MANCHESTER 1824

The University of Manchester

PHYS20762 Computational Physics

Project 1: Data Analysis

Measuring Spreading Law

2021/2022 Session, Semester 2

Project Description

The type of data analysis discussed in this project was covered in year 1. Therefore, you will most likely find it to be straightforward. However, the purpose of this project is to teach you how to use **Python code**, **Jupyter notebook** & Blackboard pages for the Computational Physics course, and hence completing the project is mandatory. The Physics motivation for this project & the statistical techniques used for data analysis are presented in the videos associated with **weeks2&3**. The page also contains Jupyter notebooks with examples of data analysis which are useful. *You should expect to have initial results for this project within the first 3-hour of the laboratory session & two lab sessions should be more than sufficient to complete this project*. To complete the project 1, expand the file **phys20762-project1.ipynb** linked to the page **weeks2&3**. You will notice that your tasks listed below will appear as separate cells in this notebook. Fill in the missing code & submit the work before the deadline by following the instructions on Blackboard. The marks for this project will not contribute to your final mark, but you are expected to submitted it before the project 2 gets marked.

In the first project, you will analyse experimental data from the spreading of picolitre droplets on a flat substrate to determine the corresponding spreading law. A spreading law is a relationship between the speed of the contact line and the contact angle. The **data** analysed here follows experiments of Kant et al. (2017)[1] & already appears in the **phys20762-project1.ipynb** file. If you would like a challenge[2], upload the original experimental data from the three files, which can be access via **weeks2&3** page. Each of these files contain 35 data points, listed as two columns: time from the start of the experiment, measured in *seconds*, and droplet footprint radius, measured in *micrometres*.

You should:

- Examine the data by plotting the time-evolution of the position of the contact line, i.e. radius should increase as a function of time $R=R(t)$. Make sure that you include errors on R estimated from the data obtained in the three experimental runs (*week 1 of project*).
- Relate the instantaneous radii R of the drops' footprints to their contact angles ϑ using the spherical cap approximation:

[1] The paper by Kant et al. (2017), "Controlling droplet spreading with topography", is available on Blackboard.

[2] This is completely optional in project 1, but is quite common when using Python for data analysis.

$$V = \frac{\pi}{6} H(3R^2 + H^2), \quad (*)$$

where $V = 7.6pL$ is the volume of the drops and H is the maximum droplet height. From the same approximation it follows that

$$\vartheta = \frac{\pi}{2} - \tan^{-1} \frac{R^2 - H^2}{2RH}.$$

The relationship (*) allows you to solve for H using the cubic formula for root of polynomials or you could treat (*) as an implicit equation for H and apply the nonlinear least square method to solve it (e.g. use *scipy.optimize.fsolve*). *Make sure that you understand the geometrical arguments leading to the formulas above.* As above, estimate the errors on ϑ by relating R to ϑ for each set of data. Produce the corresponding plots with errorbars (*week 1 of project*).

- Use the radius data to find the speed *U* of the contact line for each of the experiments & plot it in a graph. Then find the mean speed and estimate the corresponding errors. Plot this result on a graph. Finally, produce the plots of the speed versus the contact angle with error bars. This is the data needed to infer the empirical spreading law for the droplet (*week 1 of project*).
- Find the best-fit function and the fitting coefficients (with errors) for the dependence of the interface speed on the contact angle. Try the Cox-Voinov law [$U(\vartheta) = U_0(\vartheta^3 - \vartheta_0^3)$] and the de Gennes law [$U(\vartheta) = U_0(\vartheta^2 - \vartheta_0^2)$] (*week 2 of project*).
- Determine the *chi2* for both types of fits & examine the plots of the residuals. Reason which of the laws is more appropriate for this drop (*week 2 of project*).
- Plot the spreading laws obtained from the experiments and the corresponding fits. Display the corresponding velocity scales U_0 with errors and the equilibrium angle ϑ_0 with errors (*week 2 of project*).
- Submit your program (the ".ipynb" file), which can generate all *appropriate* graphs, error estimates, chi-squared analysis, etc. Examiners should be able to run your code on the Jupyter hub. The code must contain comments, from which it should be clear which spreading law is more appropriate for the drop (this information can also be conveyed using graphs). Please also comment on what does the result imply about the range of the contact angles for the particular drop. Don't forget to think about the Physics of the project: (a) are the obtained characteristic speeds of the appropriate order of magnitude? (b) are the calculated errors reasonable considering the scales in your problem (*week 2 of project*).

On "stretch yourself" & "initiative" points (beyond project 1 as well)

Every (marked) projects has *a stretch yourself section* (for which additional marks are awarded). Initiative when doing projects is also rewarded. Below are examples of both in the context of the project 1. This part of the project 1 is not mandatory, but you are encouraged to give it a go!

Stretch yourself for project 1: Try fitting $U(\vartheta) = A + B'\vartheta + U_0\vartheta^2$ and $U(\vartheta) = = A + B'\vartheta + C'\vartheta^2 + U_0\vartheta^3$ to the experimental data and show that the corresponding U_0 are different to the one obtained using when fitting the Cox-Voinov and de Genned laws. Discuss very briefly (using a Markdown cell in the notebook) the reason for the discrepancy (*week 2 of project*).

Example of initiative in project 1: The data of $U = U(\vartheta)$ is not uniformly distributed, i.e. there are more data for smaller values of the contact angle. Is this important when obtaining the drop spreading law? To test this, apply splining to the experimental data to obtain uniformly distributed data & then apply fitting. Is there a significant discrepancy with the previous findings? What is the best way of splining the data? (*week 2 of project*)

Useful Information

How to load data files?

The three data files were recorded using an optical system to capture the profile of a liquid drop on a solid substrate. Each file contains data points in two columns. The first column corresponds to time from the start of spreading and is measured in s. The second column is the drop instantaneous radius measured in μm.

Before loading the data files into the program, don't forget to include the appropriate library, e.g. **import numpy as np**. Then you could apply, e.g.,

data = np.loadtxt(`experimental_run1.txt')

time = data[:, 0]

Radius = data[:, 1]

Experiment with these commands so that you know what they're doing.

How to plot?

There are many graphics libraries in Python, but the most useful one for this unit is **matplotlib.pyplot** (which can be included as **import matplotlib.pyplot as plt**). You can also plot any part of a vector, e.g. from index L (lower) to U (upper), and in green, with **plt.plot(t[L:U], R1_t[L:U], 'g')** (don't forget **plt.show()** to display the plot). Or, you can 'subset' the array between two chosen indices using **array[L:U]**. To plot error bars (like in the case of a vector **y** vs. vector **x**, when the y and dx errors are stored in the vector **dx** and **dy**, respectively) type **plt.errorbar(x, y, dx, dy, 'g')**, or, if instead only error bars in **x** or **y** are required, use **plt.errorbar(x, y, xerr=dx, color = 'green')** or **plt.errorbar(x, y, yerr=dy, color = 'green')**, respectively.

Performing polynomial fitting

Let's assume that you are fitting a polynomial of degree 1 or 2 to data containing column vectors **x** and **y.** To obtain a linear fit, use **p = np.polyfit(x, y, 1)** (but don't forget to load the library first by doing **import numpy as np**). This produces a vector of best-fit parameters **p = [B A]** for the fit. Similarly, a quadratic fit **q = np.polyfit(x, y, 2)** produces the best-fit parameters **q = [C B A]**. The vector **fitp = np.polyval(p, x)** corresponds to **p[0]*x + p[1],** and **fitq = np.polyval(q, x)** to **q[0]*x.^2 + q[1]*x + q[2].** You should be able to demonstrate clearly that the residuals, i.e. **y-fitp**, are sufficiently small where appropriate.

To compute the errors in fitting coefficients, you will have to use a more advanced form of these functions, e.g. for the quadratic fit **p, cov = np.polyfit(x, y, 2, cov=True)**. The covariance matrix contains the information, which allows you to compute statistical errors for the fitting polynomial. The procedures for the cubic fit are the same.

How to solve implicit equations?

In order to obtain the contact angles from the radii data, you might want to solve an implicit nonlinear equation **f(x) = 0** (in this project you don't have to, but you will most certainly face this problem at some point during your career as a physicist). Use **fsolve** from **scipy.optimize** (to import it, use **from scipy.optimize import fsolve**) as **x = fsolve(f, x_0)**, where $\mathbf{x_0}$ is the initial guess for **x**.

How to do cubic spline interpolation?

To interpolate the data, use **interp1d** from **scipy.interpolate** (use **from scipy.interpolate import interp1d**). By default it creates the linear interpolation, but it can be used as **f = interp1d(x, y, kind='cubic')** for the cubic spline interpolation. Using **f(X)** applies the interpolation at the sites **X.**

23/01/2022 Dr Draga Pihler-Puzović (adapted from previous course material)