SciThermo

Release v0.0.1

Robert F. DeJaco

May 23, 2020

CONTENTS:

CHAPTER

ONE

HEAT CAPACITY

class scithermo.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]

$$C_{\mathrm{p}}^{\mathrm{IG}} = C_1 + C_2 \left[\frac{C_3/T}{\sinh\left(C_3/T\right)}\right] + C_4 \left[\frac{C_5/T}{\cosh\left(C_5/T\right)}\right]^2 \tag{1.1}$$

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

Computing integrals of Equation (1.1) 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 (1.1)
- **C2** (*float, derived from input*) – parameter in Equation (1.1)
- **C3** (*float, derived from input*) – parameter in Equation (1.1)
- **C4** (*float, derived from input*) – parameter in Equation (1.1)
- **C5** (*float, derived from input*) – parameter in Equation (1.1)
- **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*)

$$\int_{T_a}^{T_b} C_{\mathrm{p}}^{\mathrm{IG}}(T')\mathrm{d}T' \tag{1.2}$$

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'>*)

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 (1.1))

get_numerical_percent_difference()
Calculate the percent difference with numerical integration obtained by scipy

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

class scithermo.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

$$C_{\mathrm{p}}^{\star} = \frac{C_{\mathrm{p}}^{\mathrm{IG}}}{\mathrm{R}} \tag{1.3}$$

$$T^{\star} = \frac{T}{T_{\mathrm{ref}}} \tag{1.4}$$

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

$$C_{\mathrm{p}}^{\star} = C_1^{\star} + C_2^{\star}\left[\frac{C_3^{\star}/T^{\star}}{\sinh\left(C_3^{\star}/T^{\star}\right)}\right] + C_4^{\star}\left[\frac{C_5^{\star}/T^{\star}}{\cosh\left(C_5^{\star}/T^{\star}\right)}\right]^2 \tag{1.5}$$

where

$$C_1^\star = \frac{C_1}{\mathrm{R}} \tag{1.6}$$
$$C_2^\star = \frac{C_2}{\mathrm{R}} \tag{1.7}$$
$$C_3^\star = \frac{C_3}{T_{\mathrm{ref}}} \tag{1.8}$$
$$C_4^\star = \frac{C_4}{\mathrm{R}} \tag{1.9}$$
$$C_5^\star = \frac{C_5}{T_{\mathrm{ref}}} \tag{1.10}$$

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 (1.5))

class scithermo.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'*)

From raw data for Cp(T) * 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

TWO

CRITICAL PROPERTIES

class scithermo.critical_constants.**CriticalConstants**(*dippr_no: str = None*, *compound_name: str = None*, *cas_number: str = None*, ***kwargs*)

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** (*float, derived from input*) – molecular weight in g/mol
- **T_c** (*float, derived from input*) – critical temperature [K]
- **P_c** (*float, derived from input*) – critical pressure [Pa]
- **V_c** (*float, derived from input*) – critical molar volume [m^3/mol]
- **Z_c** (*float, derived from input*) – critical compressibility factor [dimensionless]
- **w** (*float, derived from input*) – 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

class scithermo.critical_constants.**CriticalConstants**(*dippr_no: str = None*, *compound_name: str = None*, *cas_number: str = None*, ***kwargs*)

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** (*float, derived from input*) – molecular weight in g/mol
- **T_c** (*float, derived from input*) – critical temperature [K]
- **P_c** (*float, derived from input*) – critical pressure [Pa]
- **V_c** (*float, derived from input*) – critical molar volume [m^3/mol]
- **Z_c** (*float, derived from input*) – critical compressibility factor [dimensionless]
- **w** (*float, derived from input*) – 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

THREE

VIRIAL EQUATION OF STATE

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

Parameters

- **R** (*float, hard-coded*) – gas constant, set to SI units
- **pow** (*callable, optional*) – function for computing power, defaults to numpy.power
- **exp** (*callable, optional*) – function for computing logarithm, defaults to numpy.exp

d_B0_d_Tr_expr(*T_r*)

$$\frac{\mathrm{d}B^0}{\mathrm{d}T_\mathrm{r}}$$

d_B1_d_Tr_expr(*T_r*)

$$\frac{\mathrm{d}B^1}{\mathrm{d}T_\mathrm{r}}$$

hat_phi_i_expr(**args*)
expression for fugacity coefficient

class scithermo.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*)
Dimensionless residual gibbs

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

Returns Expression for residual gibbs free (divided by RT) – dimensionless

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

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

Returns Expression for residual enthalpy (divided by RT) – dimensionless

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

$$\frac{S^{\mathrm{R}}}{R} = -P_{\mathrm{r}}\left(\frac{\mathrm{d}B^0}{\mathrm{d}T_{\mathrm{r}}} + \omega\frac{\mathrm{d}B^1}{\mathrm{d}T_{\mathrm{r}}}\right)$$

Returns Expression for residual entropy (divided by R) – dimensionless

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

Note: single-component version

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

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

class scithermo.eos.virial.**BinarySecondVirial**(*i_kwargs=None*, *j_kwargs=None*, *k_ij=0.0*, *pow: callable = <ufunc 'power'>*, *exp: callable = <ufunc 'exp'>*)
Second virial with combining rules from [PLdeAzevedo86]

$$w_{ij} = \frac{w_i + w_j}{2} \tag{3.1}$$

$$T_{\mathrm{c},ij} = \sqrt{T_{\mathrm{c},i}T_{\mathrm{c},j}}(1 - k_{ij}) : label : eq:Tcij \tag{3.2}$$

$$P_{\mathrm{c},ij} = \frac{Z_{\mathrm{c},ij}RT_{\mathrm{c},ij}}{V_{\mathrm{c},ij}} \tag{3.3}$$

(3.4)

where

$$Z_{\mathrm{c},ij} = \frac{Z_{\mathrm{c},i} + Z_{\mathrm{c},j}}{2} \tag{3.5}$$

$$V_{\mathrm{c},ij} = \left(\frac{V_{\mathrm{c},i}^{1/3} + V_{\mathrm{c},j}^{1/3}}{2}\right)^3 \tag{3.6}$$

Parameters

- **k_ij** (*float*) – equation of state mixing rule in calculation of critical temperautre, see Equation eq_Tcij. 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.
- **cas_pairs** (*list(tuple(str))*) – pairs of cas registry numbers, derived from cas_numbers calculated
- **other_cas** (*dict*) – map from one cas number to other

B_ij_expr (*cas_1*, *cas_2*, *T*)

Returns B_{ij} considering that i=j or i!=j

If i=j, find the component k for which k=i=j (all cas no's equal). Then return B_{kk} where

$$B_{kk} = \frac{RT_{c,k}}{P_{c,k}}\left[B^0\left(\frac{T}{T_{c,k}}\right) + \omega_k\left(\frac{T}{T_{c,k}}\right)\right]$$

Otherwise, if i!=j, return B_{ij}, where

$$B_{ij} = \frac{RT_{c,ij}}{P_{c,ij}}\left[B^0\left(\frac{T}{T_{c,ij}}\right) + \omega_{ij}\left(\frac{T}{T_{c,ij}}\right)\right]$$

This is implemented in a simplified fashion usintg *BinarySecondVirial.get_w_Tc_Pc()* and then calling the generic expression for B

Parameters

- **cas_1** (*str*) – cas number of first component
- **cas_2** (*str*) – cas number of second component
- **T** – temperature [K]

B_mix_expr (*y_k*, *T*)

Parameters

- **y_k** (dict[cas_numbers, float]) – mole fractions of each component k
- **T** – temperature in K

Returns B [m^3/mol], where $Z = 1 + BP/R/T$

G_R_RT (**args*)

Residual free energy of mixture G^{R}

H_R_RT (**args*)

Residual enthalpy of mixture H^{R}

S_R_R (**args*)

Residual entropy of mixture S^{R}

Tstar_d_lnphi_dTstar (*cas_i*, *y_i*, *P*, *T*)

Returns

$$T^{\star}\frac{\partial \ln \hat{\phi}_i}{\partial T^{\star}} = \frac{TT_{\text{ref}}}{T_{\text{ref}}}\frac{\partial \ln \hat{\phi}_i}{\partial T} = \frac{T}{T_{\text{c}}}\frac{\partial \ln \hat{\phi}_i}{\partial T_{\text{r}}} \tag{3.7}$$

$$= \frac{T}{T_{\text{c}}}\left[\frac{P}{RT}\left(\frac{\partial B_{ii}}{\partial T_r} + (1-y_i)^2\frac{\partial \delta_{ij}}{\partial T_r}\right) - \frac{T_c \ln \hat{\phi}_i}{T}\right] \tag{3.8}$$

$$= \frac{P}{RT_{\text{c}}}\left(\frac{\partial B_{ii}}{\partial T_r} + (1-y_i)^2\frac{\partial \delta_{ij}}{\partial T_r}\right) - \ln \hat{\phi}_i \tag{3.9}$$

$$= \frac{\bar{H}_i^{\text{R}}}{RT} \tag{3.10}$$

where $\frac{\partial \delta_{ij}}{\partial T_r}$ is given by (3.21)

Parameters

- **cas_i** (*str*) – cas number for component of interest
- **y_i** (*float*) – mole fraction of component of interest
- **P** (*float*) – pressure in Pa
- **T** (*float*) – temperature in K

X_R_dimensionless (*method: callable*, *cas_i: str*, *y_i: float*, *P: float*, *T: float*)
Residual property of X for mixture.

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

bar_GiR_RT (**args*)
Dimensionless residual partial molar free energy of component i

$$\frac{\bar{G}_i^{\text{R}}}{RT} = \ln \hat{\phi}_i \tag{3.11}$$

bar_HiR_RT (*cas_i*, *y_i*, *P*, *T*)
Dimensionless residual partial molar enthalpy of component i

$$\frac{\bar{H}_i^{\text{R}}}{RT} = -T\left(\frac{\partial \ln \hat{\phi}_i}{\partial T}\right)_{P,y} \tag{3.12}$$

$$= -\frac{T}{T_c}\left(\frac{\partial \ln \hat{\phi}_i}{\partial T_r}\right)_{P,y} \tag{3.13}$$

$$= -\frac{T}{T_c}\left(\frac{P}{RT}\left[\frac{\partial B_{ii}}{\partial T_r} + (1-y_i)^2\frac{\partial \delta_{ij}}{\partial T_r}\right] - \frac{T_c \ln \hat{\phi}_i}{T}\right) \tag{3.14}$$

where $\frac{\partial \delta_{ij}}{\partial T_r}$ is given by (3.21) so that we obtain

$$\frac{\bar{H}_i^{\text{R}}}{RT} = -\frac{P}{RT_c}\left[\frac{\partial B_{ii}}{\partial T_r} + (1-y_i)^2\frac{\partial \delta_{ij}}{\partial T_r}\right] + \ln \hat{\phi}_i \tag{3.15}$$

Parameters

- **cas_i** (*str*) – cas number for component of interest
- **y_i** (*float*) – mole fraction of component of interest
- **P** (*float*) – pressure in Pa
- **T** (*float*) – temperature in K

bar_SiR_R(*cas_i*, *y_i*, *P*, *T*)

Dimensionless residual partial molar entropy of component i

Since

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

In terms of partial molar properties, then

$$\bar{S}_i^{\mathrm{R}} = \frac{\bar{H}_i^{\mathrm{R}} - \bar{G}_i^{\mathrm{R}}}{T} \tag{3.16}$$

$$\frac{\bar{S}_i^{\mathrm{R}}}{R} = \frac{\bar{H}_i^{\mathrm{R}}}{RT} - \frac{\bar{G}_i^{\mathrm{R}}}{RT} \tag{3.17}$$

(3.18)

By comparing Equation (3.11) and (3.15) it is observed that

$$\frac{\bar{S}_i^{\mathrm{R}}}{R} = -\frac{P}{RT_c}\left[\frac{\partial B_{ii}}{\partial T_r} + (1-y_i)^2\frac{\partial \delta_{ij}}{\partial T_r}\right]$$

where $\frac{\partial \delta_{ij}}{\partial T_r}$ is given by (3.21)

Parameters

- **cas_i** (*str*) – cas number for component of interest
- **y_i** (*float*) – mole fraction of component of interest
- **P** (*float*) – pressure in Pa
- **T** (*float*) – temperature in K

bar_ViR_RT(*cas_i*, *y_i*, *P*, *T*)

residual Partial molar volume for component *i*

$$\frac{\bar{V}_i^{\mathrm{R}}}{RT} = \left(\frac{\partial \ln \hat{\phi}_i}{\partial P}\right)_{T,y} \tag{3.19}$$

$$= \frac{B_{ii} + (1-y_i)^2\delta_{ij}}{RT} \tag{3.20}$$

Note: This expression does not depend on P

Parameters

- **cas_i** (*str*) – cas number for component of interest
- **y_i** (*float*) – mole fraction of component of interest
- **P** (*float*) – pressure in Pa
- **T** (*float*) – temperature in K

Returns $\bar{V}_i^{\mathrm{R}}/R/T$

calc_Z(*y_k*, *P*, *T*)

Parameters

- **y_k** (*dict*) – mole fractions of each component k

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

Returns Z [mol/m^3], where $Z = 1 + BP/R/T$

d_dij_d_Tr(T)

$$\frac{\partial \delta_{ij}}{\partial T_r} = 2\frac{\partial B_{ij}}{\partial T_r} - \frac{\partial B_{ii}}{\partial T_r} - \frac{\partial B_{jj}}{\partial T_r} \tag{3.21}$$

Parameters **T** – temperature [K]

d_ij_expr(T)

$$\delta_{ij} = 2B_{ij} - B_{ii} - B_{jj} \tag{3.22}$$

Parameters **T** – temperature [K]

Returns δ_{ij} [m**3/mol]

fugacity_i_expr(*cas_i*, *y_i*, *P*, *T*)
Fugacity of component i in mixture $f_i = \hat{\phi}_i y_i P$

Parameters

- **cas_i** (*str*) – cas number for component of interest
- **y_i** (*float*) – mole fraction of component of interest
- **P** (*float*) – pressure in Pa
- **T** (*float*) – temperature in K

get_w_Tc_Pc(*cas_i*, *cas_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_i_expr(*cas_i*, *y_i*, *P*, *T*)
logarithm of fugacity coefficient

$$\ln \hat{\phi}_i = \frac{P}{RT}\left[B_{ii} + (1-y_i)^2 \delta_{ij}\right]$$

Parameters

- **cas_i** (*str*) – cas number for component of interest
- **y_i** (*float*) – mole fraction of component of interest
- **P** (*float*) – pressure in Pa
- **T** (*float*) – temperature in K

where δ_{ij} is given by Equation (3.22)

CHAPTER

FOUR

CUBIC EQUATION OF STATE

class scithermo.eos.cubic.**Cubic**(*sigma: float*, *epsilon: float*, *Omega: float*, *Psi: float*, *dippr_no: str = None*, *compound_name: str = None*, *cas_number: str = None*, ***kwargs*)

Generic Cubic Equation of State Defined as in [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. $a(T)$ and b are substance-dependent.

Parameters

- **R** (*float, hard-coded*) – gas constant, set to SI units
- **sigma** (*float*) – Parameter defined by specific equation of state, σ
- **epsilon** (*float*) – Parameter defined by specific equation of state, ϵ
- **Omega** (*float*) – Parameter defined by specific equation of state, Ω
- **Psi** (*float*) – Parameter defined by specific equation of state, Ψ
- **tol** (*float*) – tolerance for percent difference between compressilibity factor calculated iteratively, set to 0.01

G_R_RT_expr(*P*, *V*, *T*, *log=None*)

Dimensionless residual gibbs

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

Returns Expression for residual gibbs free (divided by RT) – dimensionless

H_R_RT_expr(*P*, *V*, *T*, *log=None*)

Dimensionless residual enthalpy

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

Returns Expression for residual enthalpy (divided by RT) – dimensionless

H_expr(*P*, *V*, *T*, *T_ref*, *val_ref=0.0*, *log=None*)

Expression for fluid enthalpy

Parameters

- **P** – pressure in Pa
- **V** – molar volume [m**3/mol]

- **T** – temperautre in K
- **T_ref** – reference temperature in K
- **val_ref** – value at reference temperature [J/mol/K]
- **log** – function used for logarithm

Returns H [J/mol/K]

I_expr(*P*, *V*, *T*, *log=None*)

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

Parameters **log** (*callable, optional*) – function to be used for natural logarithm, defaults to np.log

S_R_R_expr(*P*, *V*, *T*, *log=None*)
Dimensionless residual entropy

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

Returns Expression for residual entropy (divided by R) – dimensionless

Z_liquid_RHS(*Z*, *beta*, *q*)
Compressibility of vapor [GP07]

$$\beta + (Z + \epsilon\beta)(Z + \sigma\beta)\left(\frac{1 + \beta - Z}{q\beta}\right) \tag{4.1}$$

Z_vapor_RHS(*Z*, *beta*, *q*)
Compressibility of vapor [GP07]

$$1 + \beta - \frac{q\beta(Z - \beta)}{(Z + \epsilon\beta)(Z + \sigma\beta)} \tag{4.2}$$

Returns

a_expr(*T*)

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

Parameters **T** – temperautre in K

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 $\alpha(T_r)$

beta_expr(*T*, *P*)

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

Parameters

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

Returns β

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

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

Parameters

- **T** – temperature in K
- **P** – pressure in Pa
- **phase** (*str*) – phase [vapor or liquid]

Returns compressibility factor

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*, *P_min*, *P_max*, *phase='vapor'*, *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

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

q_expr(*T*)

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

Parameters **T** – temperautre in K

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

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

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 $\alpha(T_r)$

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 scithermo.eos.cubic.**SoaveRedlichKwong**(***kwargs*)

Soave Redlich-Kwong Equation of State [Soa72]

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

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 $\alpha(T_r)$

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 scithermo.eos.cubic.**PengRobinson**(***kwargs*)
Peng-Robinson Equation of State [PR76]

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

CHAPTER

FIVE

VAPOR THERMAL CONDUCTIVITY

CHAPTER

SIX

VAPOR VISCOSITY

CHAPTER

SEVEN

REFERENCES

CHAPTER

EIGHT

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.

PYTHON MODULE INDEX

S

INDEX

H

I

L

M

N

P

Q

R

S

T

V

X

Z