of the parameter b in (3.9)?