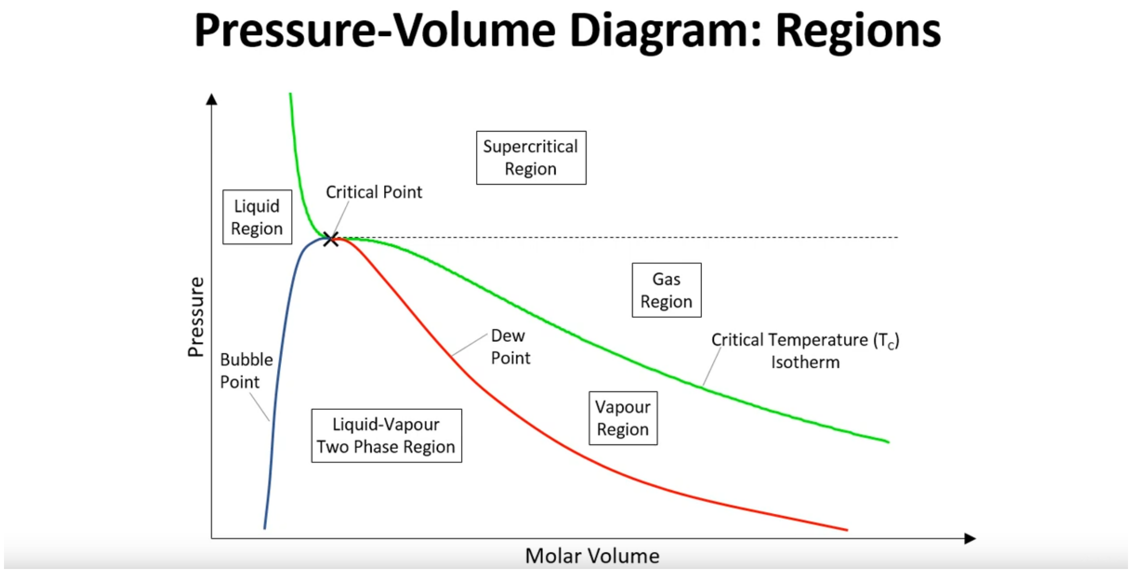
student_example

March 24, 2023

1 Thermo B - Student Exercises

2 Pressure-Volume Diagram



In a pressure-volume diagram, each region corresponds to the range of combinations of temperature and pressure over which that phase is stable 1

Liquid Region: The liquid region would be where the substance is in its liquid phase at a given combination of temperature and pressure.

Liquid-Vapor Region: The liquid-vapor region on a pressure-volume diagram represents the range of values where both liquid and vapor phases can coexist at equilibrium

Vapour Region: The vapour region would be where the substance is in its vapour phase at a given combination of temperature and pressure.

Gas Region: The gas region would be where the substance is in its gas phase at a given combination of temperature and pressure.

Supercritical Region: In a pressure-temperature phase diagram, the supercritical region is where the liquid and gas phases disappear to become a single supercritical phase 1. This occurs at conditions beyond the critical point, where temperature and pressure are both higher than their respective critical values.

2.1 Ideal Gas Law

The ideal gas law is a mathematical expression of the state of a hypothetical ideal gas, derived from the kinetic theory of gases and named by the French physicist Jacques Charles in 1787 1. It is a special case of the general gas law, which describes the state of any gas.

$$PV = nRT$$

where:

- P is the pressure of the gas
- V is the volume of the gas
- n is the number of moles of gas
- R is the ideal gas constant
- T is the temperature of the gas

2.1.1 Task 1: Idea Gas Law Calculation

Calculate the theoretical pressure of the gas, CO2, at various temperatures and volumes.

> **Python**: For this task, you will need to use the ideal gas law function from the `thermo` module. The function is called `ideal_gas_law` and takes the following arguments: `ideal_gas_law(gas_constant: float, list_of_volumes: List, list_of_temperatures: List)`. The function returns a Dataframe with the calculated pressures for each combination of temperature and volume. The index of the Dataframe is the volume and the columns are the temperatures.

2.1.2 Task 2: Ideal Gas Law Plot and Critical Point

In this task you will need to on the same plot: 1. Plot the pressure of the gas, CO2, at various temperatures and volumes using the Dataframe from Task 1. 2. Plot the critical point of the gas, CO2, on the pressure-volume diagram.

> **Python**: For this task, you will need to use the `plot_ideal_gas_law` function from the `thermo` module. The function is called `plot_ideal_gas_law` and takes the following arguments: `plot_ideal_gas_law(ideal_gas_law_df: pd.DataFrame)`. The function returns a plot of the Dataframe. However, the plot will not be shown in the notebook. You will need to use the `plt.show()` function to display the plot. Additionally, you will need to use the `plt.figure(figsize=(width, height))` function to set the size of the plot. Recommended is a width of 10 and a height of 10. You will also need to use the `plt.title()` function to set the title of the plot. You can use the `plt.xlabel()` and `plt.ylabel()` functions to set the x and y axis labels.
>
> ***Tip***: Before calling `plt.show()`, ensure you have plotted both the ideal gas law and the critical point on the same plot.
>
> ***Tip***: You can get the docstring for a function by using the ? symbol after the function name

[1]:
```
# Call the plot_ideal_gas_law function with the appropriate argument - You will␣
↪not need to set it as a variable
#
%%script false --no-raise-error
plot_ideal_gas_law()
```

```
UsageError: Line magic function `%%script` not found.
```

As it can bve seen, the Critical Point of CO2 is nowhere near the ideal gas law for temperature 304.1K

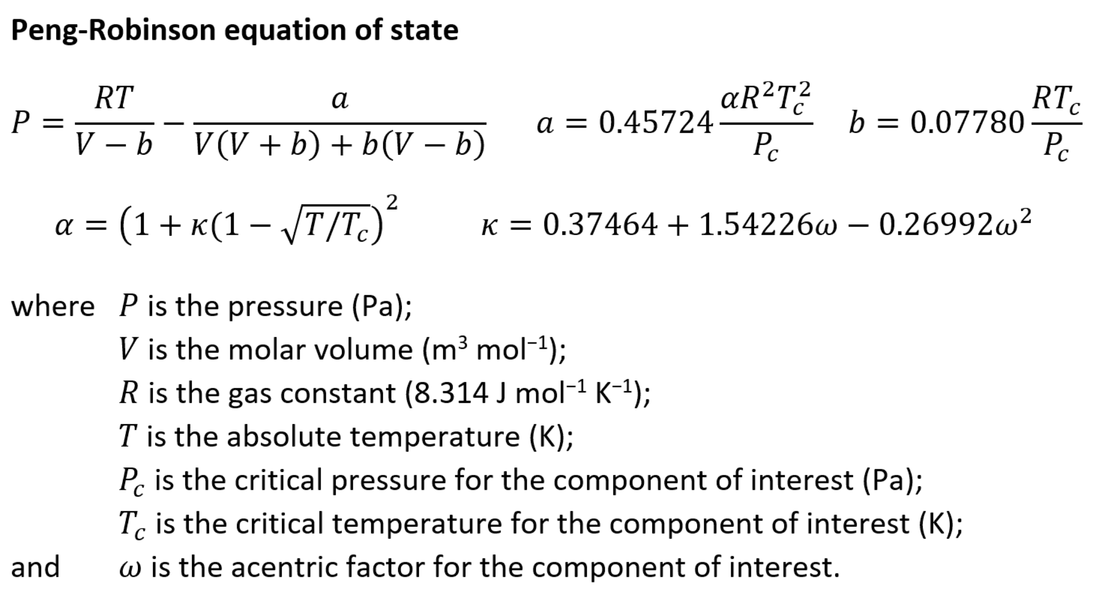
Assumptions:

- The gas is a perfect gas
- The volume of the gas is negligible compared to the volume of the container
- There is no intermolecular attraction or repulsion between the gas molecules

Limitations:

- The ideal gas law is only valid for a small range of temperatures and pressures
- It is unrealiable at high pressures and low temperatures
- Cannot predict vapur-liquid coexistence

2.2 Peng-Robinson Equation of State



Peng-Robinson equation of state

$$P = \frac{RT}{V-b} - \frac{a}{V(V+b)+b(V-b)} \qquad a = 0.45724\frac{\alpha R^2 T_c^2}{P_c} \qquad b = 0.07780\frac{RT_c}{P_c}$$

$$\alpha = \left(1+\kappa(1-\sqrt{T/T_c}\right)^2 \qquad \kappa = 0.37464 + 1.54226\omega - 0.26992\omega^2$$

where P is the pressure (Pa);
V is the molar volume (m^3 mol^{-1});
R is the gas constant (8.314 J mol^{-1} K^{-1});
T is the absolute temperature (K);
P_c is the critical pressure for the component of interest (Pa);
T_c is the critical temperature for the component of interest (K);
and ω is the acentric factor for the component of interest.

The Peng-Robinson equation of state (PR EOS) is a cubic equation of state that was developed in 1976 at The University of Alberta by Ding-Yu Peng and Donald Robinson [1]. It is used to predict the behavior of fluids, particularly natural gas systems [2].

The PR EOS is more accurate than the Ideal Gas Law for predicting the behavior of real gases because it takes into account molecular interactions and volume [3].

2.2.1 Task 3: Peng-Robinson Equation of State Calculation

Calculate the theoretical pressure of the gas, CO2, at various temperatures and volumes using the Peng-Robinson Equation of State.

> **Python**: For this task, you will need to use the Peng-Robinson Equation of State function from the `thermo` module. The function is called `peng_robinson_eos` and takes the following arguments: `peng_robinson_eos(gas_constant: float, list_of_volumes: List, list_of_temperatures: List, kappa: float, bc: float, ac: float, alpha: Callable)`. The function returns a Dataframe with the calculated pressures for each combination of temperature and volume. The index of the Dataframe is the volume and the columns are the temperatures.

Kappa The kappa parameter is a dimensionless parameter that is calculated from the acentric factor of the substance. The acentric factor is used to measure the non-sphericity of a molecule. The kappa parameter is calculated using the following equation:

$$\kappa = 0.37464 + 1.54226\omega - 0.26992\omega^2$$

where:

- ω is the acentric factor of the substance

Alpha The alpha function is a function of temperature that is used to calculate the acentric factor of the gas. The acentric factor is a measure of the deviation of the gas from ideal gas behavior. The acentric factor is used to calculate the a and b constants of the Peng-Robinson Equation of State. The alpha function is defined as:

$$\alpha(T) = \left(1 + \kappa\left(1 - \sqrt{\frac{T}{T_c}}\right)\right)^2$$

where:

- T is the temperature
- κ is the acentric factor
- T_c is the critical temperature

As we already know κ and T_c, we can create a function that takes in T and returns the value of alpha:

Two ways to do this - *lambda* or defining a *function*

You can run either Cell below

a variable The a variable is defined as:

$$a = 0.45724\frac{\alpha R^2 T_c^2}{P_c}$$

where: * α is the alpha function * R is the ideal gas constant * T_c is the critical temperature * P_c is the critical pressure

b variable The b variable is defined as:

$$b = 0.07780 \frac{RT_c}{P_c}$$

where:

- R is the ideal gas constant
- T_c is the critical temperature
- P_c is the critical pressure

Peng-Robinson Equation of State Function Now that you have the required variables for the Peng-Robinson Equation of State function, you can create the function.

2.2.2 Task 4: Peng-Robinson Equation of State Plot

In this task you will need to on the same plot:

1. Plot the pressure of the gas, CO2, at various temperatures and volumes using the Dataframe from Task 3
2. Plot the critical point of the gas, CO2 on the pressure-volume diagram

   Python: For this task, you will need to use the `plot_peng_robinson_eos` function from the `thermo` module. The function takes the following arguments: `plot_peng_robinson_eos(peng_robinson_eos_df: pd.DataFrame)`. The function returns a plot of the Dataframe. However, the plot will not be shown in the notebook. You will need to use the `plt.show()` function to display the plot. Additionally, you will need to use the `plt.figure(figsize(width, height))` function to set the size of the plot. Recommended is a width of 10 and a height of 10. You will also need to use the `plt.title()`. function to set the title of the plot. You can use the `plt.xlabel()` and `plt.ylabel()` functions to set the x and y axis labels.

   Tip: Before calling `plt.show()`, ensure you have plotted both the ideal gas law and the critical point on the same plot.

   Tip: You can get the docstring for a function by using the ? symbol after the function name

2.2.3 Peng-Robinson Equation of State Polynomial Form

In a more compact and dimensionless form, the Peng-Robinson EOS can be restated as a cubic in Z. This is the polynomial form of the Peng Robinson Equation of State. This sets up a cubic equation in Z, which has three real roots (as opposed to being imaginary). The largest root is the value of Z_v; the smallest root is the value of Z_L; and the third root is discarded as having no physical meaning.

So how do we solve for the roots? In Python, we will use the module `CubicEquationSolver` which contains the function `solve(a, b, c, d)` which takes the coefficients of the cubic equation and returns the roots in an array.

Polynomial Structure:

$$ax^3 + bx^2 + cx + d = 0$$

Breaking up the equation: Z^3 Define the variable Z_3 as the coefficient of Z^3 from the polynomial

Breaking up the equation: Z^2 Define the variable Z_2 as the coefficient of Z^2 from the polynomial

Tip: $-(1-B)$

Tip: $B = \frac{bP}{RT}$

Tip: Use the variable b from the previous task

Breaking up the equation: Z_1 Define the variable Z_1 as the coefficient of Z^1 from the polynomial

Tip: $(A - 2B - 3B^2)$

Tip: $A = \frac{aP}{R^2T^2}$

Tip: a is the variable from the previous task **BUT it must be adjusted due to the alpha function, which is a function of T!**

Breaking up the equation: Z_0 Define the variable Z_0 as the coefficient of Z_O from the polynomial

Tip: $-(AB - B^2 - B^3)$

Solving for the roots Recall that you will need to use the `CubicEquationSolver` module to solve for the roots. The function is `solve(a, b, c, d)` where it takes the coefficients of the polynomial. Furthermore, the return value is an array of the roots. As there are three roots, you will need to store each root in a variable. The variables should be called:

Z_v: The largest root

Z_L: The smallest root

Z_none: The middle root

2.2.4 Task 6: Peng-Robinson Equation of State Plot - Roots

Now let's take a look at the plot of CO2 at temperature 284.0K and pressure at 4.6 MPa. On the plot we will also plot the saturation pressure and the saturation temperature, acquired from literature.

Task 6.1: Calculating the Volumes at the roots Calculate the volumes at the roots of the Peng-Robinson Equation of State for the gas, CO2, at the temperature 284.0K and pressure at 4.6 MPa.

The equation for the volume is:

$$V = \frac{RTZ}{P}$$

where:

- R is the ideal gas constant
- T is the temperature
- P is the pressure
- Z is the root of the Peng-Robinson Equation of State

The volume variable names should be

V_v: The volume at the largest root

V_L: The volume at the smallest root

V_none: The volume at the middle root

> ***Tip***: Use the variables from the previous task

Task 6.2: Plot the volumes at the roots on the Peng-Robinson Equation of State Plot of CO2 at 284.0K and 4.6 MPa Plot the volumes at the roots on the Peng-Robinson Equation of State Plot of CO2 at 284.0K and 4.6 MPa.

You will need to use the `plt.scatter()` function to plot the points. The function takes the following arguments: `plt.scatter(x, y, label)`. The x and y arguments are the x and y coordinates of the point and the label is what will be shown in the legend.

Further to this, you will need to annotate two of the points with either `Dew Point` or `Bubble Point`.You will need to use the `plt.annotate()` function to annotate the points. The function takes the following arguments: `plt.annotate(text, xy=(x, y), rotation=45)`. The text argument is the text to be displayed. The xy argument is the x and y coordinates of the point. The rotation argument is the rotation of the text. In this instance, it will be set to 45 degrees.

After you have plotted the points, you will need to use the the function `plot_example_ideal_gas_law_and_peng_robinson_eos()`, as can be seen in the example above of the plot.

2.2.5 Task 7: Peng-Robinson Equation of State - Mixing Rules & Fugacity Coefficients

In the previous tasks we have only been looking at the Peng-Robinson Equation of State for a single component. However, in reality, we are often dealing with mixtures of components. In this task, we will look at how to use the Peng-Robinson Equation of State to predict the fugacity coefficients of a mixture of components.

The fugacity coefficient is defined as the ratio of fugacity to pressure [1]. For gases at low pressures (where the ideal gas law is a good approximation), fugacity is roughly equal to pressure. Thus, for an ideal gas, the ratio between fugacity and pressure (the fugacity coefficient) is equal to 1 [1]. This ratio can be thought of as 'how closely' a real gas behaves like an ideal gas [1].

Task 7.1: Peng-Robinson Equation of State - Mixing Rules The next cell provides you with the variables required and the chemical data for components in a DataFrame. The DataFrame is called `chemical_data_df`

The chemical data contains the following columns:

- `Element`: The element name
- `Tc(K)`: The critical temperature of the element
- `Pc(MPa)`: The critical pressure of the element
- `Vc(cm3/mol)`: The critical volume of the element
- `w`: The acentric factor of the element

Below is the element names will be printed, as well as the number of rows and names of the columns.

[2]:
```python
# Shift + Enter to run the code
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

R = 8.314 # J/(mol*K)
T = 233.2 # K
P = 0.1 # MPa

chemical_data_df = pd.read_csv("src/Critical_Constants_and_Acentric_Factors.
 ↪csv")

print(chemical_data_df.columns)
print('-------------------')
print('Number of rows: ', chemical_data_df.shape[0])
print('Number of columns: ', chemical_data_df.shape[1])
print('-------------------')
print('List of elements: ', chemical_data_df['Element'].unique())
```

```
Index(['Element', 'Tc_K', 'Pc_MPa', 'Vc_cm3_mol', 'w'], dtype='object')
-------------------
Number of rows:  76
Number of columns:  5
-------------------
List of elements:  ['Argon' 'Bromine' 'Chlorine' 'Fluorine' 'Helium-4'
'Hydrogen' 'Iodine'
 'Krypton' 'Neon' 'Nitrogen' 'Oxygen' 'Xenon' 'Acetylene' 'Benzene'
 'n-Butane' '1-Butene' 'Cyclobutane' 'Cyclohexane' 'Cyclopropane' 'Ethane'
 'Ethylene' 'n-Heptane' 'n-Hexane' 'Isobutane' 'Isobutylene' 'Isopentane'
 'Methane' 'Naphthalene' 'n-Octane' 'n-Pentane' 'Propadiene' 'Propane'
 'Propylene' 'Toluene' 'm-Xylene' 'o-Xylene' 'p-Xylene' 'Ammonia'
 'Carbon dioxide' 'Carbon disulfide' 'Carbon monoxide'
'Carbon tetrachloride' 'Carbon tetrafluoride' 'Chloroform' 'Hydrazine'
'Hydrogen chloride' 'Hydrogen fluoride' 'Hydrogen sulfide' 'Nitric oxide'
'Nitrous oxide' 'Sulfur dioxide' 'Sulfur trioxide' 'Water' 'Acetaldehyde'
'Acetic acid' 'Acetone' 'Acetonitrile' 'Aniline' 'n-Butanol'
'Chlorobenzene' 'Dichlorodifluoromethane (Freon 12)' 'Diethyl ether'
'Dimethyl ether' 'Ethanol' 'Ethylene oxide' 'Isobutanol'
'Isopropyl alcohol' 'Methanol' 'Methyl chloride' 'Methyl ethyl ketone'
'Phenol' '1-Propanol' 'Pyridine' 'Trichlorotrifluoroethane (Freon 113)'
'Trichlorofluoromethane (Freon 11)' 'Trimethylamine']
```

Task 7.1.1: Peng-Robinson Equation of State - Mixing Rules - Binary Mixture You have been provided with the chemical data for two components, Methane and Propane. The chemical data is stored in the DataFrame `chemical_data_df`.

The mol fractions of the components are:

- $x_{CH_4} = 0.4$
- $x_{C_3H_8} = 0.6$

You will need to do the following:

1. Create a table of the chemical data for the two components, Methane2 and Propane2. The table should be called `chemical_data_df_binary_mixture`. It should have the following columns:

   - `Methane`
   - `Propane`

   It then should have the following index:

   - `mol_frac`
   - `Tc_K`
   - `Pc_MPa`
   - `w`
   - `ai`
   - `bi`
   - `ki`

2. Fill in the table with the correct values. The values for the `mol_frac` column should be the mol fractions of the components. The values for the `Tc_K` column should be the critical temperature of the components. The values for the `Pc_MPa` column should be the critical pressure of the components. The values for the `w` column should be the acentric factor of the components. The values for the `ai` column should be the attraction parameter of the component. The values for the `bi` column should be the covolume for the component. The values for the `ki` column should be the binary interaction of component.

   Tip: Creating an empty table (or rather, DataFrame) can be done using the `pd.DataFrame()` function. The function takes the following arguments: `pd.DataFrame(columns, index)`. The index argument is the index of the DataFrame and the columns argument is the columns of the DataFrame. These both accept only lists - `[]`.

Tip: You will need to use the `chemical_data_df` DataFrame to get the values for the `Tc_K`, `Pc_MPa` and `w` columns. You will need to use the `ai()` function to get the values for the `ai` column. You will need to use the `bi()` function to get the values for the `bi` column. You will need to use the `ki()` function to get the values for the `ki` column.

Tip: Access rows and setting them will require the use of the `.loc[]` function. The function takes the following arguments: `.loc[column]`. The column argument is the column to be accessed. The function will return the row of the column. To set the row of the column, you will need to use the following syntax: `.loc[column] = row`. The column argument is the column to be accessed. The row argument is the row to be set. Ensure the column is a string and the row is a list.

[4]:
```
from src.thermo import get_chemical_values


# Get the values for Tc_K - You will need to run the get_chemical_values␣
↪function. Remember to use ?get_chemical_values to get help

chemical_data_df_binary_mixture.loc['Tc_K'] = None # < - Enter your code here,␣
↪remove None

# Get the values for Pc_MPa - You will need to run the get_chemical_values␣
↪function. Remember to use ?get_chemical_values to get help
chemical_data_df_binary_mixture.loc['Pc_MPa'] = None # < - Enter your code␣
↪here, remove None

# Get the values for w - You will need to run the get_chemical_values function.␣
↪Remember to use ?get_chemical_values to get help
chemical_data_df_binary_mixture.loc['w'] = None # < - Enter your code here,␣
↪remove None

print(chemical_data_df_binary_mixture)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 6
      1 from src.thermo import get_chemical_values
      4 # Get the values for Tc_K - You will need to run the get_chemical_value␣
  ↪function. Remember to use ?get_chemical_values to get help
----> 6 chemical_data_df_binary_mixture.loc['Tc_K'] = None # < - Enter your cod␣
  ↪here, remove None
      8 # Get the values for Pc_MPa - You will need to run the␣
  ↪get_chemical_values function. Remember to use ?get_chemical_values to get hel
      9 chemical_data_df_binary_mixture.loc['Pc_MPa'] = None # < - Enter your␣
  ↪code here, remove None

NameError: name 'chemical_data_df_binary_mixture' is not defined
```