

# Data analysis with Python

Dr Benjamin J. Morgan<sup>1</sup> and Dr Andrew R. McCluskey<sup>1,2</sup>

<sup>1</sup>*Department of Chemistry, University of Bath, email: b.j.morgan@bath.ac.uk*

<sup>2</sup>*Diamond Light Source, email: andrew.mccluskey@diamond.ac.uk*

October 28, 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 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. 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.

## 2 Interpolation

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  $3000\text{ cm}^{-1}$ , and then at higher energy (higher wavenumber), there is a broad feature in the benzyl alcohol data, between  $3250\text{ cm}^{-1}$  to  $3750\text{ cm}^{-1}$ , that can be found in the experimental data. However, you may also note 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. As we will see, in order to compare the datasets, and to evaluate the composition, we must have the wavenumber values at the same points in the  $x$ -axis. 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 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 between the model data on the  $x$ -axis
- Create a new variable called `optimisation_x` that is the range of  $x$ -values to be interpolated over (make it equal to the wavenumber 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

### 3 Fitting

Now that all three models have the same  $x$ -axis, 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,

$$\bar{\nu}_{\text{mixture}} = c\bar{\nu}_1 + (1 - c)\bar{\nu}_2, \quad (1)$$

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 1, 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. In the optimisation of a model to some experimental data, the value that we typically aim to minimise is some goodness-of-fit metric, such as the  $\chi^2$ ,

$$\chi^2 = \sum_{i=0}^N \left[ \frac{y_{\text{model}, i}(c) - y_{\text{exp}, i}}{dy_{\text{exp}, i}} \right]^2, \quad (2)$$

where,  $y_{\text{model}, i}(c)$  is the model transmittance at a particular composition,  $c$ ,  $y_{\text{exp}, i}$  is the experimentally measured transmittance,  $dy_{\text{exp}, i}$  is the uncertainty in the experimentally measured transmittance, and  $N$  is the number of points in the dataset. The closer the value of  $\chi^2$  is to zero, the better the agreement between the model and experiment, in other words we must *minimize* this value.

#### Objective 4

- Write another function that will evaluate the value of  $\chi^2$  for a given  $c$ , remember to include a docstring
- Recalling the use of `scipy.optimize.minimize` from week 3, 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

### 4 Markov chain Monte Carlo

It is the case that all experimental measurements have some associated uncertainty, this has the effect that model parameters (the  $c$  in our model above)

have an *inverse* uncertainty associated with this determination of them. There are a number of ways that we can probe the inverse uncertainty of a parameter, including the Markov chain Monte Carlo (MCMC) methods introduced herein.

The aim of MCMC is to sample the probability distribution of a variable that is statistically relevant given the experiment uncertainties on the measurement. The algorithm for an MCMC sampling process is outlined below,

1. Create an empty list called `accepted`
2. Calculate  $\chi^2$  for the initial guess of  $c$
3. Perturb  $c$  by some random amount (modulate by a `step_size` variable)
4. Calculate  $\chi^2$  for the new value of  $c$
5. Determine the probability of this “transition”
6. Check if  $p \geq n$  where  $n$  is a random number from 0 to 1
7. If yes, update values of  $c$  and  $\chi^2$
8. If beyond burn-in period store  $c$  in accepted list
9. Go to 3 and repeat until the length of the accepted list is equal to the number of iterations required