Chemical Thermodynamics, Equilibrium and Kinetics *Dr. Andrew Klose*

Curve Fitting with Gas Laws

Objectives

In this programming lab, experimental data relating to gas-phase equations of state will be reproduced with ideal and non-ideal gas law models. The `Python` computer language and `Jupyter Notebooks` will be used to perform the analysis. Expected learning outcomes include becoming familiar with and being able to explain fundamental differences between various equations of state for gas-phase species. Additionally, you will learn basic computer programming skills relating to the use of arrays, reading in experimental data, creating functions, fitting functions to data, and plotting curves.

Pre-lab Assignment

Search scientific literature for pressure vs volume data for atoms/small molecules at various isotherms (constant temperature). Find these P-V data for a particular molecule at three different temperatures; there should roughly 25 (or more) data points for each temperature.

Introductory note regarding data analysis software: this course requires data analysis that may be relative complex. The examples I will give throughout the course are written for `Python`. I will provide detailed guidance for using this software throughout the course.

The chemistry computer lab has `Python` installed on all the machines, but it is suggested that you install it on your personal computer. To install, download `Anaconda` (Python 3.7 version) from https://www.anaconda.com/products/individual. You should install the "individual" version; the "Graphical Installer" is the simplest way to install the software (for both Windows and MacOS) After installation and computer restart, start the Anaconda Navigator and open up the "Jupyter-Lab" application.

Pre-lab Discussion Questions

1. How can one quantitatively determine the goodness of a fit of a function to a data set? For example, say you have data points (1,1), (2,5), (3,8), and (4,16), and want fit the function $y = ax^2 + bx + c$ to the data. This means you want to vary the values of a, b, and c until you get a function that best reproduces the data values. What procedures can you use to quantitatively determine how well the function reproduces the data for given values of the parameters?

2. The ideal gas law is an example of an equation of state. What assumptions are used in the ideal gas law. In what conditions (*i.e.* real conditions) do the assumptions fail?

3. The Van der Walls equation in an example of an equation of state for real gases. The Van der Waals equation is given as

$$\left(P + \frac{n^2 a}{V^2}\right)(V - nb) = nRT, \tag{1}$$

   where P is the pressure, n is the amount of gas, V is the volume of the gas, R is the ideal gas constant, and T is the temperature. Additionally, a and b are coefficients specific to a particular gas sample; what is the purpose for each of these coefficients? (Things to consider: how is the Van der Waals equation different from the ideal gas law? What are the units of a and b? What are the assumptions of the ideal gas law?)

1 Python Basics

You can script multiple commands by creating a new file, putting all the commands that you would like to run in sequential order, and then running the script. All lines of code will execute, and any outputted results will be displayed graphically below the script/cell. The code described here is in a `Jupyter Notebook` file, (.ipynb extension), and is called `CurveFitting-Part1.ipynb`. Open this file in `JupyterLab`. **Note that all notebook files and data files (used later on) need to be placed in the same directory locally on your computer.** The short lines of code are setup in "cells" that you execute by clicking on and pressing SHIFT + ENTER. Below, all of the contents of the Part 1 notebook file are discussed.

When executing a script, you often need methods and functions that are pre-defined in so-called libraries. For example, how would `Python` know what square root means or how to make a plot? You need to load libraries that have functions to do such things. As such, you "import" them:

```
import matplotlib.pyplot as plt
import numpy as np
```

Let's say that you want to plot the vapor pressure as a function of temperature for the gas you picked. Think of what you would do in Microsoft Excel or Google Sheets. First you would need to enter the data into the spreadsheet (we call spreadsheets "arrays" in computer science) and then you would insert a plot.

Assume you have the following (x, y) data points: (1,2),(2,3),(3,4),(4,5),(5,6). If using a spreadsheet application, you might make one column called "X Values" and have cells with 1, 2, 3, 4, and 5 along with a second column called "Y-values" with cells 2, 3, 4, 5, and 6. Entering the data into arrays in `Python` can be accomplished using the following commands:

```
Xvals= np.array([1,2,3,4,5])
Yvals = np.array([2,3,4,5,6])
```

where `Xvals` and `Yvals` are the relevant temperature and pressure arrays, respectively. The data can be plotted by typing:

```
plt.plot(Xvals,Yvals)
```

for a "connect-the-dots" line plot, or

```
plt.scatter(Xvals,Yvals)
```

for a scatter plot.

You can manipulate the arrays using algebraic expressions. For example, to compute $y = x^2$, type:

```
Yvals2 = Xvals**2
```

Now, a new array, `Yvals2`, has been created. You can print this array by typing:

```
print Yvals2
```

Often times, it is advantageous to have a single 2-dimensional array for X and Y values. One can make a 2D array with 2 columns by putting in the (x,y) elements pairwise:

```
XY2Dcols = np.array([[1,2],[2,5],[3,10],[4,17],[5,26]])
```

A 2D array with 2 rows can be constructed by putting the x and y rows in as:

```
XY2Drows = np.array([[1,2,3,4,5],[3,6,11,18,27]])
```

To access element (row$_i$,col$_j$), just type:

```
XY2Dcols[i,j]
```

Use a colon to access an entire row i or column j, type:

```
XY2Drows[i,:]
XY2Dcols[:,j]
```

These commands will give all elements in row i or column j, respectively. You can then plot using the 2D arrays:

```
plt.scatter(XY2Dcols[:,0],XY2Dcols[:,1])
```

for the 2 column array, or

```
plt.scatter(XY2Drows[0,:],XY2Drows[1,:])
```

for the 2 row array.

2 Fitting your data with the van der Waals equation

We want to read in our P-V data and, ultimately, fit those data with real gas law equations of state.

In `CurveFitting-Part2.ipynb`, a data array, called `DataArray` is loaded from the comma-separated file `DataFile`,

```
DataFile = 'CO2-pv-150degC.csv'
```

by typing the command

```
DataArray = np.genfromtxt(DataFile,skip_header=2,delimiter=",")
```

The argument `skip_header` indicates that the header of the file is 3 lines long, and the data start on line 4 of the file. `Delimiter` is the character that separates entries in each row; in this case it is a comma. Open the .csv file in Wordpad, (or using the Jupyter Notebook application), to visually inspect the format of the file.

The resulting `DataArray` is a 2-dimensional array, and the X values (temperature) and Y values (pressure) can be defined in terms of 1-dimensional arrays as

```
Xvals = DataArray[:,0]
Yvals = DataArray[:,1]
```

Now, this notebook contains the van der Waals equation:

$$\left(P + \frac{n^2 a}{V^2}\right)(V - nb) = nRT. \tag{2}$$

The function is programmed like this:

```
def vdW(x,a,b):
    return (R*T)/(x-b) - (a)(x**2)
```

where x here is the MOLAR volume, and the returned value is the pressure. Note that $**$ in `Python` is the power character. e.g. `x**2` is x^2. Also, note that you will have to define the ideal gas constant, R, in appropriate units, as well as T, and the molar volume. Note that in the fitting routine, T and R are constants, and only a and b change.

In the van der Waals function, the `def` command defines the function, in this case called `vdW`. Please note that the indent is important (indentation is part of the `Python` language). The function needs three parameters - `x, a, and b` - passed to it.

Before we fit the function to our data, we need to give initial guesses for the a and b

```
p0 = np.array([3.6,0.04])
```

where `p0` is our array of guesses of 3.6 and 0.04 for for a and b, respectively. Python has a defined least-squares fitting routine, and we included this at the beginning of the file by typing

```
from scipy.optimize import curve_fit
```

Now to fit the function to the data we execute:

```
popt,pcov = curve_fit(vdW,Xvals,Yvals,p0)
```

where `popt` is an array of the optimum parameter values, and `pcov` is the estimated covariance matrix of the optimum parameters. The arguments passed to `curve fit` include:

1. The function to fit
2. X-values of the data
3. Y-values of the data
4. Array of initial guesses for parameters

We can output the optimum parameters to text by typing

```
print(popt)
```

and plot the fitting function by

```
plt.plot(Xvals,vdW(Xvals,popt[0],popt[1]),color="red")
```

Fit the example P-V data sets provided for carbon dioxide. Then, fit your own data with the van der Waals equation. Once you are done, you will need to fit your PC data to another real equation of state (*e.g.* the Virial equation). Follow the prompts in the lab report section below for considerations of what to investigate and report.