start

July 22, 2024

1 Documenting Data Analysis

In the presentation supported by this book I want to introduce you to the **power** of using computers to analyze and visualize data. Spreadsheets are great for budgeting but they are difficult to use when data gets "big." And **big data** is here to stay. **Biology** and **chemistry** are rapidly becoming data-intensive fields. The tools to command all this data reside in specialized tools like "***R***" and general-purpose languages like ***Python***.

Most science students have never really commanded a computer. Apps command you, not the other way around. Learning to **interpret and adapt** a computer program for your own purposes is an essential skill in science.

The goal of our program is to explore **chemistry**, not computer science. I will not be teaching **programming**. We will use programs that have been written and adapt them as needed. Along the way the basics of commanding a computer with written instructions will be revealed to you.

1.1 In a Nutshell

The presentation will start with an argument that you can **plot with *python* tools** just as effectively as in Microsoft excel. Then we will digress to plead that you never commit one of Excels greatest sins, using the smoothed line to connect data points. One of the most important ideas in data analysis is correct **propagation of experimental error**. You will be introduced to the **uncertainties package** in *Python* that will handle your errors for you. Finally we will explore using *Python* plotting tools and the uncertainties package to analyze kinetic data using the Erying equation. At every step the **self-documenting** nature of *Python* code will be emphasized.

1.2 Steal it All

We will use *Python* only as a **tool**. This book is intended to provide some basic **examples** of *Python* code for performing **calculations and visualizing data**. All code for the demonstrations in this presentation is already written for you. Steal it from me; steal it from the internet; give your code to friends. Always **give credit** where credit is due. Examine, play with, and **steal the code** in the interactive notebooks linked below. To learn, you must first imitate. Later, you will inovate. The journey from here to there is up to you.

1.3 Notebooks for the Presentation

The python notebooks described below contain all the code and documentation for the plots and data sets used in the presentation. Steal it run with it.

1.4 Part 1: Quick Examples of *Python*

This notebook provides some simple examples of data analysis in *Python*. We will use the example of the Michaelis-Menten plot to highlight the dangers in linearization for data analysis.

> **Title**: Part 1: Quick Examples *Python* **Tools**: *Python* math operators, *NumPy*, *SciPy*, *matplotlib* **Skills**: Basic math operations and math functions using the tools of *NumPy*. Math with *NumPy* arrays, formatting numbers in f-strings. Accessing physical constants in *SciPy*. Data analysis tools in *SciPy*. Simple data visualization using *matplotlib*.

1.5 Part 2: Please Don't Use "Smooth Lines"

Some data is not meant to be fit to a mathematical model. How should these data sets be presented? Quite often a simple plot with points connected by lines will do. Just don't chose the "smooth line" option. Along the way we will explore some of the smooth line options available via *Python* and spline functions. And we will then make a promise to never use them.

> **Title**: Part 2: Please Don't Use "Smooth Lines" **Tools**: *NumPy*, *SciPy*, *matplotlib*, *Pandas* **Skills**: Importing data from a csv file using the *Pandas* library. Basic math operations and math functions using the tools of *NumPy*. Math with *NumPy* arrays, formatting numbers in f-strings. Simple data visualization using *matplotlib*. Data interpolation using fits to spline function in the `SciPy.interpolate` sublibrary

1.6 Part 3: Uncertainty and Confidence Intervals

The Erying plot is a classic way to explore how *Python* tools can assist in analysis. Experimental data is often known only to within a range that defines it prescission. This experimental error will be reflected in parameters calculated from the mathematical model used in the analysis. The error will be transmitted through any calculation using these values. If we use the uncertainty of the parameters to calculate the prescission of the predicted results of the model then we will have the confidence interval across the range of experimental data. This can be plotted as well to provide a visual indication of the experimental error within the model.

Error propagation can get complicated. But we will use a set of tools provided by the *Uncertainties* library to handle all of this automatically. We will also use a robust and extremely computationaly expensive method to obtain a more "real-world" estimate of the confidence interval.

> **Title**: Part 3: The Eyring Equation and Experimental Error

1.6.1 Part 3A: A Simple Data Analysis

This notebook will plot a 5-point Erying plot and determine the $\Delta H^{\ddagger}$ and $\Delta S^{\ddagger}$ for the reaction. The standard deviations for the parameters will be examined and used to calculate the confidence interval for the rate constant at a given temperature. The `scipy.stats.linregress()` tool will be demonstrated along with the `matplotlib.pyplot` library for plotting. Along the way we will learn the correct way to use the *uncertainties* module in our calculations.

> **Title**: Part 3A: A Simple Data Analysis **Tools**: *NumPy*, *SciPy*, *matplotlib*, *Uncertainties.ufloat*, *Uncertainties.unumpy* **Skills**: Math functions using the tools of *NumPy*. Math with *NumPy* arrays, formatting numbers in f-strings. Simple data visualization using *matplotlib*. Data interpolation using linear regression with `SciPy.stats.linregress()`. Creating uncertain values with

`Uncertainties.ufloat()`. The consequences of strongly correlated errors in fit parameters (covariance) and ways to deal with this effect.

1.6.2 Part 3B: Curve Fitting and Covariance

The `scipy.optimize` sublibrary provides a better set of tools for fitting data to models and interfacing with the *uncertainties* package. We will be using `scipy.optimize.curvefit()` in this exercise and using the covariance matrix returned by that function to create ufloat values that reflect the coupling between errors in fit parameters.

> **Title**: Part 3B: Curve Fitting and Covariance **Tools**: *NumPy*, *SciPy*, *matplotlib*, *Uncertainties.correlated_values*, , *Uncertainties.unumpy* **Skills**: Math functions using the tools of *NumPy*. Math with *NumPy* arrays, formatting numbers in f-strings. Data visualization using *matplotlib*. Using `SciPy.optimize.curve_fit()` to fit data to arbitrary mathematical models. Creating uncertain values with `Uncertainties.correlated_values()`. Propagating error and determining confidence intervals

1.6.3 Part 3C: Using LMFit

The *lmfit* library provides another set of tools for handling data and interfacing with the *uncertainties* package. It has many built-in tools for analyzing and visualizing the curve fit. It returns ufloat values for fit parameters directly that already contain information on any covariance.

> **Title**: Part 3C: Using LMFit **Tools**: *NumPy*, *SciPy*, *matplotlib*, *Uncertainties*, *LMFit* **Skills**: Math functions using the tools of *NumPy*. Math with *NumPy* arrays, formatting numbers in f-strings. Data visualization using *matplotlib*. Using *LMFit* to fit data to arbitrary mathematical models and plot data. Propagating error and determining confidence intervals

1.6.4 Part 3D: Reading Data

This notebook will read in data from a text file. Creating data files for your experimental data will allow you to use a notebook to analyze a new set of data by changing the file name and nothing else. This will enable your data analysis notebook to be more versatile. This notebook is almost identical to Part 3B except that the data is read from a file rather than entered directly. Here we use the full dfata set from the publication and can see the result of increasing data on prescission.

> **Title**: Part 3D: Reading Data **Tools**: *NumPy*, *SciPy*, *matplotlib*, *Pandas*, *Uncertainties* **Skills**: Importing data from a csv file using the *Pandas* library. Math functions using the tools of *NumPy*. Math with *NumPy* arrays, formatting numbers in f-strings. Data visualization using *matplotlib*. Using `SciPy.optimize.curve_fit()` to fit data to arbitrary mathematical models. Creating uncertain values with `Uncertainties.correlated_values()`. Propagating error and determining confidence intervals

1.6.5 Part 3E: Bootstrapping Confidence Intervals

The confidence intervals produced by `lmfit` or by using the covariance matrix from `curve_fit` follow typical rules for error propagation and present a range that is symmetrical above and below any value. This is the common $a \pm b$ way of presenting error. However, the confidence range may not be symmetrical (this is especially true sparse data sets.) One way to express this is to use more sophisticated error propagation tools in *python* such as `soerp` or `mcerp`. We won't be exploring those here. In this notebook we will use a robust but inefficient method for determining the confidence interval for a data set and generate a confidence band on a plot that reflects the "real world" error in your data. This method is called "bootstrapping."

> **Title**: Part 3E: Bootstrapping Confidence Intervals **Tools**: *NumPy*, *SciPy*, *matplotlib*, *Pandas*, *Uncertainties* **Skills**: Importing data from a csv file using the *Pandas* library. Math functions using the tools of *NumPy*. Math with *NumPy* arrays, formatting numbers in f-strings. Data visualization using *matplotlib*. Using `SciPy.optimize.curve_fit` to fit data to arbitrary mathematical models. Creating uncertain values with `Uncertainties.correlated_values`. Propagating error and determining confidence intervals

1.7 Part 4: Pick a Style

Journals often have a specific style for plots. In a thesis, you should use the same style for all plots. In this notebook we will explore styling plots using many option available in the `matplotlib.pyplot` library. Once you get a style you like in a notebook, you never need to change. I haven't changed my style since 1985, that why I look so good.

> **Title**: Part 4: Styling Plots" **Tools**: *matplotlib* **Skills**: Applying style files to plots. using standard styles and styles provided by journals.

These is a **Juptyter-book** that was built from a set of **interactive** ***Python*** **Jupyter notebooks**. The original notebook for any given chapter can be obtained using the **download link** at the top of the page.