VCM.py Documentation

Purpose
This document is meant to explain the design and development of VCM.py – a python program created to interpolate the saturation pressure of vinyl chloride monomer (VCM) from a data set of experimental measurements.

This document assumes a basic understanding of scientific notation, thermodynamics, statistics, linear algebra, and calculus. This document also assumes a basic understanding of python concepts: lists, list functions, lambda functions, numpy arrays, numpy modules, etc.

Background
This program, along with many others, was created to simulate the separation process of a flash evaporator unit. The inlet stream, also know as the feed, was specified to be a saturated liquid mixture of four chemical species: vinyl chloride monomer, isobutane, 1-butene, and cis-2-butene.

To simulate this process, it was necessary to interpolate the saturation pressure of each feed component at a given operating temperature from a data set of experimental measurements. The data set for vinyl chloride monomer (VCM) is presented below.

Index	1	2	3	4	5	6	7
Temperature (K)	140	160	180	200	220	240	260
Saturation Pressure (Bar)	2.03E-05	5.04E-04	5.41E-03	3.30E-02	1.35E-01	4.16E-01	1.04

Index	8	9	10	11	12	13	14	15
Temperature (K)	280	300	320	340	360	380	400	420
Saturation Pressure (Bar)	2.20	4.15	7.12	11.4	17.2	24.9	34.8	47.5

Model Selection & Variable Formatting

In accordance with Antoine's Equation, I decided to use a quadratic regression model to relate the natural logarithm of vapor pressure and the inverse of temperature. To reflect this, the variables and functions used in the program were formatted as:

Variable Name	Variable Type	Meaning of Value(s)
T	List	Temperatures at which saturation measurements were taken
P	List	Saturation pressure measurements.
X	NumPy Array	Inverse of temperature values in T
Xs	NumPy Array	Second power of values in X
Xc	NumPy Array	Third power of values in X
Xq	NumPy Array	Fourth power of values in X
Y	NumPy Array	Natural logarithm of saturation pressure values in P
Sx_0	Floating Point Number	Length of X
Sx_1	Floating Point Number	Sum of entries of X
Sx_2	Floating Point Number	Sum of entries of Xs
Sx_3	Floating Point Number	Sum of entries of Xc
Sx_4	Floating Point Number	Sum of entries of Xq
Sy	Floating Point Number	Sum of entries of Y
Syx_1	Floating Point Number	Dot product of Y and X
Syx_2	Floating Point Number	Dot product of Y and Xs
M	NumPy Array	Coefficient Matrix – Least Squares Linear System
d	NumPy Array	Constant Vector – Least Squares Linear System
Minv	NumPy Array	Inverse matrix of M
Z	NumPy Array	Solution Vector – Least Squares Linear System
A	Floating Point Number	Second Order Coefficient – Quadratric Regression Model
B	Floating Point Number	First Order Coefficient – Quadratic Regression Model
C	Floating Point Number	Zeroth Order Coefficient – Quadratric Regression Model

Function Name	Dummy Variable	Output
f	x	x^(-1)
g	x	ln(x)
s	x	x^2
c	x	x^3
q	x	x^4

Quadratric Regression: Least Squares Minimization & Linear System of Equations
With the variable formats selected, the quadratic regression model can be expressed as:

$$x = \frac{1}{T_{operation}}$$
$$y = Ax^2 + Bx + C$$
$$y = \ln\left(P_{saturation}\right)$$

The values of the coefficients are determined from the least squares method. That is to say, we determine the coefficient values that minimize the following function:

$$F\left(A, B, C\right) = \sum_{j=0}^{N=14} \left(Y_j - AX_j^2 - BX_j - C\right)^2$$

According to the principles of minimization, the gradient of this function equals the zero vector at the minimizing coefficient values. By setting the gradient equal to the zero vector, one obtains a linear system equations that can be solved to obtain the desired coefficient values.

This linear system was simplified and is presented in matrix form below:

$$M = \begin{bmatrix} \sum_{j=0}^{N=14} X_j^4 & \sum_{j=0}^{N=14} X_j^3 & \sum_{j=0}^{N=14} X_j^2 \\ \sum_{j=0}^{N=14} X_j^3 & \sum_{j=0}^{N=14} X_j^2 & \sum_{j=0}^{N=14} X_j \\ \sum_{j=0}^{N=14} X_j^2 & \sum_{j=0}^{N=14} X_j & \sum_{j=0}^{N=14} X_j^0 \end{bmatrix} = \begin{bmatrix} Sx_4 & Sx_3 & Sx_2 \\ Sx_3 & Sx_2 & Sx_1 \\ Sx_2 & Sx_1 & Sx_0 \end{bmatrix}$$

$$d = \begin{bmatrix} \sum_{j=0}^{N=14} Y_j X_j^2 \\ \sum_{j=0}^{N=14} Y_j X_j \\ \sum_{j=0}^{N=14} Y_j \end{bmatrix} = \begin{bmatrix} Syx_2 \\ Syx_1 \\ Sy \end{bmatrix}$$

$$z = \begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

$$Mz = d$$

This system can be solved by computing the inverse matrix of M and using the following formula:

$$z = M_{inv} d$$

Code

```
import numpy as np
import math

T = [140,160,180,200,220,240,260,280,300,320,340,360,380,400,420]
P = [2.03e-5,5.04e-4,5.41e-3,3.3e-2,0.135,0.416,1.04,2.20,4.15,7.12,11.4,17.2,24.9,34.8,47.5]

f = lambda x: x**(-1)
g = lambda x: math.log(x)

s = lambda x: x**2
c = lambda x: x**3
q = lambda x: x**4

X = np.array(list(map(f,T)))
Xs = np.array(list(map(s,X)))
Xc = np.array(list(map(c,X)))
Xq = np.array(list(map(q,X)))
Y = np.array(list(map(g,P)))

Sx_0 = len(X)
Sx_1 = sum(X)
Sx_2 = sum(Xs)
Sx_3 = sum(Xc)
Sx_4 = sum(Xq)
Sy = sum(Y)
Syx_1 = Y @ X
Syx_2 = Y @ Xs

M = np.array([[Sx_4,Sx_3,Sx_2],[Sx_3,Sx_2,Sx_1],[Sx_2,Sx_1,Sx_0]])
Minv = np.linalg.inv(M)
d = np.array([[Syx_2],[Syx_1],[Sy]])

z = Minv @ d
A = z[0][0]
B = z[1][0]
C = z[2][0]

def Psat1(T):
    x = T**(-1)
    y = (A)*(x**2) + (B*x) + C
    p = math.exp(y)
    return p
```

Code Breakdown

Line 1: numpy module is imported as np
Line 2: math module is imported

Line 4: T is defined using values from data set
Line 5: P is defined using values from data set

Line 7: f is defined as a lambda function
Line 8: g is defined as a lambda function using math module

Line 10: s is defined as a lambda function
Line 11: c is defined as a lambda function
Line 12: q is defined as a lambda function

Line 14: The function f is mapped to T. The list of the map's outputs is used to construct a numpy array. This numpy array is used to define X.
Line 15: The function s is mapped to X. The list of the map's outputs is used to construct a numpy array. This numpy array is used to define Xs.
Line 16: The function c is mapped to X. The list of the map's outputs is used to construct a numpy array. This numpy array is used to define Xc.
Line 17: The function q is mapped to X. The list of the map's outputs is used to construct a numpy array. This numpy array is used to define Xq.
Line 18: The function g is mapped to P. The list of the map's outputs is used to construct a numpy array. This numpy array is used to define Y.

Line 20: Sx_0 is defined as a length of X
Line 21: Sx_1 is defined as the sum of the entries of X
Line 22: Sx_2 is defined as the sum of the entries of Xs
Line 23: Sx_3 is defined as the sum of the entries of Xc
Line 24: Sx_4 is defined as the sum of the entries of Xq
Line 25: Sy is defined as the sum of the entries of Y
Line 26: Syx_1 is defined as the dot product of Y and X
Line 27: Syx_2 is defined as the dot product of Y and Xs

Line 29: M is defined as numpy array using the values of Sx_0, Sx_1, Sx_2, Sx_3, and Sx_4
Line 30: The inverse of M is calculated using the linalg.inv function of the numpy module. This matrix is used to define Minv
Line 31: d is defined as a numpy array using the values of Sy, Sy_1, and Sy_2

Line 33: z is defined as the matrix multiplication of the Minv and d
Line 34: A is defined as the first entry of z
Line 35: B is defined as the second entry of z
Line 36: C is defined as the third entry of z

Line 38: See next section for description of code from this line forward.

Example Interpolation Function

This program includes a definition of the interpolation function created from the quadratic regression model. It is important to note that this is included for demonstration purposes only.

Since the coefficient values never change, it is inefficient to calculate them every time the interpolation function is evaluated. For this case in particular, the coefficients are calculated by matrix inversion, which is computationally expensive.

In practice, the coefficient values obtained in lines 34 – 36 would be recorded. These values would be used to define the function in a separate program.

In fact, it would be most efficient to create a function that returns the saturation pressures of each feed component from a single operating temperature input. Of course, it would be necessary to determine the quadratic regression model coefficients for the other species.

That being said, I provide a breakdown of the code for the example function:

Line 38: A function Psat1 is defined as a single variable function with argument T. This argument refers to the operating temperature of the flash evaporator unit.
Line 39: A local variable x is defined as the inverse of the value of T.
Line 40: The quadratic regression model is evaluated at the value of x. The output is used to define a local variable y.
Line 41: The exponential function is evaluated at the value of y. The output is used to define a local variable p.
Line 42: The return of the function is defined as the value of p.