gasthermo

Release 0.0.6

Robert F. De Jaco

May 22, 2020

CONTENTS:

1 Definitions **1**
- 1.1 Nomenclature . . . 1

2 Examples **3**
- 2.1 Heat Capacity . . . 3
- 2.2 Equations of State . . . 4
  - 2.2.1 Cubic . . . 4
  - 2.2.2 Virial . . . 5
- 2.3 Mixtures . . . 5
  - 2.3.1 Residual Properties . . . 5
- 2.4 Other Utilities . . . 6
- 2.5 Gotchas . . . 7

3 Heat Capacity **9**

4 Critical Properties **13**

5 Virial Equation of State **15**
- 5.1 Theory . . . 15
- 5.2 Fugacity Coefficients . . . 16
- 5.3 Residual Partial Molar Properties . . . 16
- 5.4 Mixtures . . . 20

6 Cubic Equation of State **25**
- 6.1 Theory . . . 25
- 6.2 Residual Molar Properties . . . 26
- 6.3 Fugacity Coefficients . . . 26
- 6.4 Mixtures . . . 26

7 Vapor Thermal Conductivity **33**

8 Vapor Viscosity **35**

9 References **37**

10 Indices and tables **39**

Bibliography **41**

Python Module Index **43**

Index **45**

CHAPTER

ONE

DEFINITIONS

Residual properties for a given thermodynamic property M are defined as

$$M = M^{\mathrm{IG}} + M^{\mathrm{R}}$$

where M^{IG} is the value of the property in the ideal gas state and M^{R} is the residual value of the property.

More information on residual properties can be found in standard texts [SVanNessA05]

We **define** partial molar property $\bar{M}_i$ of species *i* in a mixture as

$$\bar{M}_i = \left(\frac{\partial (nM)}{\partial n_i} \right)_{P,T,n_j} \tag{1.1}$$

The mixture property is related to the partial molar property as

$$nM = \sum_i n_i \bar{M}_i$$

or, in terms of gas-phase mole fractions y_i,

$$M = \sum_i y_i \bar{M}_i$$

The following relationships also hold

$$\bar{M}_i = \bar{M}^{\mathrm{IG}} + \bar{M}^{\mathrm{R}} \tag{1.2}$$

$$M^{\mathrm{R}} = \sum_i y_i \bar{M}^{\mathrm{R}} \tag{1.3}$$

1.1 Nomenclature

Code	Symbol	Description
P	P	Pressure in Pa
V	V	Molar Volume in $\mathrm{m}^3/\mathrm{mol}$
R	R	gas constant SI units ($\mathrm{m}^3 \times \mathrm{Pa}/\mathrm{mol}/\mathrm{K}$)
T	T	temperature in K
T_c	T_c	critical temperature in K
P_c	P_c	critical pressure in Pa
T_r	T_r	reduced temperature (dimensionless)
V_c	V_c	Critical volume $\mathrm{m}^3/\mathrm{mol}$
w	ω	Accentric factor
y_i	y_i	mole fraction of component *i*
–	n_i	number of moles of component *i*

CHAPTER

TWO

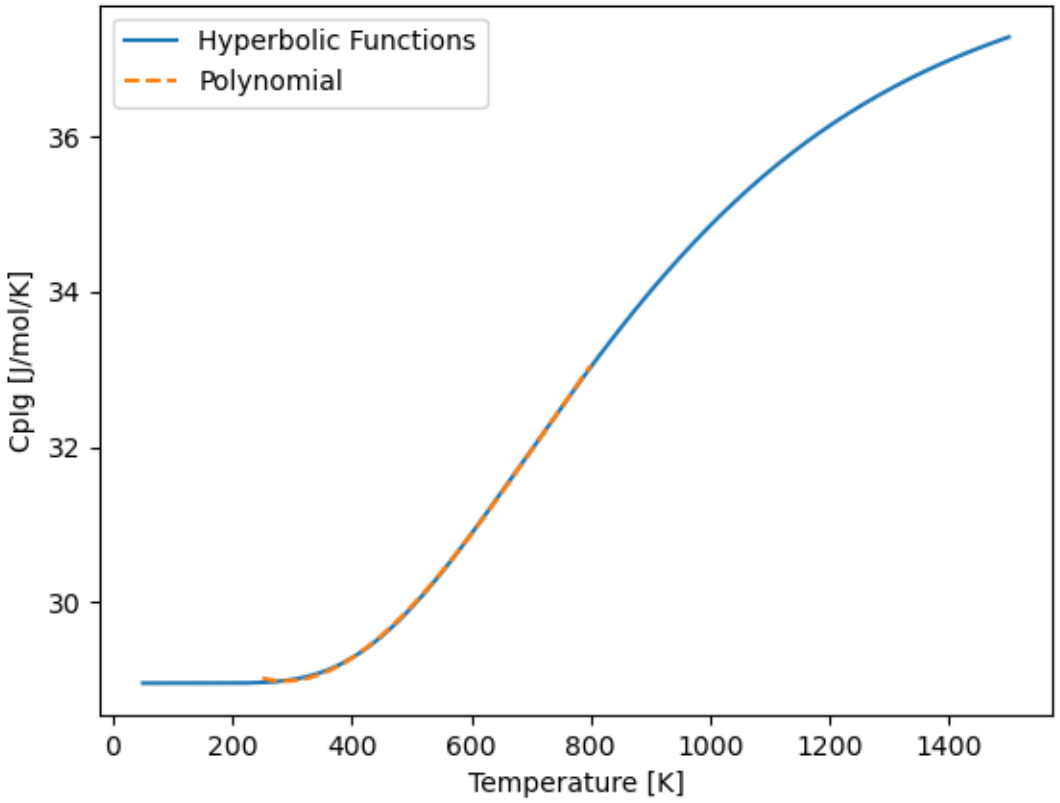
EXAMPLES

2.1 Heat Capacity

Determine the temperature dependence of the ideal gas heat capacity, C_p^{IG} for water

```
>>> import matplotlib.pyplot as plt
>>> from gasthermo.cp import CpIdealGas
>>> I = CpIdealGas(compound_name='Air', T_min_fit=250., T_max_fit=800., poly_order=3)
>>> I.eval(300.), I.Cp_units
(29.00369515161452, 'J/mol/K')
>>> I.eval(300.)/I.MW
1.0015088104839267
>>> # we can then plot and visualize the resutls
>>> fig, ax = I.plot()
>>> fig.savefig('docs/source/air.png')
>>> del I
```

And we will get something that looks like the following



and we notice that the polynomial (orange dashed lines) fits the hyperbolic function well.

2.2 Equations of State

2.2.1 Cubic

```
>>> from gasthermo.eos.cubic import PengRobinson, RedlichKwong, SoaveRedlichKwong
>>> P = 8e5  # Pa
>>> T = 300. # K
>>> PengRobinson(compound_name='Propane').iterate_to_solve_Z(P=P, T=T)
0.8568255826283575
>>> RedlichKwong(compound_name='Propane').iterate_to_solve_Z(P=P, T=T)
0.8712488647564147
>>> cls_srk = SoaveRedlichKwong(compound_name='Propane')
>>> Z = cls_srk.iterate_to_solve_Z(P=P, T=T)
>>> Z
0.8652883337846884
>>> # calculate residual properties
>>> from chem_util.chem_constants import gas_constant as R
>>> V = Z*R*T/P
>>> cls_srk.S_R_R_expr(P, V, T)
-0.3002887932902908
>>> cls_srk.H_R_RT_expr(P, V, T)
```

(continues on next page)

(continued from previous page)

```
-0.4271408507179967
>>> cls_srk.G_R_RT_expr(P, V, T) - cls_srk.H_R_RT_expr(P, V, T) + cls_srk.S_R_R_
↪expr(P, V, T)
0.0
```

2.2.2 Virial

```
>>> from gasthermo.eos.virial import SecondVirial
>>> Iv2 = SecondVirial(compound_name='Propane')
>>> Iv2.calc_Z_from_units(P=8e5, T=300.)
0.8726051032825523
```

2.3 Mixtures

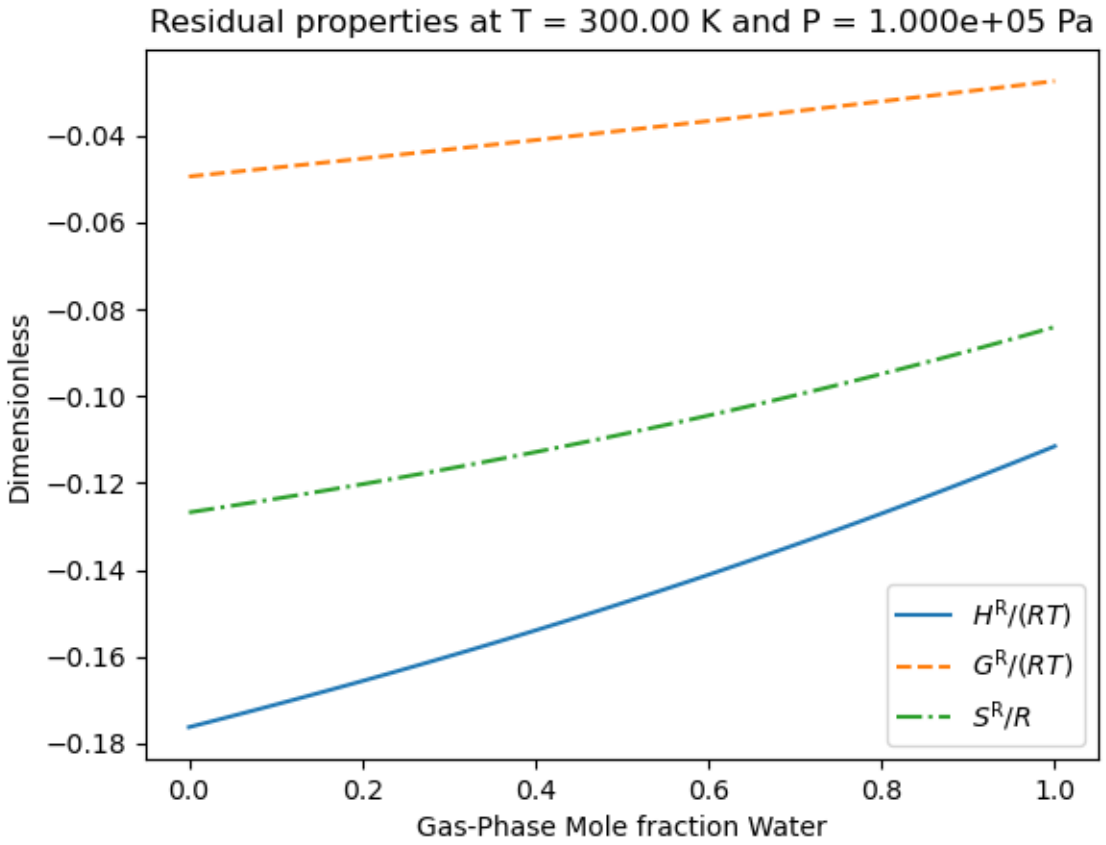
Note: currently only implemented for virial equation of state

2.3.1 Residual Properties

Below, an example is shown for calculating residual properties of THF/Water mixtures

```
>>> from gasthermo.eos.virial import SecondVirialMixture
>>> P, T = 1e5, 300.
>>> mixture = SecondVirialMixture(compound_names=['Water', 'Tetrahydrofuran'], k_ij=0.
↪)
>>> import matplotlib.pyplot as plt
>>> fig, ax = mixture.plot_residual_HSG(P, T)
>>> fig.savefig('docs/source/THF-WATER.png')
```

So that the results look like the following



We note that the residual properties will not always vanish in the limit of pure components like excess properties since the pure-components may not be perfect gases.

2.4 Other Utilities

Determine whether a single real root of the cubic equation of state can be used for simple computational implementation. In some regimes, the cubic equation of state only has 1 real root–in this case, the compressibility factor can be obtained easily.

```
>>> from gasthermo.eos.cubic import PengRobinson
>>> pr = PengRobinson(compound_name='Propane')
>>> pr.num_roots(300., 5e5)
3
>>> pr.num_roots(100., 5e5)
1
```

2.5 Gotchas

- All units are SI units

CHAPTER

THREE

HEAT CAPACITY

class gasthermo.cp.**CpIdealGas**(*dippr_no: str = None*, *compound_name: str = None*, *cas_number: str = None*, *T_min_fit: float = None*, *T_max_fit: float = None*, *n_points_fit: int = 1000*, *poly_order: int = 2*, *T_units='K'*, *Cp_units='J/mol/K'*)

Heat Capacity $C_{\mathrm{p}}^{\mathrm{IG}}$ [J/mol/K] at Constant Pressure of Inorganic and Organic Compounds in the Ideal Gas State Fit to Hyperbolic Functions [RWO+07]

$$C_{\mathrm{p}}^{\mathrm{IG}} = C_1 + C_2 \left[\frac{C_3/T}{\sinh\left(C_3/T\right)}\right] + C_4 \left[\frac{C_5/T}{\cosh\left(C_5/T\right)}\right]^2 \tag{3.1}$$

where $C_{\mathrm{p}}^{\mathrm{IG}}$ is in J/mol/K and T is in K.

Computing integrals of Equation (3.1) is challenging. Instead, the function is fit to a polynomial within a range of interest so that it can be integrated by using an antiderivative that is a polynomial.

Parameters

- **dippr_no** (*str, optional*) – dippr_no of compound by DIPPR table, defaults to None
- **compound_name** (*str, optional*) – name of chemical compound, defaults to None
- **cas_number** (*str, optional*) – CAS registry number for chemical compound, defaults to None
- **MW** (*float, derived from input*) – molecular weight in g/mol
- **T_min** (*float, derived from input*) – minimum temperature of validity for relationship [K]
- **T_max** (*float, derived from input*) – maximum temperature of validity [K]
- **T_min_fit** – minimum temperature for fitting, defaults to Tmin
- **T_max_fit** – maximum temperature for fitting, defaults to Tmax
- **C1** (*float, derived from input*) – parameter in Equation (3.1)
- **C2** (*float, derived from input*) – parameter in Equation (3.1)
- **C3** (*float, derived from input*) – parameter in Equation (3.1)
- **C4** (*float, derived from input*) – parameter in Equation (3.1)
- **C5** (*float, derived from input*) – parameter in Equation (3.1)
- **Cp_units** (*str, optional*) – units for $C_{\mathrm{p}}^{\mathrm{IG}}$, defaults to J/mol/K
- **T_units** (*str, optional*) – units for T, defaults to K

- **n_points_fit** (*int, optional*) – number of points for fitting polynomial and plotting, defaults to 1000
- **poly_order** (*int, optional*) – order of polynomial for fitting, defaults to 2

cp_integral(*T_a*, *T_b*)
Evaluate integral

$$\int_{T_a}^{T_b} C_{\mathrm{p}}^{\mathrm{IG}}(T')\mathrm{d}T' \tag{3.2}$$

Parameters

- **T_a** – start temperature in K
- **T_b** – finish temperature in K

Returns integral

eval(*T*, *f_sinh=<ufunc 'sinh'>*, *f_cosh=<ufunc 'cosh'>*)
Evaluate heat capacity

Parameters

- **T** – temperature in K
- **f_sinh** (*callable*) – function for hyperbolic sine, defaults to `np.sinh`
- **f_cosh** (*callable*) – function for hyperbolic cosine, defaults to `np.cosh`

Returns $C_{\mathrm{p}}^{\mathrm{IG}}$ J/mol/K (see equation (3.1))

get_numerical_percent_difference()
Calculate the percent difference with numerical integration

numerical_integration(*T_a*, *T_b*) → tuple
Numerical integration using scipy

class gasthermo.cp.**CpStar**(*T_ref: float*, ***kwargs*)
Dimensionless Heat Capacity at Constant Pressure of Inorganic and Organic Compounds in the Ideal Gas State Fit to Hyperbolic Functions [RWO+07]

The dimensionless form is obtained by introducing the following variables

$$C_{\mathrm{p}}^{\star} = \frac{C_{\mathrm{p}}^{\mathrm{IG}}}{\mathrm{R}} \tag{3.3}$$

$$T^{\star} = \frac{T}{T_{\mathrm{ref}}} \tag{3.4}$$

where R is the gas constant in units of J/mol/K, and T_{ref} is a reference temperature [K] input by the user (see `T_ref`)

The heat capacity in dimensionless form becomes

$$C_{\mathrm{p}}^{\star} = C_1^{\star} + C_2^{\star}\left[\frac{C_3^{\star}/T^{\star}}{\sinh\left(C_3^{\star}/T^{\star}\right)}\right] + C_4^{\star}\left[\frac{C_5^{\star}/T^{\star}}{\cosh\left(C_5^{\star}/T^{\star}\right)}\right]^2 \tag{3.5}$$

where

$$C_1^\star = \frac{C_1}{\mathrm{R}} \tag{3.6}$$
$$C_2^\star = (3.7)$$
$$C_3^\star = (3.8)$$
$$C_4^\star = (3.9)$$
$$C_5^\star = (3.10)$$

Parameters **T_ref** (*float*) – reference temperature [K] for dimensionless computations

eval(*T*, *f_sinh=<ufunc 'sinh'>*, *f_cosh=<ufunc 'cosh'>*)

Parameters

- **T** – temperature in K
- **f_sinh** (*callable*) – function for hyperbolic sine, defaults to `np.sinh`
- **f_cosh** (*callable*) – function for hyperbolic cosine, defaults to `np.cosh`

Returns $C_\mathrm{p}^\star$ [dimensionless] (see equation (3.5))

class gasthermo.cp.**CpRawData**(*T_raw: list*, *Cp_raw: list*, *T_min_fit: float = None*, *T_max_fit: float = None*, *poly_order: int = 2*, *T_units='K'*, *Cp_units='J/mol/K'*)

Obtain heat capacity relationships from raw data

Using input raw data # fit to polynomial of temperature # fit polynomial to antiderivative

Parameters

- **T_min_fit** (*float, optional*) – minimum temperature for fitting function [K]
- **T_max_fit** (*float, optional*) – maximum temperature for fitting function [K]
- **poly_order** (*int, optional*) – order of polynomial for fitting, defaults to 2
- **T_raw** (*list*) – raw temperatures in K
- **Cp_raw** (*list*) – raw heat capacities in J/K/mol
- **Cp_units** (*str, optional*) – units for C_p, defaults to J/mol/K
- **T_units** (*str, optional*) – units for T, defaults to K

get_max_percent_difference()

Get largest percent difference

CHAPTER

FOUR

CRITICAL PROPERTIES

class gasthermo.critical_constants.**CriticalConstants**(*dippr_no: str = None*, *compound_name: str = None*, *cas_number: str = None*, *MW: float = None*, *P_c: float = None*, *V_c: float = None*, *Z_c: float = None*, *T_c: float = None*, *w: float = None*)

Get critical constants of a compound

If critical constants are not passed in, reads from DIPPR table

Parameters

- **dippr_no** (*str, optional*) – dippr_no of compound by DIPPR table, defaults to None
- **compound_name** (*str, optional*) – name of chemical compound, defaults to None
- **cas_number** (*str, optional*) – CAS registry number for chemical compound, defaults to None
- **MW** – molecular weight in g/mol
- **T_c** – critical temperature [K]
- **P_c** – critical pressure [Pa]
- **V_c** – critical molar volume [m^3/mol]
- **Z_c** – critical compressibility factor [dimensionless]
- **w** – accentric factor [dimensionless]
- **tol** (*float, hard-coded*) – tolerance for percent difference in Zc calulcated and tabulated, set to 0.5

Z_c_percent_difference()

calculate percent difference between Z_c calculated and tabulated

calc_Z_c()

Calculate critical compressibility, for comparison to tabulated value

CHAPTER

FIVE

VIRIAL EQUATION OF STATE

5.1 Theory

The second order virial equation of state is [GP07]

$$Z = 1 + B\rho = 1 + \frac{BP}{RT} \tag{5.1}$$

Where the composition dependency of B is given by the *exact* mixing rule

$$B = \sum_i \sum_j y_i y_j B_{ij} \tag{5.2}$$

where $B_{ij} = B_{ji}$, and B_{ii} and B_{jj} are virial coefficients for the pure species

In this package, the useful correlation

$$\frac{BP_\mathrm{c}}{RT_\mathrm{c}} = B^0 + \omega B^1$$

or

$$B = \frac{RT_\mathrm{c}}{P_\mathrm{c}}\left(B^0 + \omega B^1\right) \tag{5.3}$$

So that, combining Equations (5.1) and (5.3), the compressibility can be calculated from dimensionless quantities as

$$Z = 1 + \left(B^0 + \omega B^1\right)\frac{P_\mathrm{r}}{T_\mathrm{r}} \tag{5.4}$$

where [SVanNessA05]

$$B^0 = 0.083 - \frac{0.422}{T_\mathrm{r}^{1.6}} \tag{5.5}$$

$$B^1 = 0.139 - \frac{0.172}{T_\mathrm{r}^{4.2}} \tag{5.6}$$

so that

so that the following derivatives can be computed as

$$\frac{dB^0}{dT_\mathrm{r}} = \frac{0.675}{T_\mathrm{r}^{2.6}} \tag{5.7}$$

$$\frac{dB^1}{dT_\mathrm{r}} = \frac{0.722}{T_\mathrm{r}^{5.2}} \tag{5.8}$$

Which allow the H^{R}, S^{R}, and G^{R} to be readily computed as follows [GP07]

$$\frac{G^{\mathrm{R}}}{RT} = \left(B^0 + \omega B^1\right) \frac{P_{\mathrm{r}}}{T_{\mathrm{r}}} \tag{5.9}$$

$$\frac{H^{\mathrm{R}}}{RT} = P_{\mathrm{r}} \left[\frac{B^0}{T_{\mathrm{r}}} - \frac{\mathrm{d}B^0}{\mathrm{d}T_{\mathrm{r}}} + \omega \left(\frac{B^1}{T_{\mathrm{r}}} - \frac{\mathrm{d}B^1}{\mathrm{d}T_{\mathrm{r}}} \right) \right] \tag{5.10}$$

$$\frac{S^{\mathrm{R}}}{R} = -P_{\mathrm{r}} \left(\frac{dB^0}{dT_{\mathrm{r}}} - \omega \frac{dB^1}{dT_{\mathrm{r}}} \right) \tag{5.11}$$

The cross coefficients are calculated as

$$B_{ij} = \frac{RT_{\mathrm{c}ij}}{P_{\mathrm{c}ij}} \left(B^0 + \omega_{ij} B^1\right) \tag{5.12}$$

so that the cross derivatives can be computed as

$$\frac{dB_{ij}}{dT} = \frac{RT_{\mathrm{c}ij}}{P_{\mathrm{c}ij}} \left(\frac{dB^0}{dT} + \omega_{ij} \frac{dB^1}{dT} \right)$$

$$\frac{dB_{ij}}{dT} = \frac{R}{P_{\mathrm{c}ij}} \left(\frac{dB^0}{dT_{\mathrm{r}ij}} + \omega_{ij} \frac{dB^1}{dT_{\mathrm{r}ij}} \right)$$

or, equivalently,

$$\frac{dB_{ij}}{dT_{\mathrm{r}ij}} = \frac{RT_{\mathrm{c}ij}}{P_{\mathrm{c}ij}} \left(\frac{dB^0}{dT_{\mathrm{r}ij}} + \omega_{ij} \frac{dB^1}{dT_{\mathrm{r}ij}} \right) \tag{5.13}$$

5.2 Fugacity Coefficients

The Fugacity coefficients are calculated as

$$\ln \hat{\phi}_i = \left(\frac{\partial (nG^{\mathrm{R}}/R/T)}{\partial n_i} \right)_{P,T,n_j}$$

For the virial equation of state, this becomes [VanNessA82]

$$\ln \hat{\phi}_k = \frac{P}{RT} \left[B_{kk} + \frac{1}{2} \sum_i \sum_j y_i y_j \left(2\delta_{ik} - \delta_{ij}\right) \right] \tag{5.14}$$

where *both* i and j indices run over all species

$$\delta_{ik} = 2B_{ik} - B_{ii} - B_{kk} = \delta_{ki} \tag{5.15}$$

and

$$\delta_{ii} = 0$$

5.3 Residual Partial Molar Properties

The partial molar residual free energy of component *k* is

$$\frac{\bar{G}_k^{\mathrm{R}}}{RT} = \ln \hat{\phi}_k = \frac{P}{RT} \left[B_{kk} + \frac{1}{2} \sum_i \sum_j y_i y_j \left(2\delta_{ik} - \delta_{ij}\right) \right] \tag{5.16}$$

The partial molar residual enthalpy of component *k* is

$$\frac{\bar{H}_k^{\mathrm{R}}}{RT} = -T\left(\frac{\partial \ln \hat{\phi}_k}{\partial T}\right)_{P,y} \tag{5.17}$$

$$= -\frac{T}{T_c}\left(\frac{\partial \ln \hat{\phi}_k}{\partial T_r}\right)_{P,y} \tag{5.18}$$

$$= -\frac{T}{T_c}\left\{\frac{P}{RT}\left[\frac{\partial B_{kk}}{\partial T_r} + \frac{1}{2}\sum_i\sum_j y_i y_j\left(2\frac{\partial \delta_{ik}}{\partial T_r} - \frac{\partial \delta_{ij}}{\partial T_r}\right)\right] - \frac{T_c \ln \hat{\phi}_i}{T}\right\} \tag{5.19}$$

where $\frac{\partial \delta_{ij}}{\partial T_r}$ is given by (5.33) so that we obtain

$$\frac{\bar{H}_k^{\mathrm{R}}}{RT} = -\frac{P}{RT_{\mathrm{c}}}\left[\frac{\partial B_{kk}}{\partial T_r} + \frac{1}{2}\sum_i\sum_j y_i y_j\left(2\frac{\partial \delta_{ik}}{\partial T_r} - \frac{\partial \delta_{ij}}{\partial T_r}\right)\right] + \ln \hat{\phi}_i \tag{5.20}$$

Since

$$G^{\mathrm{R}} = H^{\mathrm{R}} - TS^{\mathrm{R}}$$

In terms of partial molar properties, then

$$\bar{S}_i^{\mathrm{R}} = \frac{\bar{H}_i^{\mathrm{R}} - \bar{G}_i^{\mathrm{R}}}{T} \tag{5.21}$$

$$\frac{\bar{S}_i^{\mathrm{R}}}{R} = \frac{\bar{H}_i^{\mathrm{R}}}{RT} - \frac{\bar{G}_i^{\mathrm{R}}}{RT} \tag{5.22}$$

(5.23)

By comparing Equation (5.16) and (5.20) it is observed that

$$\frac{\bar{S}_k^{\mathrm{R}}}{R} = -\frac{P}{RT_{\mathrm{c}}}\left[\frac{\partial B_{kk}}{\partial T_r} + \frac{1}{2}\sum_i\sum_j y_i y_j\left(2\frac{\partial \delta_{ik}}{\partial T_r} - \frac{\partial \delta_{ij}}{\partial T_r}\right)\right] \tag{5.24}$$

where $\frac{\partial \delta_{ij}}{\partial T_r}$ is given by (5.33)

The partial molar residual volume of component *i* is calculated as

$$\frac{\bar{V}_k^{\mathrm{R}}}{RT} = \left(\frac{\partial \ln \hat{\phi}_i}{\partial P}\right)_{T,y} \tag{5.25}$$

$$= \frac{\partial}{\partial P}\left\{\frac{P}{RT}\left[B_{kk} + \frac{1}{2}\sum_i\sum_j y_i y_j\left(2\delta_{ik} - \delta_{ij}\right)\right]\right\} \tag{5.26}$$

which simplifies to

$$\frac{\bar{V}_k^{\mathrm{R}}}{RT} = \frac{1}{RT}\left[B_{kk} + \frac{1}{2}\sum_i\sum_j y_i y_j\left(2\delta_{ik} - \delta_{ij}\right)\right] \tag{5.27}$$

from which we obtain the intuitive result of

$$\bar{V}_k^{\mathrm{R}} = B_{kk} + \frac{1}{2}\sum_i\sum_j y_i y_j\left(2\delta_{ik} - \delta_{ij}\right)$$

class gasthermo.eos.virial.**Virial**(*pow: callable = <ufunc 'power'>*, *exp: callable = <ufunc 'exp'>*)

Parameters

- **pow** (*callable, optional*) – function for computing power, defaults to numpy.power
- **exp** (*callable, optional*) – function for computing logarithm, defaults to numpy.exp

B0_expr(*T_r*)

Parameters **T_r** – Reduced temperature

Returns Equation (5.5)

B1_expr(*T_r*)

Parameters **T_r** – reduced temperature

Returns Equation eq:*B1_expr*

B_expr(*T_r*, *w*, *T_c*, *P_c*)

Parameters

- **T_r** – reduced temperature
- **w** – accentric factor
- **T_c** – critical temperautre [K]
- **P_c** – critical pressure [Pa]

Returns Equation (5.3)

d_B0_d_Tr_expr(*T_r*)

Parameters **T_r** – reduced temperature

Returns Equation (5.7)

d_B1_d_Tr_expr(*T_r*)

Parameters **T_r** – reduced temperature

Returns Equation (5.8)

hat_phi_i_expr(**args*)

expression for fugacity coefficient :returns: $\exp\left(\ln \hat{\phi}_i\right)$

class gasthermo.eos.virial.**SecondVirial**(*dippr_no: str = None*, *compound_name: str = None*, *cas_number: str = None*, *pow: callable = <ufunc 'power'>*, ***kwargs*)

Virial equation of state for one component. See [GP07][SVanNessA05]

G_R_RT_expr(*P*, *T*)

Parameters

- **P** – pressure in Pa
- **T** – Temperautre in K

Returns Equation (5.9)

H_R_RT_expr(*P*, *T*)

Parameters

- **P** – pressure in Pa
- **T** – temperature in K

Returns Equation (5.10)

S_R_R_expr(*P*, *T*)

Parameters

- **P** – pressure in Pa
- **T** – temperature in K

Returns Equation (5.11)

calc_Z_from_dimensionless(*P_r*, *T_r*)

Parameters

- **P_r** – reduced pressure, dimensionless
- **T_r** – reduced temperature, dimensionless

Returns Equation (5.4)

calc_Z_from_units(*P*, *T*)

Parameters

- **P** – pressure in Pa
- **T** – temperature in K

Returns Equation (5.1)

ln_hat_phi_k_expr(*P*, *T*)
logarithm of fugacity coefficient

Note: single-component version

In this case, Equation (5.14) simplifies to

$$\ln \hat{\phi}_i = \frac{PB}{RT}$$

Parameters

- **P** (*float*) – pressure in Pa
- **T** (*float*) – temperature in K

plot_Z_vs_P(*T*, *P_min*, *P_max*, *symbol='o'*, *ax=None*, ***kwargs*)
Plot compressibility as a function of pressure

Parameters

- **T** (*float*) – temperature [K]
- **P_min** (*float*) – minimum pressure for plotting [Pa]
- **P_max** (*float*) – maximum pressure for plotting [Pa]
- **phase** (*str*) – phase type (liquid or vapor), defaults to vapor
- **symbol** (*str*) – marker symbol, defaults to 'o'
- **ax** (*plt.axis*) – matplotlib axes for plotting, defaults to None

- **kwargs** – keyword arguments for plotting

5.4 Mixtures

class gasthermo.eos.virial.**MixingRule**(*pow: callable = <ufunc 'power'>, exp: callable = <ufunc 'exp'>*)

Van der Walls mixing rule

combining rules from [PLdeAzevedo86]

$$\omega_{ij} = \frac{\omega_i + \omega_j}{2} \tag{5.28}$$

$$T_{cij} = \sqrt{T_{ci} T_{cj}}(1 - k_{ij}) \tag{5.29}$$

$$P_{cij} = \frac{Z_{cij} R T_{cij}}{V_{cij}} \tag{5.30}$$

$$Z_{cij} = \frac{Z_{ci} + Z_{cj}}{2} \tag{5.31}$$

$$V_{cij} = \left(\frac{V_{ci}^{1/3} + V_{cj}^{1/3}}{2} \right)^3 \tag{5.32}$$

P_cij_rule(*Z_ci*, *V_ci*, *T_ci*, *Z_cj*, *V_cj*, *T_cj*, *k_ij*)

Returns Equation (5.30)

T_cij_rule(*T_ci*, *T_cj*, *k_ij*)

Parameters

- **T_ci** – critical temperature of component i [K]
- **T_cj** – ** j [K]
- **k_ij** – k_ij parameter

Returns Equation (5.29)

V_cij_rule(*V_ci*, *V_cj*)

Parameters

- **V_ci** – critical molar volume of component *i* [m**3/mol]
- **V_cj** – critical molar volume of component *j* [m**3/mol]

Returns Equation eq:*Vc_combine*

Z_cij_rule(*Z_ci*, *Z_cj*)

Parameters

- **Z_ci** – critical compressibility factor of component *i*
- **Z_cj** – critical compressibility factor of component *j*

Returns Equation (5.31)

w_ij_rule(*w_i*, *w_j*)

Parameters

- **w_i** – accentric factor of component *i*
- **w_j** – accentric factor of component *j*

Returns Equation (5.28)

class gasthermo.eos.virial.**SecondVirialMixture**(*num_components=0*, *dippr_nos: List[str] = None*, *compound_names: List[str] = None*, *cas_numbers: List[str] = None*, *MWs: List[float] = None*, *P_cs: List[float] = None*, *V_cs: List[float] = None*, *Z_cs: List[float] = None*, *T_cs: List[float] = None*, *ws: List[float] = None*, *k_ij: Union[float, List[float]] = None*, *pow: callable = <ufunc 'power'>*, *exp: callable = <ufunc 'exp'>*)

Second virial with mixing rule from *MixingRule*

Note: can only input both custom critical properties or both from DIPPR–cant have mixed at the moment

Parameters

- **num_components** (*int*) – number of components
- **dippr_nos** (*typing.List[str]*) – dippr numbers of components
- **compound_names** (*typing.List[str]*) – names of components
- **cas_numbers** (*typing.List[str]*) – cas registry numbers
- **MWs** (*typing.List[float]*) – molecular weights of component sin g/mol
- **P_cs** (*typing.List[float]*) – critical pressures of componets in Pa
- **T_cs** (*typing.List[float]*) – critical temperatures of pure components in K
- **V_cs** (*typing.List[float]*) – critical molar volumes of pure components in m**3/mol
- **ws** (*typing.List[float]*) – accentric factors of components
- **k_ij** (*typing.List[typing.List[float]]*) – equation of state mixing rule in calculation of critical temperautre, see Equation (5.29). When $i = j$ and for chemical similar species, $k_{ij} = 0$. Otherwise, it is a small (usually) positive number evaluated from minimal PVT data or, in the absence of data, set equal to zero.

B_ij_expr(*i: int*, *j: int*, *T*)

Parameters

- **i** – index of first component
- **j** – index of second component
- **T** – temperature [K]

Returns Equation (5.12)

B_mix_expr(*y_k: List[Union[float, Any]]*, *T*)

Parameters

- **y_k** – mole fractions of each component k

- **T** – temperature in K

Returns Equation (5.2)

G_R_RT(**args*)
Residual free energy of mixture $G^{\mathrm{R}}/R/T$

H_R_RT(**args*)
Residual enthalpy of mixture $H^{\mathrm{R}}/R/T$

M_R_dimensionless(*method: callable*, *ys: List[Union[float, Any]]*, *P: float*, *T: float*)
Residual property of X for mixture.

Similar to Equation (1.3) but in dimensionless form

Parameters **method** (*callable*) – function to compute partial molar property of compound

S_R_R(**args*)
Residual entropy of mixture S^{R}/R

bar_GiR_RT(*cas_k: str*, *ys: List[Union[float, Any]]*, *P*, *T*)
Dimensionless residual partial molar free energy of component i

Parameters

- **cas_k** – cas number of component k
- **ys** – mole fractions
- **P** – pressure in Pa
- **T** – temperature in K

Returns Equation (5.16)

bar_HiR_RT(*cas_k: str*, *ys: List[Union[float, Any]]*, *P*, *T*)
Dimensionless residual partial molar enthalpy of component i

Parameters

- **cas_k** – cas number for component of interest
- **ys** – mole fractions
- **P** – pressure in Pa
- **T** – temperature in K

bar_SiR_R(*cas_k: str*, *ys: List[Union[float, Any]]*, *P*, *T*)
Dimensionless residual partial molar entropy of component i

Parameters

- **cas_k** – cas number for component of interest
- **ys** – mole fractions
- **P** – pressure in Pa
- **T** – temperature in K

Returns Equation (5.24)

bar_ViR_RT(*cas_k: str*, *ys: List[Union[float, Any]]*, *P*, *T*)
residual Partial molar volume for component *i*

Note: This expression does not depend on P

Parameters

- **cas_k** – cas number for component of interest
- **ys** – mole fractions
- **P** – pressure in Pa
- **T** – temperature in K

Returns Equation (5.27)

calc_Z_from_units(*y_k: List*, *P*, *T*)

Parameters

- **y_k** – mole fractions of each component k
- **P** – pressure in Pa
- **T** – temperature in K

Returns Equation (5.1)

d_Bij_d_Trij(*i: int*, *j: int*, *T*)

Parameters

- **i** – index for component *i*
- **j** – index for componen t *j*
- **T** – temperature in K

Returns Equation (5.13)

d_dij_d_Tr(*i*, *j*, *T*)

$$\frac{\partial \delta_{ij}}{\partial T_{\mathrm{r}ij}} = 2\frac{\partial B_{ij}}{\partial T_{\mathrm{r}ij}} - \frac{\partial B_{ii}}{\partial T_{\mathrm{r}ij}} - \frac{\partial B_{jj}}{\partial T_{\mathrm{r}ij}} \tag{5.33}$$

Todo: test this with symbolic differentiation of d_ik expression

Parameters

- **i** – index for component *i*
- **j** – index for componen t *j*
- **T** – temperature [K]

d_ik_expr(*i: int*, *k: int*, *T*)

Parameters

- **i** – index of component *i*
- **k** – index of component *k*
- **T** – temperature [K]

Returns Equation (5.15)

fugacity_i_expr(*cas_i: str*, *ys: List[Union[float, Any]]*, *P*, *T*)
Fugacity of component i in mixture $f_i = \hat{\phi}_i y_i P$

Parameters

- **cas_i** – cas number for component of interest
- **ys** – mole fractions
- **P** – pressure in Pa
- **T** – temperature in K

get_w_Tc_Pc(*i: int*, *j=None*)
Returns critical constants for calculation based off of whetner i = j or not

Returns (w, T_c, P_c)

Return type tuple

ln_hat_phi_k_expr(*k: int*, *ys: List[Union[float, Any]]*, *P*, *T*)
logarithm of fugacity coefficient

Parameters

- **k** – index of component *k*
- **ys** – mole fractions
- **P** – pressure in Pa
- **T** – temperature in K

Returns Equation (5.14)

plot_residual_HSG(*P*, *T*, *ax=None*, *fig=None*) → Tuple[matplotlib.pyplot.figure, matplotlib.pyplot.subplot]
Plot dimensionless residual properties as a function of mole fraction

Parameters

- **P** – pressure in Pa
- **T** – Temperature in K
- **ax** – matplotlib ax, defaults to None
- **fig** – matplotlib figure, defautls to None

CHAPTER

SIX

CUBIC EQUATION OF STATE

6.1 Theory

The generic cubic equation of state is [GP07]

$$P = \frac{RT}{V-b} - \frac{a(T)}{(V+\epsilon b)(V+\sigma b)}$$

where ϵ and σ are pure numbers (the same for all substances), and $a(T)$ and b are given by the following equations

$$a(T) = \Psi \frac{\alpha(T_r) R^2 T_\mathrm{c}^2}{P_\mathrm{c}} \tag{6.1}$$

$$b = \Omega \frac{RT_\mathrm{c}}{P_\mathrm{c}}$$

The compressibility factor can be calculated by solving the following equation

$$Z - \left(1 + \beta - \frac{q\beta(Z-\beta)}{(Z+\epsilon\beta)(Z+\sigma\beta)}\right) = 0 \tag{6.2}$$

where

$$\beta = \Omega \frac{P_\mathrm{r}}{T_\mathrm{r}} \tag{6.3}$$

and

$$q = \frac{\Psi \alpha(T_\mathrm{r})}{\Omega T_\mathrm{r}} \tag{6.4}$$

An iterative routine to calculate Z using Equation (6.2) is implemented, following [GP07], where

$$Z^{\mathrm{new}} = 1 + \beta - \frac{q\beta(Z^{\mathrm{old}}-\beta)}{(Z^{\mathrm{old}}+\epsilon\beta)(Z^{\mathrm{old}}+\sigma\beta)} \tag{6.5}$$

that is continued until the following is true

$$\left\| \frac{Z^{\mathrm{new}} - Z^{\mathrm{old}}}{Z^{\mathrm{new}} + Z^{\mathrm{old}}} \times 200 \right\| < \mathrm{tol} \tag{6.6}$$

6.2 Residual Molar Properties

$$\frac{H^{\mathrm{R}}}{RT} = Z - 1 + \left[\frac{\mathrm{d}\ln\alpha(T_{\mathrm{r}})}{\mathrm{d}\ln T_{\mathrm{r}}} - 1\right] qI \tag{6.7}$$

$$\frac{G^{\mathrm{R}}}{RT} = Z - 1 + \ln(Z-\beta) - qI \tag{6.8}$$

$$\frac{S^{\mathrm{R}}}{R} = \ln(Z-\beta) + \frac{\mathrm{d}\ln\alpha(T_{\mathrm{r}})}{\mathrm{d}\ln T_{\mathrm{r}}} qI \tag{6.9}$$

where the following functions have been defined

$$I = \frac{1}{\sigma-\epsilon}\ln\left(\frac{Z+\sigma\beta}{Z+\epsilon\beta}\right) \tag{6.10}$$

$$1 + \beta - \frac{q\beta(Z-\beta)}{(Z+\epsilon\beta)(Z+\sigma\beta)} \tag{6.11}$$

6.3 Fugacity Coefficients

The pure-component fugacity coefficient is *defined* as

$$\frac{G_i^{\mathrm{R}}}{RT} = \ln\hat{\phi}_i$$

So that, from Equation (6.8),

$$\ln\hat{\phi}_i = Z_i - 1 + \ln(Z_i - \beta_i) - q_i I_i \tag{6.12}$$

6.4 Mixtures

Warning: Not implemented yet!

Todo: Implement mixtures with cubic equations of state

class gasthermo.eos.cubic.**Cubic**(*sigma: float*, *epsilon: float*, *Omega: float*, *Psi: float*, *dippr_no: str = None*, *compound_name: str = None*, *cas_number: str = None*, *log: callable = <ufunc 'log'>*, *exp: callable = <ufunc 'exp'>*, *name: str = 'cubic'*, ***kwargs*)

Generic Cubic Equation of State

Parameters

- **sigma** – Parameter defined by specific equation of state, σ
- **epsilon** – Parameter defined by specific equation of state, ϵ
- **Omega** – Parameter defined by specific equation of state, Ω
- **Psi** – Parameter defined by specific equation of state, Ψ
- **tol** – tolerance for iteration (see Equation (6.6)), set to 0.01

- **log** – function for computing natural log, defaults to `numpy.log`
- **exp** – function for computing exponential, defaults to `numpy.exp`

G_R_RT_expr(*P*, *V*, *T*)
Dimensionless residual gibbs

Parameters

- **P** – pressure in Pa
- **V** – molar volume in m**3/mol
- **T** – temperature in K

Returns $\frac{G^{\mathrm{R}}}{RT}$, see Equation (6.8)

H_R_RT_expr(*P*, *V*, *T*)
Dimensionless residual enthalpy

Parameters

- **P** – pressure in Pa
- **V** – molar volume in m**3/mol
- **T** – temperature in K

Returns $\frac{H^{\mathrm{R}}}{RT}$, see Equation (6.7)

I_expr(*P*, *V*, *T*)

Parameters

- **P** – pressure in Pa
- **V** – molar volume in m**3/mol
- **T** – temperature in K

Returns I (see Equation (6.10))

S_R_R_expr(*P*, *V*, *T*)
Dimensionless residual entropy

Parameters

- **P** – pressure in Pa
- **V** – molar volume in m**3/mol
- **T** – temperature in K

Returns $\frac{S^{\mathrm{R}}}{R}$, see Equation (6.9)

Z_right_hand_side(*Z*, *beta*, *q*)
Estimate of compressibility of vapor [GP07], used for iterative methods

Returns RHS of residual, see Equation (6.11)

a_expr(*T*)

Parameters **T** – temperautre in K

Returns $a(T)$ (see Equation (6.1))

alpha_expr(*T_r*)
An empirical expression, specific to a particular form of the equation of state

Parameters **T_r** – reduced temperature (T/Tc), dimensionless

Returns $\alpha(T_r)$ see Table ??

beta_expr(*T*, *P*)

Parameters

- **T** – temperature in K
- **P** – pressure in Pa

Returns β (see Equation (6.3))

cardano_constants(*T*, *P*)

Parameters

- **T** – temperature [T]
- **P** – pressure [Pa]

Returns cardano constants p, q

Return type tuple

coefficients(*T*, *P*)

Polynomial oefficients for cubic equation of state

$$Z^3 c_0 + Z^2 c_1 + Z * c_2 + c_3 = 0$$

Returns (c_0, c_1, c_2, c_3)

d_ln_alpha_d_ln_Tr(*T_r*)

Parameters **T_r** – reduced temperature [dimensionless]

Returns Expression for $\frac{\mathrm{d} \ln \alpha(T_\mathrm{r})}{\mathrm{d} \ln T_\mathrm{r}}$

hat_phi_i_expr(**args*)

expression for fugacity coefficient :returns: $\exp\left(\ln \hat{\phi}_i\right)$

iterate_to_solve_Z(*T*, *P*) $\rightarrow$ float

Iterate to compute Z using Equation (6.5) util termination condition (Equation (6.6) is met)

Parameters

- **T** – temperature in K
- **P** – pressure in Pa

Returns compressibility factor

ln_hat_phi_k_expr(*P*, *V*, *T*)

Parameters

- **P** – pressure in Pa
- **T** – temperature in K
- **V** – molar volume in m**3/mol

Returns $\ln \hat{\phi}_k$, see Equation $ln_hat_phi_i$

num_roots(*T*, *P*)

Find number of roots

See [ML12][Dei02]

Parameters

- **T** – temperature in K
- **P** – pressure in Pa

Returns number of roots

plot_Z_vs_P(*T: float*, *P_min: float*, *P_max: float*, *symbol='o'*, *ax: matplotlib.pyplot.axis = None*, *fig: matplotlib.pyplot.figure = None*, ***kwargs*)
Plot compressibility as a function of pressure

Parameters

- **T** – temperature [K]
- **P_min** – minimum pressure for plotting [Pa]
- **P_max** – maximum pressure for plotting [Pa]
- **symbol** – marker symbol, defaults to 'o'
- **ax** – matplotlib axes for plotting, defaults to None
- **kwargs** – keyword arguments for plotting

print_roots(*T*, *P*)
Check to see if all conditions have one root

q_expr(*T*)

Parameters **T** – temperautre in K

Returns q (see Equation (6.4))

residual(*P*, *V*, *T*)

Parameters

- **P** – pressure in Pa
- **V** – volume in [mol/m**3]
- **T** – temperature in K

Returns residual for cubic equation of state (Equation (6.2))

class gasthermo.eos.cubic.**RedlichKwong**(***kwargs*)
Redlich-Kwong Equation of State [RK49]

This Equation of state has the following parameters [SVanNessA05]

Symbol	Value
$\alpha(T_\mathrm{r})$	$1/\sqrt{T_\mathrm{r}}$
σ	1
ϵ	0
Ω	0.08664
Ψ	0.42748

```
>>> from gasthermo.eos.cubic import RedlichKwong
>>> model = RedlichKwong(compound_name='Propane')
>>> model.sigma
1
>>> model.epsilon
0
>>> model.Omega
```

(continues on next page)

(continued from previous page)

```
0.08664
>>> model.Psi
0.42748
```

alpha_expr(*T_r*)

An empirical expression, specific to a particular form of the equation of state

Parameters **T_r** – reduced temperature (T/Tc), dimensionless

Returns $\alpha(T_r)$ see Table ??

d_ln_alpha_d_ln_Tr(*T_r*)

Parameters **T_r** – reduced temperature [dimensionless]

Returns Expression for $\frac{\mathrm{d}\ln\alpha(T_\mathrm{r})}{\mathrm{d}\ln T_\mathrm{r}}$

class gasthermo.eos.cubic.**SoaveRedlichKwong**(***kwargs*)

Soave Redlich-Kwong Equation of State [Soa72]

This equation of state has the following parameters

Symbol	Value
$\alpha(T_\mathrm{r})$	$\left[1+f_w(1-\sqrt{T_\mathrm{r}})\right]^2$
σ	1
ϵ	0
Ω	0.08664
Ψ	0.42748

where

$$f_w = 0.480 + 1.574\omega - 0.176\omega^2 \tag{6.13}$$

```
>>> from gasthermo.eos.cubic import SoaveRedlichKwong
>>> model = SoaveRedlichKwong(compound_name='Water')
>>> model.Omega
0.08664
>>> model.sigma
1
>>> model.epsilon
0
>>> model.Psi
0.42748
>>> model.f_w_rule(0.)
0.48
>>> model.f_w_rule(1.)
1.878
```

Parameters **f_w** (*float, derived*) – empirical expression used in *alpha* [dimensionless], see Equation (6.13)

alpha_expr(*T_r*)

An empirical expression, specific to a particular form of the equation of state

Parameters **T_r** – reduced temperature (T/Tc), dimensionless

Returns $\alpha(T_r)$ see Table ??

d_ln_alpha_d_ln_Tr(*T_r*)

Parameters **T_r** – reduced temperature [dimensionless]

Returns Expression for $\frac{\mathrm{d}\ln\alpha(T_\mathrm{r})}{\mathrm{d}\ln T_\mathrm{r}}$

f_w_rule(*w*)

Parameters **w** – accentric factor

Returns f_w, see Equation (6.13)

class gasthermo.eos.cubic.**PengRobinson**(***kwargs*)

Peng-Robinson Equation of State [PR76]

This equation of state has the following parameters

Symbol	Value
$\alpha(T_\mathrm{r})$	$\left[1+f_w(1-\sqrt{T_\mathrm{r}})\right]^2$
σ	$1+\sqrt{2}$
ϵ	$1-\sqrt{2}$
Ω	0.07780
Ψ	0.45724

where

$$f_w = 0.480 + 1.574\omega - 0.176\omega^2 \tag{6.14}$$

```
>>> from gasthermo.eos.cubic import PengRobinson
>>> model = PengRobinson(compound_name='Water')
>>> model.Omega
0.0778
>>> model.sigma
2.4142135
>>> model.epsilon
-0.414213
>>> model.Psi
0.45724
>>> model.f_w_rule(0.)
0.37464
>>> model.f_w_rule(1.)
1.64698
```

Parameters **f_w** (*float, derived*) – empirical expression used in *alpha* [dimensionless?]

f_w_rule(*w*)

Parameters **w** – accentric factor

Returns f_w, see Equation (6.14)

CHAPTER

SEVEN

VAPOR THERMAL CONDUCTIVITY

class gasthermo.thermal_conductivity.**ThermalConductivity**(*dippr_no: str = None*, *compound_name: str = None*, *cas_number: str = None*, *T_min_fit: float = None*, *T_max_fit: float = None*, *n_points_fit: int = 1000*, *poly_order: int = 2*)

Thermal Conductivity of Inorganic and Organic Substances [W/m/K] [RWO+07]

$$k = \frac{C_1 T^{C_2}}{1 + C_3/T + C_4/T^2} \tag{7.1}$$

where k is the thermal conductivity in W/m/K and T is in K. Thermal conductivites are either at 1 atm or the vapor pressure, whichever is lower.

Parameters

- **dippr_no** (*str, optional*) – dippr_no of compound by DIPPR table, defaults to None
- **compound_name** (*str, optional*) – name of chemical compound, defaults to None
- **cas_number** (*str, optional*) – CAS registry number for chemical compound, defaults to None
- **MW** (*float, derived from input*) – molecular weight in g/mol
- **T_min** (*float, derived from input*) – minimum temperature of validity for relationship [K]
- **T_max** (*float, derived from input*) – maximum temperature of validity [K]
- **C1** (*float, derived from input*) – parameter in Equation (7.1)
- **C2** (*float, derived from input*) – parameter in Equation (7.1)
- **C3** (*float, derived from input*) – parameter in Equation (7.1)
- **C4** (*float, derived from input*) – parameter in Equation (7.1)
- **units** (*str*) – units for k, set to W/m/K
- **T_min_fit** – minimum temperature for fit, defaults to T_min
- **T_max_fit** – maximum temperature for fit, defaults to T_max

eval(*T*)

Parameters **T** – temperature in K

Returns k W/m/K (see equation (7.1))

class gasthermo.thermal_conductivity.**ThermalConductivityMixture**(*name_to_cas: dict*, *mixing_rule='Simple'*)

Viscosity of vapor mixture using Wilke mixing rule

Parameters

- **name_to_cas** (dict[components, str]) – mapping of chemical name to cas registry number
- **mixing_rule** (*str, optional*) – mixing rule for calculation of viscosity, defaults to Simple
- **pure** (dict[components, Viscosity]) – pure component viscosity info, obtained rom gasthermo.vapor_viscosity.Viscosity

eval_HR(*y_i*, *T*)

Weights based off of sqrt of molecular weights

eval_simple(*y_i*, *T*)

Calculate thermal conductivity using simple relationship

Parameters **y_i** (dict[component, float]) – mole fraction of each component i

CHAPTER

EIGHT

VAPOR VISCOSITY

class gasthermo.viscosity.**Viscosity**(*dippr_no: str = None*, *compound_name: str = None*, *cas_number: str = None*, *T_min_fit: float = None*, *T_max_fit: float = None*, *n_points_fit: int = 1000*, *poly_order: int = 2*)

Vapor Viscosity of Inorganic and Organic Substances [W/m/K] [RWO+07]

$$\mu = \frac{C_1 T^{C_2}}{1 + C_3/T + C_4/T^2} \tag{8.1}$$

where μ is the thermal conductivity in W/m/K and T is in K. Viscosities are either at 1 atm or the vapor pressure, whichever is lower.

Parameters

- **dippr_no** (*str, optional*) – dippr_no of compound by DIPPR table, defaults to None
- **compound_name** (*str, optional*) – name of chemical compound, defaults to None
- **cas_number** (*str, optional*) – CAS registry number for chemical compound, defaults to None
- **MW** (*float, derived from input*) – molecular weight in g/mol
- **T_min** (*float, derived from input*) – minimum temperature of validity for relationship [K]
- **T_max** (*float, derived from input*) – maximum temperature of validity [K]
- **C1** (*float, derived from input*) – parameter in Equation (8.1)
- **C2** (*float, derived from input*) – parameter in Equation (8.1)
- **C3** (*float, derived from input*) – parameter in Equation (8.1)
- **C4** (*float, derived from input*) – parameter in Equation (8.1)
- **units** (*str*) – units for μ, set to Pa*s
- **T_min_fit** – minimum temperature for fit, defaults to T_min
- **T_max_fit** – maximum temperature for fit, defaults to T_max

eval(*T*)

Parameters **T** – temperature in K

Returns μ Pa*s (see equation (8.1))

class gasthermo.viscosity.**ViscosityMixture**(*name_to_cas: dict = None*, *mixing_rule='Herning Zipperer'*, ***kwargs*)

Viscosity of vapor mixture using Wilke or HR mixing rule

Parameters

- **name_to_cas** (dict[components, str]) – mapping of chemical name to cas registry number
- **mixing_rule** (*str, optional*) – mixing rule for calculation of viscosity, defaults to Herning Zipperer
- **pure** (dict[components, Viscosity]) – pure component viscosity info, obtained rom *Viscosity*

eval_Wilke(*y_i*, *T*)

Calculate mixture viscosity in Pa*s using Wilke mixing rule

Parameters **y_i** (dict[component, float]) – mole fraction of each component i

phi_ij(*i: str*, *j: str*, *T: float*)

Coefficient for each pair of components in a mixtures

Parameters

- **i** – name of component i
- **j** – name of component j

CHAPTER

NINE

REFERENCES

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Dei02] U K Deiters. Calculation of Densities from Cubic Equations of State. *AIChE J.*, 48:882–886, 2002.

[GP07] D W Green and R H Perry. *Perry's Chemical Engineers' Handbook*. McGraw-Hill Professional Publishing, 8th edition, 2007.

[ML12] R Monroy-Loperena. A Note on the Analytical Solution of Cubic Equations of State in Process Simulation. *Ind. Eng. Chem. Res.*, 51:6972–6976, 2012. doi:10.1021/ie2023004.

[PR76] D-Y Peng and D B Robinson. A New Two-Constant Equation of State. *Ind. Eng. Chem., Fundam.*, 15:59–64, 1976.

[PLdeAzevedo86] J Prausnitz, R N Lichtenthaler, and E G de Azevedo. *Molecular Thermodynamics of Fluid-Phase Equilibria*, pages 132,162. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 1986.

[RK49] O Redlich and J N S Kwong. On the Thermodynamics of Solutions. V: An Equation of State. Fugacities of Gaseous Solutions. *Chem. Rev.*, 44:233–244, 1949.

[RWO+07] R L Rowley, W V Wilding, J L Oscarson, Y Yang, N A Zundel, T E Daubert, and R P Danner. DIPPR$^\mathrm ®$ Data Compilation of Pure Chemical Properties, Design Institute for Physical Properties. In *Design Institute for Physical Properties of the American Institute of Chemical Engineers*. AIChE, New York, 2007.

[SVanNessA05] J M Smith, H C Van Ness, and M M Abbott. *Introduction to Chemical Engineering Thermodynamics*. McGraw-Hill, 7th edition, 2005.

[Soa72] G Soave. Equilibrium Constants from a Modified Redlich-Kwong Equation of State. *Chem. Eng. Sci.*, 27:1197–1203, 1972.

[VanNessA82] H C Van Ness and M M Abbott. *Classical Thermodynamics of Nonelectrolyte Solutions: With Applications to Phase Equilibria*. McGraw-Hill, New York, 1982.

PYTHON MODULE INDEX

g

gasthermo, 3
gasthermo.cp, 9
gasthermo.critical_constants, 13
gasthermo.eos.cubic, 25
gasthermo.eos.virial, 15
gasthermo.thermal_conductivity, 33
gasthermo.viscosity, 35

INDEX

A

a_expr() (*gasthermo.eos.cubic.Cubic method*), 27
alpha_expr() (*gasthermo.eos.cubic.Cubic method*), 27
alpha_expr() (*gasthermo.eos.cubic.RedlichKwong method*), 30
alpha_expr() (*gasthermo.eos.cubic.SoaveRedlichKwong method*), 30

B

B0_expr() (*gasthermo.eos.virial.Virial method*), 18
B1_expr() (*gasthermo.eos.virial.Virial method*), 18
B_expr() (*gasthermo.eos.virial.Virial method*), 18
B_ij_expr() (*gasthermo.eos.virial.SecondVirialMixture method*), 21
B_mix_expr() (*gasthermo.eos.virial.SecondVirialMixture method*), 21
bar_GiR_RT() (*gasthermo.eos.virial.SecondVirialMixture method*), 22
bar_HiR_RT() (*gasthermo.eos.virial.SecondVirialMixture method*), 22
bar_SiR_R() (*gasthermo.eos.virial.SecondVirialMixture method*), 22
bar_ViR_RT() (*gasthermo.eos.virial.SecondVirialMixture method*), 22
beta_expr() (*gasthermo.eos.cubic.Cubic method*), 28

C

calc_Z_c() (*gasthermo.critical_constants.CriticalConstants method*), 13
calc_Z_from_dimensionless() (*gasthermo.eos.virial.SecondVirial method*), 19
calc_Z_from_units() (*gasthermo.eos.virial.SecondVirial method*), 19
calc_Z_from_units() (*gasthermo.eos.virial.SecondVirialMixture method*), 23
cardano_constants() (*gasthermo.eos.cubic.Cubic method*), 28
coefficients() (*gasthermo.eos.cubic.Cubic method*), 28
cp_integral() (*gasthermo.cp.CpIdealGas method*), 10
CpIdealGas (*class in gasthermo.cp*), 9
CpRawData (*class in gasthermo.cp*), 11
CpStar (*class in gasthermo.cp*), 10
CriticalConstants (*class in gasthermo.critical_constants*), 13
Cubic (*class in gasthermo.eos.cubic*), 26

D

d_B0_d_Tr_expr() (*gasthermo.eos.virial.Virial method*), 18
d_B1_d_Tr_expr() (*gasthermo.eos.virial.Virial method*), 18
d_Bij_d_Trij() (*gasthermo.eos.virial.SecondVirialMixture method*), 23
d_dij_d_Tr() (*gasthermo.eos.virial.SecondVirialMixture method*), 23
d_ik_expr() (*gasthermo.eos.virial.SecondVirialMixture method*), 23
d_ln_alpha_d_ln_Tr() (*gasthermo.eos.cubic.Cubic method*), 28
d_ln_alpha_d_ln_Tr() (*gasthermo.eos.cubic.RedlichKwong method*), 30
d_ln_alpha_d_ln_Tr() (*gasthermo.eos.cubic.SoaveRedlichKwong method*), 31

E

eval() (*gasthermo.cp.CpIdealGas method*), 10
eval() (*gasthermo.cp.CpStar method*), 11

eval() (*gasthermo.thermal_conductivity.ThermalConductivity method*), 33
eval() (*gasthermo.viscosity.Viscosity method*), 35
eval_HR() (*gasthermo.thermal_conductivity.ThermalConductivityMixture method*), 34
eval_simple() (*gasthermo.thermal_conductivity.ThermalConductivityMixture method*), 34
eval_Wilke() (*gasthermo.viscosity.ViscosityMixture method*), 36

F

f_w_rule() (*gasthermo.eos.cubic.PengRobinson method*), 31
f_w_rule() (*gasthermo.eos.cubic.SoaveRedlichKwong method*), 31
fugacity_i_expr() (*gasthermo.eos.virial.SecondVirialMixture method*), 24

G

G_R_RT() (*gasthermo.eos.virial.SecondVirialMixture method*), 22
G_R_RT_expr() (*gasthermo.eos.cubic.Cubic method*), 27
G_R_RT_expr() (*gasthermo.eos.virial.SecondVirial method*), 18
gasthermo
    module, 3
gasthermo.cp
    module, 9
gasthermo.critical_constants
    module, 13
gasthermo.eos.cubic
    module, 25
gasthermo.eos.virial
    module, 15
gasthermo.thermal_conductivity
    module, 33
gasthermo.viscosity
    module, 35
get_max_percent_difference() (*gasthermo.cp.CpRawData method*), 11
get_numerical_percent_difference() (*gasthermo.cp.CpIdealGas method*), 10
get_w_Tc_Pc() (*gasthermo.eos.virial.SecondVirialMixture method*), 24

H

H_R_RT() (*gasthermo.eos.virial.SecondVirialMixture method*), 22
H_R_RT_expr() (*gasthermo.eos.cubic.Cubic method*), 27
H_R_RT_expr() (*gasthermo.eos.virial.SecondVirial method*), 18
hat_phi_i_expr() (*gasthermo.eos.cubic.Cubic method*), 28
hat_phi_i_expr() (*gasthermo.eos.virial.Virial method*), 18

I

I_expr() (*gasthermo.eos.cubic.Cubic method*), 27
iterate_to_solve_Z() (*gasthermo.eos.cubic.Cubic method*), 28

L

ln_hat_phi_k_expr() (*gasthermo.eos.cubic.Cubic method*), 28
ln_hat_phi_k_expr() (*gasthermo.eos.virial.SecondVirial method*), 19
ln_hat_phi_k_expr() (*gasthermo.eos.virial.SecondVirialMixture method*), 24

M

M_R_dimensionless() (*gasthermo.eos.virial.SecondVirialMixture method*), 22
MixingRule (*class in gasthermo.eos.virial*), 20
module
    gasthermo, 3
    gasthermo.cp, 9
    gasthermo.critical_constants, 13
    gasthermo.eos.cubic, 25
    gasthermo.eos.virial, 15
    gasthermo.thermal_conductivity, 33
    gasthermo.viscosity, 35

N

num_roots() (*gasthermo.eos.cubic.Cubic method*), 28
numerical_integration() (*gasthermo.cp.CpIdealGas method*), 10

P

P_cij_rule() (*gasthermo.eos.virial.MixingRule method*), 20
PengRobinson (*class in gasthermo.eos.cubic*), 31
phi_ij() (*gasthermo.viscosity.ViscosityMixture method*), 36
plot_residual_HSG() (*gasthermo.eos.virial.SecondVirialMixture method*), 24
plot_Z_vs_P() (*gasthermo.eos.cubic.Cubic method*), 29
plot_Z_vs_P() (*gasthermo.eos.virial.SecondVirial method*), 19

print_roots() (*gasthermo.eos.cubic.Cubic method*), 29

Q

q_expr() (*gasthermo.eos.cubic.Cubic method*), 29

R

RedlichKwong (*class in gasthermo.eos.cubic*), 29
residual() (*gasthermo.eos.cubic.Cubic method*), 29

S

S_R_R() (*gasthermo.eos.virial.SecondVirialMixture method*), 22
S_R_R_expr() (*gasthermo.eos.cubic.Cubic method*), 27
S_R_R_expr() (*gasthermo.eos.virial.SecondVirial method*), 19
SecondVirial (*class in gasthermo.eos.virial*), 18
SecondVirialMixture (*class in gasthermo.eos.virial*), 21
SoaveRedlichKwong (*class in gasthermo.eos.cubic*), 30

T

T_cij_rule() (*gasthermo.eos.virial.MixingRule method*), 20
ThermalConductivity (*class in gasthermo.thermal_conductivity*), 33
ThermalConductivityMixture (*class in gasthermo.thermal_conductivity*), 34

V

V_cij_rule() (*gasthermo.eos.virial.MixingRule method*), 20
Virial (*class in gasthermo.eos.virial*), 17
Viscosity (*class in gasthermo.viscosity*), 35
ViscosityMixture (*class in gasthermo.viscosity*), 35

W

w_ij_rule() (*gasthermo.eos.virial.MixingRule method*), 20

Z

Z_c_percent_difference() (*gasthermo.critical_constants.CriticalConstants method*), 13
Z_cij_rule() (*gasthermo.eos.virial.MixingRule method*), 20
Z_right_hand_side() (*gasthermo.eos.cubic.Cubic method*), 27