Data analysis with Python

Dr Benjamin J. Morgan[1] and Dr Andrew R. McCluskey[1,2]

[1] *Department of Chemistry, University of Bath, email: b.j.morgan@bath.ac.uk*
[2] *Diamond Light Source, email: andrew.mccluskey@diamond.ac.uk*

October 31, 2019

Aim

Data analysis is an incredibly important skill in chemistry, however, it is often overlooked. In this exercise, you will get an introduction to using Python to fit experimental data, see how model-dependent analysis works in a simple system, and write a Markov chain Monte Carlo algorithm to quantify the inverse uncertainty in your model. The particular application in this work is the analysis of an IR spectra from a mixture of organic species, however, hopefully you will recognise that the methods used herein are generalisable.

1 Reading in and plotting experimental data

You have been tasked with determining the relative composition of a mixture of two organic species; namely toluene and benzyl alcohol (Figure 1). To do this, you have measured the infrared spectra of the mixture and obtained (from the NIST Chemistry WebBook) the model spectra of the two species in isolation, these can be found in the 'IR spectra' folder on Moodle. If you download and open the files in the Jupyter Notebook file viewer, you can see the structure of the files. You should note that the model datasets (`benzyl_alcohol.csv` and `toluene.csv`) consist of two columns; namely wavenumber ($\bar{\nu}$) and transmittance (T), while the measured experimental data (`mixture.csv`) has a third column that describes the uncertainty in the transmittance.

Objective 1

- Read in the two model datasets, plot them separately using `matplotlib` including axis labels
- Read in the experimental dataset, including the third column. Then investigate how this may be plotted using the `plt.errorbar` functionality

Hint: Googling 'matplotlib errorbar' offers some useful instructions on how to use this function, or you can read the documentation with `plt.errorbar?` in the Jupyter Notebook.

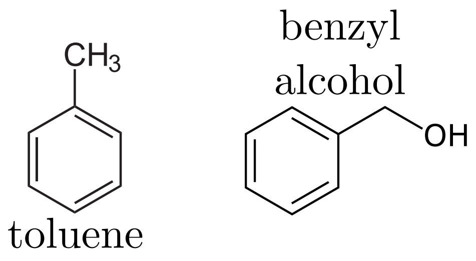

Figure 1: The organic species in the mixture.

Having plotted each of the three datasets (the two models and the experimental), you should be able to identify features present in the experimental data that match those in the models. Such as the large feature in the toluene model data, that we can see in the experiment at around $3000\,\mathrm{cm}^{-1}$, and then at higher energy (higher wavenumber), there is a broad feature in the benzyl alcohol data, between $3250\,\mathrm{cm}^{-1}$ to $3750\,\mathrm{cm}^{-1}$, that can be found in the experimental data.

2 Fitting

You will have hopefully encoutered fitting to data before, for example fitting a line of best fit to some datapoints. This is shown in Figure 2, where the blue dots are some discrete datapoints and the orange line is the line of best fit. In order to determine this line of best fit, we have a metric that we want to minimise that represents the difference between the data and the model (the line). This metric is shown in the lower plot, which is the square residual, calculated as the difference between the data point and the position on the y-axis of the line at the same x-axis position as the data point, squared. The purpose for the squaring is put a heavier emphasis on the outlining datapoints, where the difference is large. We can collapse these values for the individual datapoints down into a single value by taking the sum of this square residuals. This goodness-of-fit metric is referred to as th χ^2-value for the fit, the closer the χ^2 is to zero, the better the agreement between our model and data.

3 Interpolation

From your earlier plots, you may not that the values of wavenumber that the spectra have been measured over is not the same between the three datasets, this can be clearly seen by printing the wavenumber for each of the three datasets. In order to determine the χ^2-value between our model (outlined later) and the data (the mixture), it is necessary that the positions on the x-axis for the two are the same, and since our model will be created from mixing the toluene and benzyl alcohol spectra we will need to normalise these x-axes. This means that interpolation is necessary.

Interpolation is where we determine new data points within the range of a discrete set of known points. Essentially we use what we know about the x- and y-values and guess at the y-values of the different set of x values. It is important that the new range of x-values is from within the existing range, or else we

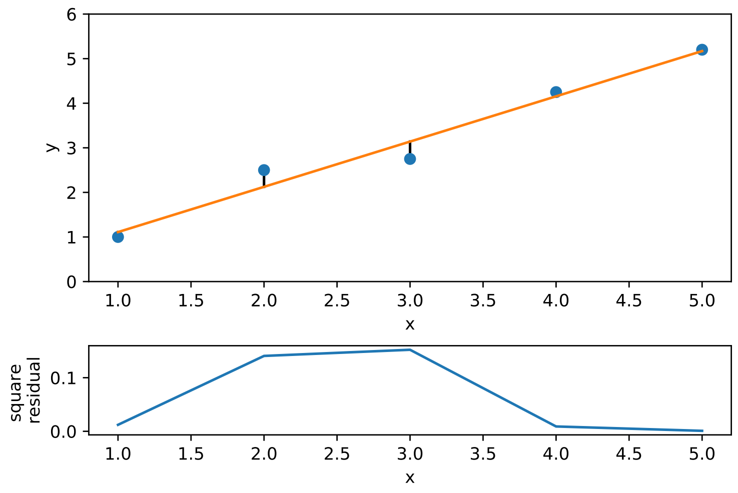

Figure 2: A line of best fit through some datapoints and the squared residuals of the fit.

would be extrapolating (which is often unscientific). For the data that you are using, the experimental data lies within the range of the other two datasets, and therefore we will use the wavenumber x-values from the experimental dataset and interpolate values for the model data.

To interpolate new y-values for the two models, we will use the `np.interp` function, the documentation for this can be found online (`https://docs.scipy.org/doc/numpy/reference/generated/numpy.interp.html`) or by using the `?` command in the Jupyter Notebook. Note that this function takes three arguments, in the documentation these are called `x` which is the new x-axis that values should be interpolated for, `xp` which is the old x-axis values, and `fp` which is the old y-axis values. This function will return a new set of y-axis values, which you should store in a suitably named variable.

Objective 2

- By evaluating the minimum and maximum of each dataset (using the `np.min` and `np.max` functions), prove to yourself that the experimental data lays within the model data on the x-axis
- Create a new variable called `optimisation_x` which is the range of x-values to be interpolated over (make it equal to the wavenumebr values for the mixture data)
- Using `np.interp`, interpolate the toluene and benzyl alcohol data within the values of wavenumber for the mixture
- Plot the interpolated data to ensure that the interpolation looks similar to the original data

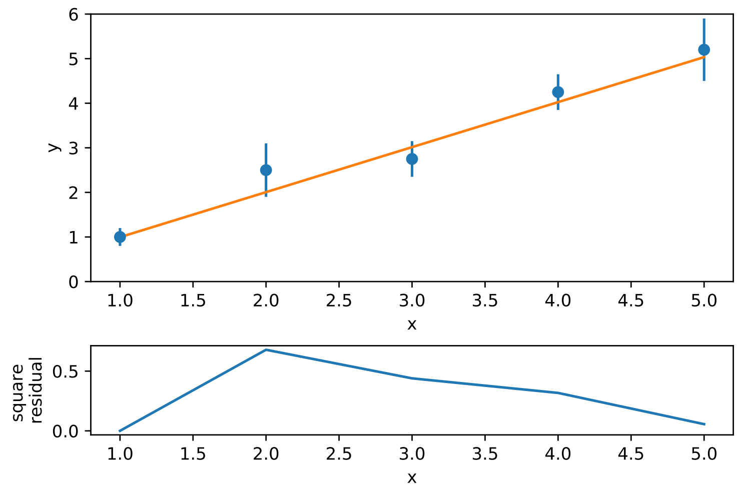

Figure 3: A line of best fit through some datapoints with uncertainty and the squared residuals of the fit.

4 Fitting real data

In the straight line example above, we were not working with "real" data as there was no uncertainty on the value of the data points. However, if we compare the slope of the straight line in Figure 2 with that in Figure 4 (once we consider uncertainty), there is a change from 1.015 to 1.037 a small but significant change. In order to consider the uncertainty, there is a slight modification to the χ^2-value calculation, where the square residual is divided by the uncertainty at the point,

$$\chi^2 = \sum_{i=0}^{N} \left[\frac{y_{\text{model, i}} - y_{\text{exp, i}}}{\mathrm{d}y_{\text{exp, i}}} \right]^2, \tag{1}$$

where, $y_{\text{model, i}}(c)$ is the model (the straight line), $y_{\text{exp, i}}$ is the experimental data point, $\mathrm{d}y_{\text{exp, i}}$ is the uncertainty in the experimental data point, and N is the number of points in the dataset.

Now that all three models have the same x-axis and we are aware of how to fit real data, it is possible to begin the procedure of fitting the experimental data to determine the composition of the mixture. The function that we will be fitting is the following,

$$T_{\text{mixture}} = cT_1 + (1 - c)T_2, \tag{2}$$

where 1 and 2 indicate toluene and benzyl alcohol respectively, and c is the fractional composition of toluene (therefore, $1 - c$ is the fractional composition of benzyl alcohol as we assume that this is only a two component mixture).

Objective 3

- Write a function that evaluates Equation 2, remember to include a docstring
- Test the function by using it to plot the spectra that would result from a 50:50 mixture of the two components against the experimental mixture data

A 50:50 mixture of the two components doesn't do a very good job of modeling the data, therefore we should optimise this value. We can optimise this value by finding the value that minimises the χ^2-value, which can be achieved using the `scipy.optimize` function that was introduced in week 3 called `minimize`.

Objective 4

- Write another function that will evaluate the value of χ^2 for a given c, remember to include a docstring
- Recalling the use of `scipy.optimize.minimize`, try and determine the optimum value to minimise the difference between the experiment and the model
- Store the result of the optimisation and plot the model with the optimised value of c on top of the experimental data to allow you to visually observe the similarity

5 Markov chain Monte Carlo

If we think of uncertainties are representing a distribution of possible values for a data point, for example if the measurement was conducted 10 times, you would get values within this uncertainty. Therefore, we can imagine that if the data in Figure 2 was collected 10 times, there would be some distribution of values, and if we independently fit a straightline to each of these 10 collection, we would get slightly different values for the slope. This distribution of values for the slope, leads to what is referred to as the *inverse uncertainty* which is the uncertainty on a model's parameter that arises due to uncertainty in the measurement that is propagated through the fitting of the model. There are a number of methods that we can use to quantify this inverse uncertainty, including the Markov chain Monte Carlo (MCMC) methods introduced here.

A Markov chain is a *random-walk* through a series of numbers while the Monte Carlo part implies that we use your probability to determine if a *transition* is possible. In MCMC we start with an initial guess for a variable, which can be taken from a traditional minimisation such as that used in `minimize` and then perturb it by some random amount. This random amount is typically obtained based on a step size change with respect to the value of the variable,

```
perturbation = step\_size * np.random.random() * c,
```

where, `c` is the variable to be perturbed. We then we determine if this random perturbation has improved the agreement to the data (the χ^2-value) or not. If it has, we **accept** this new value for our variable and perform another perturbation. If the perturbation does not improve agreement to the data, the new value is **not** immediately rejected, rather it is only rejected if the probability of this *transition* (p) is less than some random number from 0 to 1. The probability is found by,

$$p = \exp\left(\frac{-\chi^2_{\text{new}} + \chi^2}{2}\right), \tag{3}$$

where, χ^2 is the original goodness-of-fit value and χ^2_{new} is the goodness of fit after the perturbation. This means that it is possible for the agreement to get worse overtime, however, the amount by which it can get worse is affected by Equation 3. Due to the application of Equation 3, the only values of χ^2 that may be accessed by the Markov chain (these random perturbations on our value) are those that are statistically feasible given the uncertainty on the experimental measurements.

The algorithm for a typical MCMC sampling process is as follows (this is also described with a flow diagram in Figure 5),

1. Create an empty list called `accepted`
2. Evaluate χ^2 for the initial guess of c, typically this initial guess would be obtained from a standard optimisation
3. Perturb c
4. Calculate χ^2_{new} for the new value of c
5. Determine the probability of this *transition*, from Equation 3
6. Check if $p \geq n$ where n is a random number from 0 to 1 (this can be obtained from `np.random.random()`, note that if the new χ^2 is less than the old one then $p > 1$ and therefore this is always accepted
7. If yes, update values of c and χ^2
8. Go to 3 and repeat until the desired number of iterations has been achieved

Objective 5

- Write a function to perform the Markov chain Monte Carlo algorithm outlined above, this function should return the `accepted` list.
- Plot a histrogram of the `accepted` list (using `plt.hist`), the available values of c should be normally distributed (google a normal distribution and compare)
- Using the statistical functions in `numpy`, calculate the `mean` and standard deviation (`std`) of this distribution.

Hint: Be careful how many iterations you ask you algorithm to perform (you shouldn't ever need more than 1000). Try 100 first to see if it works!

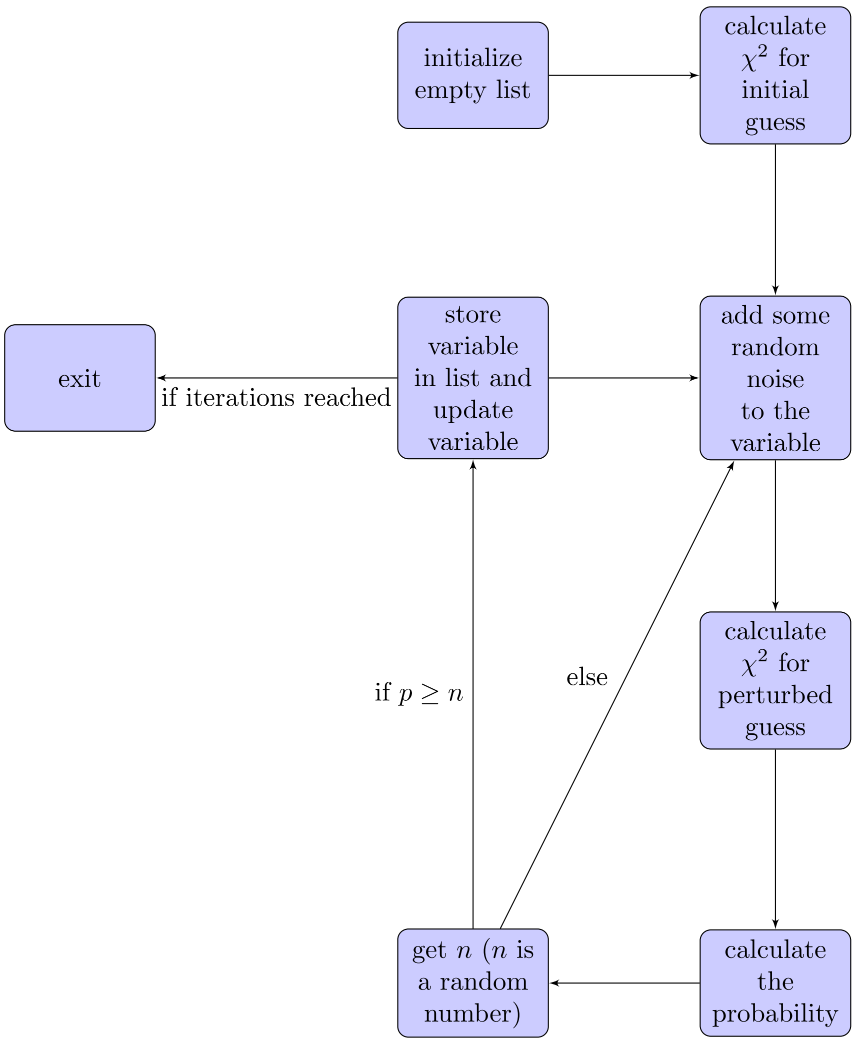
initialize empty list
calculate χ^2 for initial guess
exit
if iterations reached
store variable in list and update variable
add some random noise to the variable
calculate χ^2 for perturbed guess
if $p \geq n$
else
get n (n is a random number)
calculate the probability

Figure 4: A flow diagram describing the MCMC algorithm.