Summer Fellowship Report

On

Development and Extension of Thermodynamic Models and Functions in OpenModelica

Submitted by

Ayushi Sinha

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

August 1, 2020

Acknowledgment

I would like to extend my deepest thanks to my Internship guide Prof. Kanan M Moudgalya, Department of Chemical Engineering, IIT Bombay, for his support and insight. I would also like to thank our mentors Priyam Nayak and Rahul Nagraj with my whole heart, without whom I would not have had the opportunity to work on and experience a fellowship of this caliber. Their continuous support and encouragement has made this learning experience unforgettable.

Contents

Introduction

OpenModelica" is a free and open-source modelling environment that uses “Modelica" modelling language. It follows equation oriented approach. OpenModelica can be used for modelling, simulation, optimization and analysis of complex steady state and dynamic systems. Modelica modelling language allows users to express a system in the form of equations. OpenModelica complies expressions, equations, functions and algorithms into C code. The generated C code is combined with a library of utility functions, a run-time library, and a numerical Differential-Algebric Equation (DAE) solver. OpenModelica Connection Editor, called as OMEdit is the integrated Graphical User Interface (GUI) in OpenModelica for graphical modelling and editing. OMEdit consists of several libraries for various domains like Electrical, Magnetic, Math, Thermal, etc. It provides various user friendly features like representation of a model in the form of block diagrams. OMEdit can be used for creating custom models and for editing or drawing connections between the model interfaces. It also allows users to plot graphs between parameters of the model simulated.

Chapter 1: Thermodynamic Functions

1. Surface Tension

Description: Surface tension of a liquid is the property which makes it behave like a stretched membrane. The cohesive forces in the liquid act on the molecules on the surface to create tension. If surface molecules could be displaced slightly outward from the surface, they would be attracted back by the nearby molecules. The energy responsible for the phenomenon of surface tension may be thought of as approximately equivalent to the work or energy required to remove the surface layer of molecules in a unit area. Surface tension may be expressed, therefore, in units of energy (joules) per unit area (square meters).

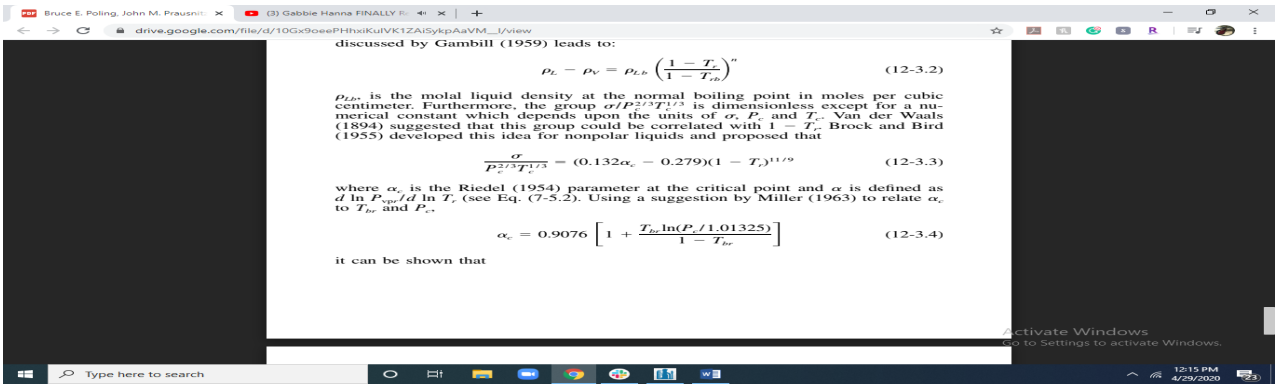
Equations:

The surface tension formula has 2 ways to go about it. In DWSIM, the Surface tension of a pure component is calculated using the formula, in a certain temperature range:

$$\mathrm{ST} = \mathrm{A} + e^{\left(\frac{B}{T} + C + DT + ET^2\right)}$$

Where A, B, C, D and E are coefficients provided in the ChemSep Database. The temperature range for each component is also provided in the database.

For components outside this temperature range, the Brock-Bird Equation is used as the empirical equation to calculate surface tension:



Here, P_c and T_c are critical pressure and temperature respectively. T_r is reduced temperature and T_{br} is reduced boiling point of the pure component.

Development:

The Brock Bird equation used for empirical calculation of Surface Tension was referred from the Solution Inspector in DWSIM. The VB code on GitHub had the following equation:

$\mathrm{ST} = \mathrm{A} + e^{\left(\frac{B}{T} + C + DT + ET^2\right)}$

This gives us identical values to DWSIM results in a middle range of temperatures. Whereas the Brock Bird equation is employed at extreme temperatures outside the specified range for each component.

Table:

Compound/s	**Temperature (K)**	**Composition**	**DWSIM value (N/m)**	**Experimental**
Water	298.15	Pure component	0.07	0.07
Ethanol	298.15	Pure component	0.221	0.221
Acetic Acid	298.15	Pure component	0.027	0.027
Acetone	298.15	Pure component	0.023	0.023

1.2 Joule Thompson Coefficient

Description: The Joule-Thompson effect refers to the temperature change which occurs when a Real Gas is expanded through a valve or porous plug in adiabatic conditions. The non-ideality of the gas along with the change in kinetic energy leads to this temperature change. The JT Coefficient is defined as the change in temperature per change in unit pressure at adiabatic conditions. At adiabatic conditions, the kinetic energy change has no effect and just the non-ideal nature contributes to the change in temperature. The JT Coefficient is a measure of the non-ideality of the gas.

Equations:

This is the derivation of the Joule coefficient, $\eta = (\partial\ T/\partial\ V)_U$.

Now entropy is a function of state – i.e. of the intensive state variables P, V and T. (V = molar volume.). Let us choose to express S as a function of V and T, so that

$$dS = \left(\frac{\partial S}{\partial V}\right)_T dV + \left(\frac{\partial S}{\partial T}\right)_V dT$$

In the Joule experiment, the internal energy of the gas is constant, so that

TdS− PdV=0

dS = $\frac{PdV}{T}$

For the first term on the right hand side of equation we make use of the Maxwell relation:

$$\left(\frac{\partial S}{\partial V}\right)_T = \left(\frac{\partial P}{\partial T}\right)_V$$

For the second term on the right hand side we obtain

$$\left(\frac{\partial S}{\partial T}\right)_V = \left(\frac{\partial S}{\partial U}\right)_V \left(\frac{\partial U}{\partial T}\right)_V = \frac{\left(\frac{\partial U}{\partial T}\right)_V}{\left(\frac{\partial U}{\partial S}\right)_V} = \frac{Cv}{T}$$

$$\frac{PdV}{T} = \left(\frac{\partial P}{\partial T}\right)_V dV + \frac{\text{Cv dT}}{T}$$

Multiply through by T, and divide by dV, taking the infinitesimal limit as $dV \rightarrow 0$ and given that internal energy is constant, and we get

$$\text{P=T } \left(\tfrac{\partial P}{\partial T}\right)_V \quad + \quad Cv \left(\tfrac{\partial T}{\partial V}\right)_U$$

from which we obtain

$$\left(\frac{\partial T}{\partial V}\right)_U = \frac{1}{Cv}\left[P - T\left(\frac{\partial P}{\partial T}\right)_V\right]$$

Let us now consider the *Joule-Thomson coefficient*

$$\mu = (\partial\ T/\partial\ P)_H$$

Differentiating S as a function of P and T, so that

$$dS = \left(\frac{\partial S}{\partial P}\right)_T dP + \left(\frac{\partial S}{\partial T}\right)_P dT$$

In the Joule-Thomson experiment, the enthalpy of the gas is constant, so that

$$\mathrm{TdS} + \mathrm{VdP} = 0$$

$$\mathrm{dS} = -\ \mathrm{VdP/T}$$

From the Gibbs function,

$$\left(\frac{\partial S}{\partial P}\right)_T = -\left(\frac{\partial V}{\partial T}\right)_P$$

$$\left(\frac{\partial S}{\partial T}\right)_P = \left(\frac{\partial S}{\partial H}\right)_P \left(\frac{\partial H}{\partial T}\right)_P = \left(\frac{\partial H}{\partial T}\right)_P \left(\frac{\partial H}{\partial S}\right)_P = Cp/T$$

$$\text{Hence, } -\ \frac{VdP}{T} = -\left(\frac{\partial V}{\partial T}\right)_P \mathrm{dP} + \frac{CvdT}{T}$$

Multiply through by T, and divide by dP, taking the infinitesimal limit as $dP \rightarrow 0$ and we arrive at

$$-\ \mathrm{V} = -\ \mathrm{T}\left(\frac{\partial V}{\partial T}\right)_P + \mathrm{Cp}\left(\frac{\partial T}{\partial P}\right)_H$$

$$\text{from which we get } \left(\frac{\partial T}{\partial P}\right)_H = \frac{1}{Cp}\left[T\left(\frac{\partial V}{\partial T}\right)_P - V\right]$$

Development: For the code in OpenModelica we use the DWSIM equations provided in the Solution Inspector as well as the VB Code are given below:

For gases, we used:

$$\mu_{\ \mathrm{JT}} = 0.0048823\ *\ Tc * \frac{18}{(Tr^2-1)(Pc*Cp*\gamma)}$$

For liquids,

$$\mu_{\ \mathrm{JT}} = -\frac{1}{(1000*\rho*Cp)}$$

Where,

$\mu_{\ \mathrm{JT}}$: Joule Thompson Coefficient

Tc: Critical Temperature of the component

Tr: Reduced Temperature

Pc: Critical Pressure

Cp: Heat Capacity at constant pressure

γ : Heat capacity ratio

ρ : Density of the component

The density of the liquid and gases are calculated using LiqDens and VapDens functions available in the Simulator.

The Joule Thompson coefficient is obviously zero for Solids.

Table:

Compound/s	Temperature (K)	Composition	Om Result (K/Pa)	DWSIM value (K/Pa)	State
Ethanol+Benzene	300	{ 0.3,0.7}	-6.1 e -10	-2.94 e-10	liquid
Water+Pyridine	298.15	{ 0.4, 0.6}	-4.5 e-10	-2.36 e-10	liquid
Phosgene	300	{ 1}	-1.8 e -07	-1.218 e-07	gas
Ammonia	298.15	{ 1}	-1.4 e-07	-1.135 e-07	Gas

1.3 Isothermal Compressibility

Description: Isothermal compressibility is defined as the fractional differential change in volume with pressure.

$$\kappa_T = \frac{-1}{V}\left(\frac{\partial V}{\partial p}\right)_T$$

Gases are highly compressible when pressure is applied to them. In the same way liquids get compressed when subjected to high pressure and isothermal compressibility is a measure of that. It is also the reciprocal of bulk modulus.

Equations:

The equations involved in the definition of isothermal compressibility include:

$$\left(\frac{\partial V_g}{\partial p}\right)_T = \frac{nRT}{p}\left(\frac{\partial z}{\partial p}\right)_T - \frac{znRT}{p^2} = \left(\frac{znRT}{p}\right)\frac{1}{z}\frac{dz}{dp} - \left(\frac{znRT}{p}\right)\frac{1}{p}$$

From the real gas equation of state,

$$\frac{1}{V_g} = \frac{p}{znRT}$$

$$\text{and } \frac{1}{V_g}\left(\frac{\partial V_g}{\partial p}\right)_T = \frac{1}{z}\frac{\mathrm{d}z}{\mathrm{d}p} - \frac{1}{p};$$

hence,

$$c_g = \frac{1}{p} - \frac{1}{z}\left(\frac{\partial z}{\partial p}\right)_T$$

For gases at low pressures, the second term is small, and the isothermal compressibility can be approximated by $c_g \approx 1/p$. Here, z is the compressibility factor and p is pressure whereas c_g is Isothermal compressibility for gases.

For liquids, the following relation was found in the following article:

Here, v_p is the parachor, n is the moles, k is the Eötvös constant (approximately equal to 2.12) and T_c is the is the critical temperature.

Development:

Isothermal compressibility of gases is calculated in DWSIM is given by the following equation:

$$\varkappa_T = \frac{1}{P} - \frac{(Z-Z1)*0.0001}{Z}$$

Where z is ideal compressibility factor equal to 1 and Z1 is calculated using the Van der Waal EOS and P is the pressure.

The other equation being used is for liquids as discussed above.

Table:

Compound/s	Temperature (K)	Om Result (1/Pa)	Data Value	State
Water	298.15	3.08 e -10	4.58 e-10	liquid
Ethanol	298.15	1.19 e -09	1.1 e-09	liquid
Carbon dioxide	298.15	9.37 e-06	-	gas
Phosgene	298.15	7.875 e -06	-	vapour

1.4 Bulk Modulus:

Description: Bulk modulus is defined as the resistance that a substance offers to compression. It is the reciprocal of isothermal compressibility. It is also defined at the ratio of infinitesimal change in pressure and the subsequent change in volume of the substance. It is equal to the pressure of a real gas and is infinite for an ideal solid. The calculation of bulk modulus of a complex anisotropic solid is done using Hooke's law.

Equations:

The bulk modulus of any substance can be defined by

$$K = -V \frac{dP}{dV}$$

Where K is bulk modulus, V is volume and P is pressure. The negative sign is there as dV is a negative value if dP is positive and it makes the value of K positive overall. But as we are dealing with mostly just liquids and gases, the definition involving Isothermal Compressibility is used.

Development: The Bulk Modulus function requires the Isothermal compressibility function as we use the following equation to calculate it:

$$K = \frac{1}{\varkappa}$$

Where $\varkappa$ is isothermal compressibility of the pure compound or mixture.

Table:

Compound/s	Temperature (K)	Om Result (m/s)	Data Value (m/s)	State
Water	298.15	3.246 e 09	2.18 e 09	liquid
Ethanol	298.15	8.403 e 08	9.09 e 08	Liquid
Carbon dioxide	298.15	1.067 e 05	-	Gas

1.5 Speed of Sound

Description: The speed of sound is the distance travelled by sound waves in unit time. Being mechanical waves, the speed of sound waves depends on the material it is travelling in. If the waves are travelling in a fluid, the speed of sound can be expressed in terms of pressure, density and specific heat capacity ratios (γ). It can also be expressed in terms of bulk modulus and density as well.

Equations:

The Newton Laplace equation is used to calculate the speed of sound for an ideal gas:

$$c = \sqrt{\gamma \frac{P}{\rho}}$$

which can be replaced with $c = \sqrt{\frac{\gamma.k.T}{m}}$ where, γ is the specific heat ratio, P is pressure, ρ is density, k is Boltzmann's constant, T is temperature and m is molecular weight.

But for a non-ideal gas and liquid, the following equation is used:

$$c = \sqrt{\frac{K}{\rho}}$$

where K is the bulk modulus.

Development:

The bulk modulus is calculated using the thermodynamic functions and then the density is calculated for the liquid or gas based on the equations and coefficients

provided in the Thermodynamic functions available for in the Simulator. The speed of sound is then calculated using the equation:

$$c = \sqrt{\frac{K}{\rho}}$$

Compound/s	**Temperature (K)**	**Om Result (m/s)**	**Data Value**	**State**
Water	298.15	1741.754	1481	Liquid
Ethanol	298.15	1034.68	1144	Liquid
Carbon dioxide	298.15	242.93	267	Gas

Chapter 2: Thermodynamic Packages

2.1 Peng Robinson Stryjek Vera 1 (PRSV1) and PRSV2

Description: The Peng Robinson Stryjek Vera 1 is an extension (1986) to the existing Peng Robinson equation of state. Instead of the attraction term being described just by the accentric factor, the PRSV 1 equation of state introduces an adjustable pure component parameter κ which is used in the polynomial fit of the accentric factor. This adjustment leads to a more accurate calculation of the compressibility factor, which leads to a better calculation of the fugacity coefficients as well as vapour and liquid phase fractions. Stryjek and Vera gave the values for these adjustable parameters for a number of compounds.

Equations:

The equation of state defined by Peng Robinson is given by:

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

Where P and V are pressure and molar volume respectively whereas a(T) and b are constants defined as follows:

$$a = 0.45724\frac{\alpha R^2 Tc^2}{Pc}$$

$$b = 0.07780\frac{\text{R Tc}}{Pc}$$

Tc and Pc are pure component critical temperature and pressure respectively and R is real gas constant.

The value of α was updated to the following equation:

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

PRSV1:

$$\kappa = \kappa_0 \quad + \quad \kappa_1\left(1+\sqrt{Tr}\right)(0.7-Tr)$$

$$\kappa_0 = 0.378893 + 1.4897153\omega - 0.17131848\omega^2 + 0.0196554\omega^3$$

PRSV2:

The second update to Peng Robinson was the PRSV2 (1986) which improved the accuracy of the calculation by adding more adjustable pure component parameters κ_1, κ_2 and κ_3 which are provided as data and used in the following equation:

$$\kappa = \kappa_0 + \left(1 + \sqrt{Tr}\right)(0.7 - Tr)\left[\kappa_1 + \kappa_2(\kappa_3 - Tr)\left(1 - \sqrt{Tr}\right.\right.$$

$$\kappa_0 = 0.378893 + 1.4897153\omega - 0.17131848\omega^2 + 0.0196554\omega^3$$

Where Tr is reduced temperature (T/Tc), ω is the accentric factor and κ_1, κ_2 and κ_3 are the adjustable pure component parameters.

Development:

The PRSV1 and PRSV2 calculations are developed as follows:

1) The values of the interaction parameters are saved in a function called BIPPR which takes the number and names of the components and returns the interaction parameter values.

2) Values for each of the above parameters $\mathbf{a}\alpha$ is calculated for every single component in the system and then the parameter a_{ij} is calculated following the mixing rules given below:

$$a_{ij} = \sum_{j=1}^{N}\sum_{i=1}^{N}(1 - kij)\sqrt{a_{ij}}$$

where kij is the interaction parameter between 2 components

3) Then the following parameters are calculated for both liquid and gas mixtures. The value of vapour and liquid mixture fractions is :

$$\text{aMliq} = \sum_{j=1}^{n}\sum_{i=1}^{n} xi_{ij} \quad \text{aMvap} = \sum_{j=1}^{n}\sum_{i=1}^{n} yi_{ij}$$

$$\text{bMliq} = \sum_{i=1}^{N} bi \quad \text{bMvap} = \sum_{i=1}^{N} bi$$

$$Aliq = \frac{aMliq*P}{(RT)^2} \quad \text{Avap} = \frac{aMvap*P}{(RT)^2}$$

$$Bliq = \frac{bMliq*P}{(RT)} \quad Bvap = \frac{bMvap*P}{(RT)}$$

4) The value of the compressibility factor is then calculated by finding the roots of the following equation:

$$Z^3 + (Bliq - 1)\,Z^2 + \left(Aliq - 2Bliq - 3Bliq^2\right) Z + \left(Bliq^3 + Bliq^2 - AliqBliq\right) = 0$$

The smallest root of this equation is considered the correct compressibility factor. This is also solved using the Avap and Bvap parameters abnd the largest root of that equation is considered to be the answer there.

$$\ln\phi = (Z-1) - \ln(Z-B) - \frac{A}{2\sqrt{2}\,B} \ln\left(\frac{Z + (\sqrt{2}+1)B}{Z - (\sqrt{2}-1)B}\right)$$

.

Where Φ is the fugacity coefficient, Z is the vapour or liquid compressibility factor and A and B are Aliq, Avap and Bliq, Bvap for the respective states.

5) The component activity coefficient is calculated as follows:

K= $\frac{phivap(fugacitycoefficientofvapor)}{philiq(fugacitycoefficientofliquid)}$ of each component

Table for PRSV1:

Compounds	Mole Fraction	Temperature	Vapour Phase Mole Fraction	Vapour Fugacity Coefficient		Liquid Fugacity Coefficient	
		kelvin K		OM Value	DWSIM	OM Value	DWSIM
Methane	0.33	180	0.5	0.9877	0.987765	17.304	16.3884
Ethane	0.33			0.961	0.962017	0.215	0.219543
Ethylene	0.34			0.968	0.968483	0.581	0.580671
Ethane	0.33	300	1	0.991	0.991	none	none
Methane	0.33			0.999	0.999	none	none
Propane	0.34			0.985	0.985	none	none
Ethane	0.33	300	1	0.992	0.992	0	0
Acetylene	0.33			0.994	0.994	0	0
Isobutane	0.34			0.977	0.977	0	0
Ethane	0.33	70	0	1	1	4.66 e-08	9.771 e -09
Methane	0.33			1	1	0.0077	0.0039
Isobutane	0.34			1	1	5.45 e-14	5.425 e -15

Table for PRSV2:

Compounds	Mole Fraction	Temperature	Vapour Phase Mole Fraction	Vapour Fugacity Coefficient		Liquid Fugacity Coefficient	
		kelvin K		OM Value	DWSIM	OM Value	DWSIM
Methane	0.5	200	0.5129	0.99	0.99	nan	nan
Isobutane	0.5			0.96	0.958	nan	nan
Propylene	0.5	200	0	none	none	0.289	0.278
Propane	0.5					0.21	0.199
Isobutane	0.5	300	1	0.974	0.974	nan	nan
N-butane	0.5			0.973	0.973	nan	nan
Ammonia	0.5	300	0.512	0.991	0.991	88.8345	88.8345
Water	0.5			0.985	0.985	0.03499	0.03499

2.2 Debugging Lee-Kesler Plocker Thermodynamic Package

Task Assigned: Debugging given Lee-Kesler Plocker Thermodynamic Package

Accomplished: The following parts of the debugging process were accomplished:

1. Found which part of the given Lee-Kesler Plocker flat model was causing issues, but could not fix it. Broke the flat model into smaller parts and made LKPTest, which is the portion of the model which gets converged
2. Changed the flat model to be compatible with the Simulator for future use. This model is named LKP.

Errors Encountered:

2.1.1. **Convergence Error**: The main error that I faced during the LKP debugging process is of convergence. The following code showed the error:

3.phi_l[i] = exp(z_l[i] - 1 - log(z_l[i]) + B_l[i] / V_r_l[i] + C_l[i] / (2 * V_r_l[i] ^ 2) + D_l[i] / (5 * V_r_l[i] ^ 5) + E_l[i]);

4. phi_v[i] = exp(z_v[i] - 1 - log(z_v[i]) + B_v[i] / V_r_v[i] + C_v[i] / (2 * V_r_v[i] ^ 2) + D_v[i] / (5 * V_r_v[i] ^ 5) + E_v[i]);

This error appeared only in LKP, which is the package made compatible with the Simulator and Material Stream. These lines of code did work in the LKPTest flat model.

2.1.2. **Partial Convergence**: Only a part of the original LKP file gets converged, from reduced volume calculation to enthalpy departure calculation.

2.1.3. **DWSIM Values**: Lee Kesler Plocker uses certain mixing rules to calculate constants and parameters which are used later to calculated fugacity coefficients of the reference and simple liquid. These values are then used in a separate equation to give the final value of fugacity coefficient of the liquid and vapour mixture using the

vapour and liquid mixture fractions calculated. These mixture values could not be found in DWSIM for cross-checking. The fugacity coefficient was calculated in DWSIM using another equation, by finding individual fugacity coefficients and finding their molar average in mixture.

Description: Lee-Kesler Plocker developed the value of compressibility factor to be determined from the following equation:

$$Z = \left(\frac{P_r V_r}{T_r}\right) = 1 + \frac{B}{V_r} + \frac{C}{V_r^2} + \frac{D}{V_r^5} + \frac{c_4}{T_r^3 V_r^2}\left(\beta + \frac{\gamma}{V_r^2}\right)\exp\left(-\frac{\gamma}{V_r^2}\right)$$

Where:

$$B = b_1 - b_2/T_r - b_3/T_r^2 - b_4/T_r^3$$

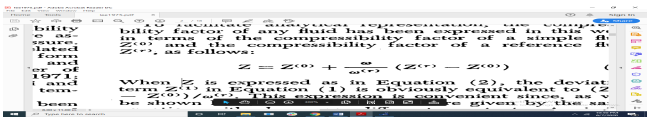
$$C = c_1 - c_2/T_r + c_3/T_r^3$$

$$D = d_1 + d_2/T_r$$

The constants are taken from the following table:

Constant	Simple fluids	Reference fluids	Constant	Simple fluids	Reference fluid
b_1	0.1181193	0.2026579	c_3	0.0	0.016901
b_2	0.265728	0.331511	c_4	0.042724	0.041577
b_3	0.154790	0.027655	$d_1 \times 10^4$	0.155488	0.48736
b_4	0.030323	0.203488	$d_2 \times 10^4$	0.623689	0.0740336
c_1	0.0236744	0.0313385	β	0.65392	1.226
c_2	0.0186984	0.0503618	γ	0.060167	0.03754

To facilitate analytical representation, the compressibility factor of any fluid has been expressed in this work in terms of the compressibility factor of a simple fluid (Z_0) and the compressibility factor of a reference fluid (Z_r), as follows:



To improve overall accuracy, several sets of mixing rules have been studied. The following set of mixing rules have been found to give the best results:

$$V_{ci} = Z_{ci} R T_{ci} / P_{ci}$$

$$Z_{ci} = 0.2905 - 0.085\omega_i$$

$$V_c = \frac{1}{8} \sum_j \sum_k x_j x_k (V_{cj}^{1/3} + V_{ck}^{1/3})^3$$

$$T_c = \frac{1}{8V_c} \sum_j \sum_k x_j x_k (V_{cj}^{1/3} + V_{ck}^{1/3})^3 \sqrt{T_{cj} T_{ck}}$$

$$\omega = \sum_j x_j \omega_j$$

$$P_c = Z_c R T_c / V_c = (0.2905 - 0.085\omega)\, R T_c / V_c$$

Calculation of fugacity follows this equation:

$$\ln\left(\frac{f}{P}\right) = Z - 1 - \ln(Z) + \frac{B}{V_r} + \frac{C}{2V_r^2} + \frac{D}{5V_r^5} + E$$

Here, E is a parameter defined as:

$$\ln\left(\frac{f}{P}\right) = Z - 1 - \ln(Z) + \frac{B}{V_r} + \frac{C}{2V_r^2} + \frac{D}{5V_r^5}$$

where

$$E = \frac{c_4}{2T_r^3\gamma}\left\{\beta + 1 - \left(\beta + 1 + \frac{\gamma}{V_r^2}\right)\exp\left(-\frac{\gamma}{V_r^2}\right)\right\}$$

Enthalpy Departure follows:

$$\frac{H - H^*}{RT_c} = T_r \left\{ Z - 1 - \frac{b_2 + 2b_3/T_r + 3b_4/T_r^2}{T_r V_r} - \frac{c_2 - 3c_3/T_r^2}{2T_r V_r^2} + \frac{d_2}{5T_r V_r^5} + 3E \right\}$$

c. Entropy departure

Where:

bl, b2, bs, bq = constants as given in Table

cl, c2, c3, c4 = constants as given in Table

dl, d2 = constants as given in Table B, C, D = coefficients

C_p = isobaric heat capacity C_v = isochoric heat capacity

E= defined above f = fugacity

H = enthalpy P = pressure

Po=reference R = gas constant

S = entropy T = temperature

V = volume $V_r = P_c V/RT_c$,

x = molar composition Z = compressibility factor

Development and Issues faced:

The values for reference and simple fluid were calculated separately and then used in the given equations for vapour and liquid phase mixtures separately. All the mixing rules and other equations related to fugacity and enthalpy are calculated separately for vapour and liquid mixtures.

Code:

- **LKP Test:** This is the flat model which gives us results for fugacity coefficients, enthalpy deviation and compressibility factor.

Table for LKP Test:

COMPOUND SYSTEM	MOLE FRACTION IN FEED	LIQUID MOLE FRACTION	VAPOUR MOLE FRACTION	VAPOUR MIXTURE FUGACITY COEFF	LIQUID MIXTURE FUGACITY COEFF	Temperature (K)
N-Butane	0.5	0.3590478	0.85280566	0.9657	0.5034	300
N-Heptane	0.5	0.6409522	0.14719434			
Ethanol	0.5	0.5	0.319988	0.9601	0.00194	300
Methanol	0.5	0.5	0.680012			
Acetone	0.5	0.5	0.949496	0.9552	0.00265	300
Acetic Acid	0.5	0.5	0.050504			

- **Package LKP:** This is the package that is compatible with the Simulator but does not simulate due to a calculation/runtime error with the logarithmic function used in the fugacity coefficient calculation. I could not find a way to resolve it.

OpenModelica Code

Surface Tension

```
package SurfaceTension
  function SurfTens
    extends Modelica.Icons.Function;
    import Simulator.Files.Thermodynamic_Functions.*;
    input Real T, coeff[6];
    output Real sigma;
  algorithm
    sigma := coeff[2] + exp(coeff[3] / T + coeff[4] + coeff[5] * T + coeff[6] * T
        * T);
  end SurfTens;

      model Test
              import Simulator.*;
              import data = Simulator.Files.ChemsepDatabase;
              parameter data.Benzene ace;
              parameter data.Phenol meth;
              parameter data.Aniline benz;
              parameter Integer Nc = 3;
              parameter data.GeneralProperties C[Nc] = {ace, meth, benz};
              Real ST;
              parameter Real x[Nc] = {0.33, 0.33, 0.34}, P = 101325, T = 300;
              Real R[Nc];
      equation
              for i in 1:Nc loop
                      if T > C[i].SigmaT[1] and T < C[i].SigmaT[2] then
                              R[i] = SurfaceTension.SurfTens(T, C[i].Sigma);
                      else
                              R[i] = SurfaceTension.SurfTensEmp(P, T, C[i].Pc /
                                  100000, C[i].Tc, C[i].Tb) / 1000;
                      end if;
              end for;
              ST = sum(x[:] .* R[:]);
      end Test;


  function SurfTensEmp
    extends Modelica.Icons.Function;
    import Simulator.Files.Thermodynamic_Functions.*;
    input Real P, T;
    input Real Pc, Tc, Tb;
    output Real sigma;
  protected
    Real Tr, Tbr;
    Real alpha;
  algorithm
    Tr := T / Tc;
    Tbr := Tb / Tc;
    alpha := 0.9076 * (1 + Tbr * log(Pc / 1.01325) / (1 - Tbr));
    sigma := Pc ^ (2 / 3) * Tc ^ (1 / 3) * (0.132 * alpha - 0.279) * (1 - Tr) ^
        (11 / 9);
  end SurfTensEmp;
end SurfaceTension;
```

Joule Thompson Coefficient

```
package JouleThompsonCoefficient
  function JTCoeff
    extends Modelica.Icons.Function;
    import Simulator.Files.Thermodynamic_Functions.*;
    input Real P, T, a;
    input Real Pc, Tc, Cp, rho;
    Real Tr, gamma;
    output Real mu;
  algorithm
    Tr := T / Tc;
    gamma := Cp / (Cp - 8.314);
    if a == 1 then
      mu := 0.0048823 * Tc * (18 / (Tr ^ 2 - 1)) / (Pc * Cp * gamma);
    elseif a == 2 then
      mu := -1 / (1000 * rho * Cp);
    else
      mu := 0;
    end if;
//compound is a gas
  end JTCoeff;

      model Test
              import Simulator.*;
              import data = Simulator.Files.ChemsepDatabase;
              parameter Integer Nc = 2;
              parameter data.Water eth;
              parameter data.Pyridine benz;
              parameter data.GeneralProperties C[Nc] = {eth, benz};
              Real Pc[Nc], Tc[Nc], Tb[Nc], Tm[Nc], rho[Nc], Cp[Nc];
              Integer a[Nc];
              Real P, T;
              Real mu[Nc], x[Nc] = {0.4, 0.6};
              Real JTC;
      equation
              P = 101325;
              T = 298.15;
              for i in 1:Nc loop
                      Pc[i] = C[i].Pc;
                      Tc[i] = C[i].Tc;
                      Tb[i] = C[i].Tb;
                      Tm[i] = C[i].Tm;
                      if T >= Tb[i] then
                              a[i] = 1;
                              rho[i] = 0;
                              Cp[i] = Files.ThermodynamicFunctions.VapCpId(C[i].
                                  VapCp, T);
                      elseif T < Tb[i] and T >= Tm[i] then
                              a[i] = 2;
                              Cp[i] = Files.ThermodynamicFunctions.LiqCpId(C[i].
                                  LiqCp, T);
                              rho[i] = Files.ThermodynamicFunctions.Dens(C[i].
                                  LiqDen, Tc[i], T, P);
                      else
                              a[i] = 3;
                              Cp[i] = 0;
                              rho[i] = 0;
                      end if;
                      mu[i] = JouleThompsonCoefficient.JTCoeff(P, T, a[i], Pc[i],
                          Tc[i], Cp[i], rho[i]);
```

```
56                end for;
57                JTC = sum(x[:] .* mu[:]);
58        end Test;
59
60
61  end JouleThompsonCoefficient;
```

beginjustify # Isothermal Compressibility

```
1   package IsothermalCompressibility
2     function CalcIsoComp
3       extends Modelica.Icons.Function;
4       import Simulator.Files.Thermodynamic_Functions.*;
5       input Real P, Z, Z1;
6       output Real K;
7     algorithm
8       K := 1 / P - (Z - Z1) / Z * 0.0001;
9     end CalcIsoComp;
10
11    model Test
12      import Simulator.*;
13      import data = Simulator.Files.ChemsepDatabase;
14      parameter data.Carbondioxide ace;
15      parameter Integer Nc = 1;
16      parameter data.GeneralProperties C[Nc] = {ace};
17      parameter Real T = 298.15, x[Nc] = {1}, P = 101325;
18      Real Z, ST, rho;
19      Real K[Nc], IC;
20    equation
21      for i in 1:Nc loop
22        if T > C[i].Tb then
23          rho = 0;
24          ST = 0;
25          Z = CompressibilityFactor(T, 101325, C[i].Tc, C[i].Pc);
26          K[i] = CalcIsoComp(101325, 1, Z);
27        elseif T > C[i].Tm and T <= C[i].Tb then
28          if T > C[i].SigmaT[1] and T < C[i].SigmaT[2] then
29            ST = SurfaceTension.SurfTens(T, C[i].Sigma);
30          else
31            ST = SurfaceTension.SurfTensEmp(P, T, C[i].Pc / 100000, C[i].Tc, C[i].Tb
                  ) / 1000;
32          end if;
33          Z = 0;
34          rho = IsothermalCompressibility.DensityRacket(T, P, C[i].Pc, C[i].Tc, C[i
                ].Racketparam, C[i].AF, C[i].MW, Files.ThermodynamicFunctions.Psat(C[i
                ].VP, T));
35          K[i] = IsothermalCompressibility.CalcIsoCompLiq(ST, C[i].MW, rho, C[i].Tc,
                 T);
36        else
37          ST = 0;
38          Z = 0;
39          rho = 0;
40          K[i] = 0;
41        end if;
42      end for;
43      IC = sum(x[:] .* K[:]);
44    end Test;
45
46    function CompressibilityFactor
47      extends Modelica.Icons.Function;
```

```
48      import Simulator.Files.ThermodynamicFunctions.*;
49      parameter Real R = 8.314;
50      input Real T, P, Tc, Pc;
51      Real V[3, 2], a, b, D[4];
52      output Real Z;
53    algorithm
54      a := IsothermalCompressibility.EOSConstants(Tc, Pc, T);
55      b := IsothermalCompressibility.EOSConstantII(Tc, Pc);
56      D[1] := 1;
57      D[2] := (-8.314 * T) / P;
58      D[3] := (-b ^ 2) - 8.314 * T * b / P + a / P;
59      D[4] := (-a * b) / P;
60      V := Modelica.Math.Vectors.Utilities.roots(D);
61      Z := P * V[1, 1] / (8.314 * T);
62    end CompressibilityFactor;
63
64    function CalcIsoCompLiq
65      extends Modelica.Icons.Function;
66      import Simulator.Files.Thermodynamic_Functions.*;
67      input Real ST, MW, rho, Tc, T;
68      output Real IC;
69    protected
70      Real Vp;
71    algorithm
72      Vp := ST ^ 0.25 * MW / rho;
73      IC := 1.33 * 10 ^ (-2) * Vp ^ 1.2 / (2.12 ^ 1.8 * (Tc - T) ^ 1.8);
74    end CalcIsoCompLiq;
75
76    function EOSConstants
77      extends Modelica.Icons.Function;
78      parameter Real R_gas = 8.314;
79      input Real Tc, Pc;
80      input Real T;
81      output Real a;
82    algorithm
83      a := 0.42748 * R_gas ^ 2 * (Tc ^ 2.5 / (Pc * T ^ 0.5));
84    end EOSConstants;
85
86    function EOSConstantII
87      extends Modelica.Icons.Function;
88      parameter Real R_gas = 8.314;
89      input Real Tc, Pc;
90      output Real b;
91    algorithm
92      b := 0.08664 * R_gas * (Tc / Pc);
93    end EOSConstantII;
94
95    function DensityRacket
96      extends Modelica.Icons.Function;
97      input Real T;
98      input Real P;
99      input Real Pc_c;
100     input Real Tc_c;
101     input Real RP_c;
102     input Real AF_c;
103     input Real MW_c;
104     input Real Psat;
105     output Real rho_c;
106     parameter Real R = 83.14;
107   protected
108     Real Tr_c, Pcbar_c, temp, Tcor_c, a, b, c_c, d, e_c, Beta_c, f, g, h, j, k,
          RPnew_c;
```

```
  algorithm
    Pcbar_c := Pc_c / 100000;
    Tr_c := T / Tc_c;
    if Tr_c > 0.99 then
      Tr_c := 0.5;
    end if;
    if RP_c == 0 then
      RPnew_c := 0.29056 - 0.08775 * AF_c;
    else
      RPnew_c := RP_c;
    end if;
    temp := R * (Tc_c / Pcbar_c) * RPnew_c ^ (1 + (1 - Tr_c) ^ (2 / 7));
    if T < Tc_c then
      a := -9.070217;
      b := 62.45326;
      d := -135.1102;
      f := 4.79594;
      g := 0.250047;
      h := 1.14188;
      j := 0.0861488;
      k := 0.0344483;
      e_c := exp(f + g * AF_c + h * AF_c * AF_c);
      c_c := j + k * AF_c;
      Beta_c := Pc_c * ((-1) + a * (1 - Tr_c) ^ (1 / 3) + b * (1 - Tr_c) ^ (2 / 3)
          + d * (1 - Tr_c) + e_c * (1 - Tr_c) ^ (4 / 3));
      Tcor_c := temp * (1 - c_c * log((Beta_c + P) / (Beta_c + Psat)));
      rho_c := 0.001 * MW_c / (Tcor_c * 0.000001);
    else
      rho_c := 0.001 * MW_c / (temp * 0.000001);
    end if;
  end DensityRacket;
end IsothermalCompressibility;
```

Bulk Modulus

```
package BulkModulus
  function BM
    extends Modelica.Icons.Function;
    import Simulator.Files.Thermodynamic_Functions.*;
    input Real K;
    output Real B;
  algorithm
    B := 1 / K;
  end BM;

  model Test
    import Simulator.*;
    import data = Simulator.Files.ChemsepDatabase;
    parameter data.Phosgene ace;
    parameter Integer Nc = 1;
    parameter data.GeneralProperties C[Nc] = {ace};
    parameter Real T = 350, x[Nc] = {1};
    Real Z[Nc];
    Real K[Nc], IC, BM;
  equation
    for i in 1:Nc loop
      Z[i] = IsothermalCompressibility.CompressibilityFactor(T, 101325, C[i].Tc, C
          [i].Pc);
      K[i] = IsothermalCompressibility.CalcIsoComp(101325, 1, Z[i]);
    end for;
```

```
25      IC = sum(x[:] .* K[:]);
26      BM = BulkModulus.BM(IC);
27    end Test;
28  end BulkModulus;
```

Speed OfS ound

```
1   package SpeedOfSound
2       function SoundSpeed
3         extends Modelica.Icons.Function;
4         import Simulator.Files.Thermodynamic_Functions.*;
5         input Real B, rho;
6         output Real S;
7       algorithm
8         S := sqrt(B / rho);
9       end SoundSpeed;
10
11
12          model Test
13           import Simulator.*;
14          import data = Simulator.Files.ChemsepDatabase;
15          parameter data.Water ace;
16          parameter Integer Nc = 1;
17          parameter data.GeneralProperties C[Nc] = {ace};
18          parameter Real T = 298.15, x[Nc] = {1}, P=101325;
19          Real Z[Nc];
20          Real K[Nc], IC, BM, S, rho[Nc], Density;
21          equation
22          for i in 1:Nc loop
23          Z[i] = IsothermalCompressibility.CompressibilityFactor(T, P, C[i].Tc, C[i].
                Pc);
24          K[i] = IsothermalCompressibility.CalcIsoComp(P,1, Z[i]);
25          if T <= C[i].Tm then
26          rho[i] = C[i].SolDen[2] + C[i].SolDen[3] * T;
27          else
28          rho[i] = Files.ThermodynamicFunctions.Dens(C[i].LiqDen, C[i].Tc, T, P);
29                                                                                  end
                                                                                      if
                                                                                      ;
30           end for;
31            IC = sum(x[:] .* K[:]);
32           Density = sum(x[:] .* rho[:]);
33           BM = BulkModulus.BM(IC);
34          S = SpeedOfSound.SoundSpeed(BM, Density);
35                    end Test;
36
37
38
39
40  end SpeedOfSound;
```

PVSV

```
1   package PRSVTest
2     model PRSV1
```

```
3       //=============================================================================
4       //Header files and Parameters
5       import Simulator.*;
6       parameter Real R = 8.314 "Ideal Gas Constant";
7       parameter Real kij_c[Nc, Nc](each start = 1) =
            Simulator.Files.ThermodynamicFunctions.BIPPR(Nc, C.name);
8       //=============================================================================
9       //Model Variables
10      Real Tr_c[Nc](each start = Tg) "Reduced temperature";
11      Real b_c[Nc];
12      Real a_c[Nc](start = xg);
13      Real m_c[Nc];
14      Real q_c[Nc];
15      Real p_c[Nc];
16      Real aij_c[Nc, Nc];
17      Real K_c[Nc](start = K_guess);
18      Real Pvap_c[Nc](start = Pg) "Saturated Vapor Pressure";
19      Real philiq_c[Nc](each start = 5) "Liquid Phase Fugasity coefficient";
20      Real phivap_c[Nc](each start = 5) "Vapor Phase Fugasity coefficient";
21      Real gmabubl_c[Nc], gmadew_c[Nc];
22      Real philiqbubl_c[Nc], phivapdew_c[Nc];
23      Real Cpres_p[3], Hres_p[3], Sres_p[3];
24      Real aMliq, bMliq;
25      Real Aliq(start = xliqg), Bliq(start = xvapg);
26      Real Cliq[4];
27      Real Z_RL[3, 2](start = xliqg);
28      Real Zliq[3](start = xliqg), Zll(start = xvapg);
29      Real sumxliq[Nc];
30      Real aMvap, bMvap;
31      Real Avap(start = xliqg), Bvap(start = xvapg);
32      Real Cvap[4];
33      Real Z_RV[3, 2](start = xvapg);
34      Real Zvap[3](start = xvapg), Zvv;
35      Real sumxvap[Nc];
36      Real A, B, Cdummy, D_c[Nc], E, F, G, H_c[Nc], I_c[Nc], J_c[Nc];
37      Real gma[Nc];
38      extends GuessModels.InitialGuess;
39      //=============================================================================
40    equation
41      for i in 1:Nc loop
42        Pvap_c[i] = Simulator.Files.ThermodynamicFunctions.Psat(C[i].VP, T);
43        gmadew_c[i] = 1;
44        gmabubl_c[i] = 1;
45        philiqbubl_c[i] = 1;
46        phivapdew_c[i] = 1;
47        gma[i] = 1;
48      end for;
49      Cpres_p[:] = zeros(3);
50      Hres_p[:] = zeros(3);
51      Sres_p[:] = zeros(3);
52      Tr_c = T ./ C.Tc;
53      b_c = 0.0778 * R * C.Tc ./ C.Pc;
54      m_c[1] = p_c[1] + 0.00159 * (1.0 + Tr_c[1] ^ 0.5) * (0.7 - Tr_c[1]);
55      m_c[2] = p_c[2] - 0.02669 * (1.0 + Tr_c[2] ^ 0.5) * (0.7 - Tr_c[2]);
56      m_c[3] = p_c[3] - 0 * (1.0 + Tr_c[3] ^ 0.5) * (0.7 - Tr_c[3]);
57      q_c = 0.45724 * R ^ 2 * C.Tc .^ 2 ./ C.Pc;
58      a_c = q_c .* (1 .+ m_c .* (1 .- sqrt(Tr_c))) .^ 2;
59      p_c = 0.378893 .+ 1.4897153 * C.AF .- 0.17131848 * C.AF .^ 2 .+ 0.0196554 *
            C.AF .^ 3;
60      aij_c = {{(1 - kij_c[i, j]) * sqrt(a_c[i] * a_c[j]) for i in 1:Nc} for j in 1:
            Nc};
61  //=============================================================================
```

```
//Liquid_Fugacity Coefficient Calculation Routine
    aMliq = sum({{x_pc[2, i] * x_pc[2, j] * aij_c[i, j] for i in 1:Nc} for j in 1:
        Nc});
    bMliq = sum(b_c .* x_pc[2, :]);
    Aliq = aMliq * P / (R * T) ^ 2;
    Bliq = bMliq * P / (R * T);
    Cliq[1] = 1;
    Cliq[2] = Bliq - 1;
    Cliq[3] = Aliq - 3 * Bliq ^ 2 - 2 * Bliq;
    Cliq[4] = Bliq ^ 3 + Bliq ^ 2 - Aliq * Bliq;
    Z_RL = Modelica.Math.Vectors.Utilities.roots(Cliq);
    Zliq = {Z_RL[i, 1] for i in 1:3};
    Zll = min({Zliq});
    sumxliq = {sum({x_pc[2, j] * aij_c[i, j] for j in 1:Nc}) for i in 1:Nc};
    if Zll + 2.4142135 * Bliq <= 0 then
      A = 1;
    else
      A = Zll + 2.4142135 * Bliq;
    end if;
    if Zll - 0.414213 * Bliq <= 0 then
      B = 1;
    else
      B = Zll - 0.414213 * Bliq;
    end if;
    if Zll - Bliq <= 0 then
      Cdummy = 0;
    else
      Cdummy = log(Zll - Bliq);
    end if;
    for i in 1:Nc loop
      if bMliq == 0 then
        D_c[i] = 0;
      else
        D_c[i] = b_c[i] / bMliq;
      end if;
    end for;
    for i in 1:Nc loop
      if aMliq == 0 then
        J_c[i] = 0;
      else
        J_c[i] = sumxliq[i] / aMliq;
      end if;
    end for;
    philiq_c = exp(Aliq / (Bliq * sqrt(8)) * log(A / B) .* (D_c .- 2 * J_c) .+ (
        Zll - 1) * D_c .- Cdummy);
//=============================================================================
//Vapour Fugacity Calculation Routine
    aMvap = sum({{x_pc[3, i] * x_pc[3, j] * aij_c[i, j] for i in 1:Nc} for j in 1:
        Nc});
    bMvap = sum(b_c .* x_pc[3, :]);
    Avap = aMvap * P / (R * T) ^ 2;
    Bvap = bMvap * P / (R * T);
    Cvap[1] = 1;
    Cvap[2] = Bvap - 1;
    Cvap[3] = Avap - 3 * Bvap ^ 2 - 2 * Bvap;
    Cvap[4] = Bvap ^ 3 + Bvap ^ 2 - Avap * Bvap;
    Z_RV = Modelica.Math.Vectors.Utilities.roots(Cvap);
    Zvap = {Z_RV[i, 1] for i in 1:3};
    Zvv = max({Zvap});
    sumxvap = {sum({x_pc[3, j] * aij_c[i, j] for j in 1:Nc}) for i in 1:Nc};
    if Zvv + 2.4142135 * Avap <= 0 then
      E = 1;
```

```
121       else
122         E = Zvv + 2.4142135 * Bvap;
123       end if;
124       if Zvv - 0.414213 * Bvap <= 0 then
125         F = 1;
126       else
127         F = Zvv - 0.414213 * Bvap;
128       end if;
129       if Zvv - Bvap <= 0 then
130         G = 0;
131       else
132         G = log(Zvv - Bvap);
133       end if;
134       for i in 1:Nc loop
135         if bMvap == 0 then
136           H_c[i] = 0;
137         else
138           H_c[i] = b_c[i] / bMvap;
139         end if;
140       end for;
141       for i in 1:Nc loop
142         if aMvap == 0 then
143           I_c[i] = 0;
144         else
145           I_c[i] = sumxvap[i] / aMvap;
146         end if;
147       end for;
148       phivap_c = exp(Avap / (Bvap * sqrt(8)) * log(E / F) .* (H_c .- 2 * I_c) .+ (
              Zvv - 1) * H_c .- G);
149       for i in 1:Nc loop
150         if philiq_c[i] == 0 or phivap_c[i] == 0 then
151           K_c[i] = 0;
152         else
153           K_c[i] = philiq_c[i] / phivap_c[i];
154         end if;
155       end for;
156   //
      ================================================================================

157     end PRSV1;
158
159     model ms
160       extends Simulator.Streams.MaterialStream;
161       extends PRSVTest.PRSV1;
162     end ms;
163
164     model Test
165       extends Modelica.Icons.Example;
166       import data = Simulator.Files.ChemsepDatabase;
167       parameter Integer Nc = 3;
168       parameter data.Methane ace;
169       parameter data.Ethane eth;
170       parameter data.Noctane noct;
171       parameter data.GeneralProperties C[Nc] = {ace, eth, noct};
172       PRSVTest.ms S1(C = C, Nc = Nc) annotation(
173         Placement(visible = true, transformation(origin = {-128, 14}, extent =
                {{-10, -10}, {10, 10}}, rotation = 0)));
174     equation
175       S1.F_p[1] = 100;
176       S1.x_pc[1, :] = {0.33, 0.33, 0.34};
177       S1.P = 101325;
178       S1.T = 150;
```

```
179   end Test;
180
181   model PRSV2
182     //=============================================================
183     //Header files and Parameters
184     import Simulator.*;
185     parameter Real R = 8.314 "Ideal Gas Constant";
186     parameter Real kij_c[Nc, Nc](each start = 1) =
          Simulator.Files.ThermodynamicFunctions.BIPPR(Nc, C.name);
187     //=============================================================
188     //Model Variables
189     Real Tr_c[Nc](each start = Tg) "Reduced temperature";
190     Real b_c[Nc];
191     Real a_c[Nc](start = xg);
192     Real m_c[Nc];
193     Real q_c[Nc];
194     Real p_c[Nc];
195     Real aij_c[Nc, Nc];
196     Real K_c[Nc](start = K_guess);
197     Real Pvap_c[Nc](start = Pg) "Saturated Vapor Pressure";
198     Real philiq_c[Nc](each start = 5) "Liquid Phase Fugasity coefficient";
199     Real phivap_c[Nc](each start = 5) "Vapor Phase Fugasity coefficient";
200     Real gmabubl_c[Nc], gmadew_c[Nc];
201     Real philiqbubl_c[Nc], phivapdew_c[Nc];
202     Real Cpres_p[3], Hres_p[3], Sres_p[3];
203     Real aMliq, bMliq;
204     Real Aliq(start = xliqg), Bliq(start = xvapg);
205     Real Cliq[4];
206     Real Z_RL[3, 2](start = xliqg);
207     Real Zliq[3](start = xliqg), Zll(start = xvapg);
208     Real sumxliq[Nc];
209     Real aMvap, bMvap;
210     Real Avap(start = xliqg), Bvap(start = xvapg);
211     Real Cvap[4];
212     Real Z_RV[3, 2](start = xvapg);
213     Real Zvap[3](start = xvapg), Zvv;
214     Real sumxvap[Nc];
215     Real A, B, Cdummy, D_c[Nc], E, F, G, H_c[Nc], I_c[Nc], J_c[Nc];
216     Real gma[Nc];
217     extends GuessModels.InitialGuess;
218     //=============================================================
219   equation
220     for i in 1:Nc loop
221       Pvap_c[i] = Simulator.Files.ThermodynamicFunctions.Psat(C[i].VP, T);
222       gmadew_c[i] = 1;
223       gmabubl_c[i] = 1;
224       philiqbubl_c[i] = 1;
225       phivapdew_c[i] = 1;
226       gma[i] = 1;
227     end for;
228     Cpres_p[:] = zeros(3);
229     Hres_p[:] = zeros(3);
230     Sres_p[:] = zeros(3);
231     Tr_c = T ./ C.Tc;
232     b_c = 0.0778 * R * C.Tc ./ C.Pc;
233     m_c[1] = p_c[1] + (0 + 0 * (0 - Tr_c[1])) * (1.0 + Tr_c[1] ^ 0.5) * (0.7 -
          Tr_c[1]);
234     m_c[2] = p_c[2] + ((-0) + 0 * (0 - Tr_c[2])) * (1.0 + Tr_c[2] ^ 0.5) * (0.7 -
          Tr_c[2]);
235     q_c = 0.45724 * R ^ 2 * C.Tc .^ 2 ./ C.Pc;
236     a_c = q_c .* (1 .+ m_c .* (1 .- sqrt(Tr_c))) .^ 2;
237     p_c = 0.378893 .+ 1.4897153 * C.AF .- 0.17131848 * C.AF .^ 2 .+ 0.0196554 *
```

```
        C.AF .^ 3;
238     aij_c = {{(1 - kij_c[i, j]) * sqrt(a_c[i] * a_c[j]) for i in 1:Nc} for j in 1:
            Nc};
239 //=============================================================================
240 //Liquid_Fugacity Coefficient Calculation Routine
241     aMliq = sum({{x_pc[2, i] * x_pc[2, j] * aij_c[i, j] for i in 1:Nc} for j in 1:
            Nc});
242     bMliq = sum(b_c .* x_pc[2, :]);
243     Aliq = aMliq * P / (R * T) ^ 2;
244     Bliq = bMliq * P / (R * T);
245     Cliq[1] = 1;
246     Cliq[2] = Bliq - 1;
247     Cliq[3] = Aliq - 3 * Bliq ^ 2 - 2 * Bliq;
248     Cliq[4] = Bliq ^ 3 + Bliq ^ 2 - Aliq * Bliq;
249     Z_RL = Modelica.Math.Vectors.Utilities.roots(Cliq);
250     Zliq = {Z_RL[i, 1] for i in 1:3};
251     Zll = min({Zliq});
252     sumxliq = {sum({x_pc[2, j] * aij_c[i, j] for j in 1:Nc}) for i in 1:Nc};
253     if Zll + 2.4142135 * Bliq <= 0 then
254       A = 1;
255     else
256       A = Zll + 2.4142135 * Bliq;
257     end if;
258     if Zll - 0.414213 * Bliq <= 0 then
259       B = 1;
260     else
261       B = Zll - 0.414213 * Bliq;
262     end if;
263     if Zll - Bliq <= 0 then
264       Cdummy = 0;
265     else
266       Cdummy = log(Zll - Bliq);
267     end if;
268     for i in 1:Nc loop
269       if bMliq == 0 then
270         D_c[i] = 0;
271       else
272         D_c[i] = b_c[i] / bMliq;
273       end if;
274     end for;
275     for i in 1:Nc loop
276       if aMliq == 0 then
277         J_c[i] = 0;
278       else
279         J_c[i] = sumxliq[i] / aMliq;
280       end if;
281     end for;
282     philiq_c = exp(Aliq / (Bliq * sqrt(8)) * log(A / B) .* (D_c .- 2 * J_c) .+ (
            Zll - 1) * D_c .- Cdummy);
283 //=============================================================================
284 //Vapour Fugacity Calculation Routine
285     aMvap = sum({{x_pc[3, i] * x_pc[3, j] * aij_c[i, j] for i in 1:Nc} for j in 1:
            Nc});
286     bMvap = sum(b_c .* x_pc[3, :]);
287     Avap = aMvap * P / (R * T) ^ 2;
288     Bvap = bMvap * P / (R * T);
289     Cvap[1] = 1;
290     Cvap[2] = Bvap - 1;
291     Cvap[3] = Avap - 3 * Bvap ^ 2 - 2 * Bvap;
292     Cvap[4] = Bvap ^ 3 + Bvap ^ 2 - Avap * Bvap;
293     Z_RV = Modelica.Math.Vectors.Utilities.roots(Cvap);
294     Zvap = {Z_RV[i, 1] for i in 1:3};
```

```
295       Zvv = max({Zvap});
296       sumxvap = {sum({x_pc[3, j] * aij_c[i, j] for j in 1:Nc}) for i in 1:Nc};
297       if Zvv + 2.4142135 * Avap <= 0 then
298         E = 1;
299       else
300         E = Zvv + 2.4142135 * Bvap;
301       end if;
302       if Zvv - 0.414213 * Bvap <= 0 then
303         F = 1;
304       else
305         F = Zvv - 0.414213 * Bvap;
306       end if;
307       if Zvv - Bvap <= 0 then
308         G = 0;
309       else
310         G = log(Zvv - Bvap);
311       end if;
312       for i in 1:Nc loop
313         if bMvap == 0 then
314           H_c[i] = 0;
315         else
316           H_c[i] = b_c[i] / bMvap;
317         end if;
318       end for;
319       for i in 1:Nc loop
320         if aMvap == 0 then
321           I_c[i] = 0;
322         else
323           I_c[i] = sumxvap[i] / aMvap;
324         end if;
325       end for;
326       phivap_c = exp(Avap / (Bvap * sqrt(8)) * log(E / F) .* (H_c .- 2 * I_c) .+ (
              Zvv - 1) * H_c .- G);
327       for i in 1:Nc loop
328         if philiq_c[i] == 0 or phivap_c[i] == 0 then
329           K_c[i] = 0;
330         else
331           K_c[i] = philiq_c[i] / phivap_c[i];
332         end if;
333       end for;
334   //
```

```
335     end PRSV2;
336   end PRSVTest;
```

Lee Kesler Plocker Package

```
1   package LKP
2     model ms
3       extends Simulator.Streams.MaterialStream;
4       extends LKP.LKPTest;
5     end ms;
6
7     model LKPTest
8       import Simulator.*;
9       import data = Simulator.Files.ChemsepDatabase;
10      //constants for simple fluid
11      parameter Real b1[2] = {0.1181193, 0.2026579} "Constants for the
            BWR-Lee_Kesler Equation";
```

```
12      parameter Real b2[2] = {0.265728, 0.331511} "Constants for the BWR-Lee_Kesler
            Equation";
13      parameter Real b3[2] = {0.154790, 0.027655} "Constants for the BWR-Lee_Kesler
            Equation";
14      parameter Real b4[2] = {0.030323, 0.203488} "Constants for the BWR-Lee_Kesler
            Equation";
15      parameter Real c1[2] = {0.0236744, 0.0313385} "Constants for the
            BWR-Lee_Kesler Equation";
16      parameter Real c2[2] = {0.0186984, 0.0503618} "Constants for the
            BWR-Lee_Kesler Equation";
17      parameter Real c3[2] = {0, 0.016901} "Constants for the BWR-Lee_Kesler
            Equation";
18      parameter Real c4[2] = {0.042724, 0.041577} "Constants for the BWR-Lee_Kesler
            Equation";
19      parameter Real d1[2] = {0.0000155488, 0.000048736} "Constants for the
            BWR-Lee_Kesler Equation";
20      parameter Real d2[2] = {0.0000623689, 0.00000740336} "Constants for the
            BWR-Lee_Kesler Equation";
21      parameter Real beta[2] = {0.65392, 1.226} "Constants for the BWR-Lee_Kesler
            Equation";
22      parameter Real gamma[2] = {0.060167, 0.03754} "Constants for the
            BWR-Lee_Kesler Equation";
23      parameter Real omega_r = 0.397 "acentric factor of N-octane";
24      parameter Real R = 8.314472 " Universal gas constant";
25      parameter data.GeneralProperties comp[:] = C[:];
26      parameter Real kij[Nc, 2] = {{1, 1.018}, {1.018, 1}} " binary interaction
            parameter for the i-j mixture rule for the pseudo critical temperature";
27      Real Cpres_p[3], Hres_p[3], Sres_p[3];
28      Real T_c[Nc] = comp[:].Tc "Critical Temperature";
29      Real P_c[Nc] = comp[:].Pc "critical Pressure";
30      Real V_c[Nc] = comp[:].Vc "Critical Volume";
31      Real omega[Nc] = comp[:].AF "Acentric Factor";
32      Real omega_l "Acentric factor in liquid phase mixture";
33      Real omega_v "Acentric factor in vapour phase mixture";
34      Real V_cm_l "pseudo-critical volume in liquid phase mixture";
35      Real V_cm_v "pseudo-critical volume in vapour phase mixture";
36      Real T_cm_l "pseudo-critical Temperature in liquid phase mixture";
37      Real T_cm_v "pseudo-critical Temperature in vapour phase mixture";
38      Real P_cm_l "pseudo-critical pressure in liquid phase mixture";
39      Real P_cm_v "pseudo-critical pressure in vapour phase mixture";
40      Real Z_cm_l "pseudo critical compressibility factor in liquid phase mixture";
41      Real Z_cm_v "pseudo critical compressibility factor in vapour phase mixture";
42      Real T_r_l "Reduced Temperature in liquid phase mixture";
43      Real T_r_v "Reduced Temperature in vapour phase mixture";
44      Real P_r_l "Reduced Pressure in liquid phase mixture";
45      Real P_r_v "Reduced Pressure in vapour phase mixture";
46      Real B_l[2] "abbreviations in BWR-Lee-Kesler equation for functions depending
            on temperature only";
47      Real C_l[2] "abbreviations in BWR-Lee-Kesler equation for functions depending
            on temperature only";
48      Real D_l[2] "abbreviations in BWR-Lee-Kesler equation for functions depending
            on temperature only";
49      Real B_v[2] "abbreviations in BWR-Lee-Kesler equation for functions depending
            on temperature only";
50      Real C_v[2] "abbreviations in BWR-Lee-Kesler equation for functions depending
            on temperature only";
51      Real D_v[2] "abbreviations  in BWR-Lee-Kesler equation for functions depending
             on temperature only";
52      Real V_r_l[2](each start = 0.1) "Reduced volume in liquid phase mixture";
53      Real V_r_v[2](each start = 1000) "Reduced volume in vapour phase mixture";
54      Real z_l[2] "compressibility factor in liquid phase(for simple and reference
            component)";
```

```
55      Real z_v[2] "compressibility factor in vapour phase(for simple and reference
            component)";
56      Real Z_l "compressibility factor in liquid phase";
57      Real Z_v "compressibility factor in vapour phase";
58      Real Tcij[Nc, Nc];
59      Real Vcij[Nc, Nc];
60      parameter Real n = 0.25 "universal exponent in mixing rule for the pseudo
            critical temperature";
61      Real phi_l[2] "fugacity coefficient for liquid phase(for simple and reference
            fluid)";
62      Real phi_v[2] "fugacity coefficient for vapour phase(for simple and reference
            fluid)";
63      Real E_l[2];
64      Real E_v[2];
65      Real phi_lm "fugacity coefficient for liquid phase mixture";
66      Real phi_vm "fugacity coefficient for vapour phase mixture";
67      Real del_h_l[2] "isothermal enthalpy departure for liquid phase(for simple and
            reference fluid)";
68      Real del_h_v[2] "isothermal enthalpy departure for vapour phase(for simple and
            reference fluid)";
69      Real del_h_lm "isothermal enthalpy departure for liquid phase mixture";
70      Real del_h_vm "isothermal enthalpy departure for vapour phase mixture";
71      Real sum1[Nc];
72      Real sum2[Nc];
73      Real sum3[Nc];
74      Real dzcm_l[Nc, Nc] "derivatives of the pseudo critical compressibility factor
            in liquid phase";
75      Real dvcm_l[Nc, Nc] "derivatives of the pseudo critical volume in liquid phase
            ";
76      Real dtcm_l[Nc, Nc] "derivatives of the pseudo critical temperature in liquid
            phase";
77      Real dpcm_l[Nc, Nc] "derivatives of the pseudo criticals pressure in liquid
            phase";
78      Real dphim_l "derivatives of the fugacity coefficient for liquid phase";
79      Real dphim_v "derivatives of the fugacity coefficient for vapour phase";
80      Real phi_liq[Nc] "fugacity coefficient in liquid phase";
81      Real sum4[Nc];
82      Real sum5[Nc];
83      Real sum6[Nc];
84      Real dzcm_v[Nc, Nc] "derivatives of the pseudo critical compressibility factor
            in vapour phase";
85      Real dvcm_v[Nc, Nc] "derivatives of the pseudo critical volume in vapour phase
            ";
86      Real dtcm_v[Nc, Nc] "derivatives of the pseudo critical temperature in vapour
            phase";
87      Real dpcm_v[Nc, Nc] "derivatives of the pseudo criticals pressure in vapour
            phase";
88      Real phi_vap[Nc] "fugacity coefficient in vapour phase";
89      Real K_c[Nc] "Equilibrium constant";
90      Real gma_c[Nc], gmabubl_c[Nc], gmadew_c[Nc];
91      Real philiqbubl_c[Nc], phivapdew_c[Nc], Pvap_c[Nc];
92    equation
93  //Mixing Rules
94      omega_l = sum({x_pc[2, i] * omega[i] for i in 1:Nc});
95      omega_v = sum({x_pc[3, i] * omega[i] for i in 1:Nc});
96      Tcij = {{(T_c[i] .* T_c[j]) ^ 0.5 .* kij[i, j] for i in 1:Nc} for j in 1:Nc};
97      Vcij = {{0.125 * (V_c[i] ^ (1 / 3) + V_c[j] ^ (1 / 3)) ^ 3 / 1000 for i in 1:
            Nc} for j in 1:Nc};
98      V_cm_l = sum({{x_pc[2, i] * x_pc[2, j] * Vcij[i, j] for i in 1:Nc} for j in 1:
            Nc});
99      V_cm_v = sum({{x_pc[3, i] * x_pc[3, j] * Vcij[i, j] for i in 1:Nc} for j in 1:
            Nc});
```

```
100     T_cm_l = 1 / V_cm_l ^ n * sum({{x_pc[2, i] * x_pc[2, j] * Vcij[i, j] ^ n *
            Tcij[i, j] for i in 1:Nc} for j in 1:Nc});
101     T_cm_v = 1 / V_cm_v ^ n * sum({{x_pc[3, i] * x_pc[3, j] * Vcij[i, j] ^ n *
            Tcij[i, j] for i in 1:Nc} for j in 1:Nc});
102     P_cm_l = (0.2905 - 0.085 * omega_l) * R * T_cm_l / V_cm_l;
103     P_cm_v = (0.2905 - 0.085 * omega_v) * R * T_cm_v / V_cm_v;
104     Z_cm_l = P_cm_l * V_cm_l / (R * T_cm_l);
105     Z_cm_v = P_cm_v * V_cm_v / (R * T_cm_v);
106     T_r_l = T / T_cm_l;
107     T_r_v = T / T_cm_v;
108     P_r_l = P / P_cm_l;
109     P_r_v = P / P_cm_v;
110     for i in 1:2 loop
111       B_l[i] = b1[i] - b2[i] / T_r_l - b3[i] / T_r_l ^ 2 - b4[i] / T_r_l ^ 3;
112       C_l[i] = c1[i] - c2[i] / T_r_l + c3[i] / T_r_l ^ 3;
113       D_l[i] = d1[i] + d2[i] / T_r_l;
114       B_v[i] = b1[i] - b2[i] / T_r_v - b3[i] / T_r_v ^ 2 - b4[i] / T_r_v ^ 3;
115       C_v[i] = c1[i] - c2[i] / T_r_v + c3[i] / T_r_v ^ 3;
116       D_v[i] = d1[i] + d2[i] / T_r_v;
117       P_r_l * V_r_l[i] / T_r_l = 1 + B_l[i] / V_r_l[i] + C_l[i] / V_r_l[i] ^ 2 +
              D_l[i] / V_r_l[i] ^ 5 + c4[i] / (T_r_l ^ 3 * V_r_l[i] ^ 2) * (beta[i] +
              gamma[i] / V_r_l[i] ^ 2) * exp(-gamma[i] / V_r_l[i] ^ 2);
118       P_r_v * V_r_v[i] / T_r_v = 1 + B_v[i] / V_r_v[i] + C_v[i] / V_r_v[i] ^ 2 +
              D_v[i] / V_r_v[i] ^ 5 + c4[i] / (T_r_v ^ 3 * V_r_v[i] ^ 2) * (beta[i] +
              gamma[i] / V_r_v[i] ^ 2) * exp(-gamma[i] / V_r_v[i] ^ 2);
119       z_v[i] = P_r_v * V_r_v[i] / T_r_v;
120     end for;
121     z_l[1] = -1 * P_r_l * V_r_l[1] / T_r_l;
122     z_l[2] = -1 * P_r_l * V_r_l[2] / T_r_l;
123     Z_l = z_l[1] + omega_l / omega_r * (z_l[2] - z_l[1]);
124     Z_v = z_v[1] + omega_v / omega_r * (z_v[2] - z_v[1]);
125 //fugacity coefficient calculation
126     for i in 1:2 loop
127       E_l[i] = c4[i] / (2 * T_r_l ^ 3 * gamma[i]) * (beta[i] + 1 - (beta[i] + 1 +
              gamma[i] / V_r_l[i] ^ 2) * exp(-gamma[i] / V_r_l[i] ^ 2));
128       E_v[i] = c4[i] / (2 * T_r_v ^ 3 * gamma[i]) * (beta[i] + 1 - (beta[i] + 1 +
              gamma[i] / V_r_v[i] ^ 2) * exp(-gamma[i] / V_r_v[i] ^ 2));
129       phi_l[i] = exp(z_l[i] - 1 - log(z_l[i]) + B_l[i] / V_r_l[i] + C_l[i] / (2 *
              V_r_l[i] ^ 2) + D_l[i] / (5 * V_r_l[i] ^ 5) + E_l[i]);
130       phi_v[i] = exp(z_v[i] - 1 - log(z_v[i]) + B_v[i] / V_r_v[i] + C_v[i] / (2 *
              V_r_v[i] ^ 2) + D_v[i] / (5 * V_r_v[i] ^ 5) + E_v[i]);
131     end for;
132     phi_lm = exp(log(phi_l[1]) + omega_l / omega_r * (log(phi_l[2]) - log(phi_l
            [1])));
133     phi_vm = exp(log(phi_v[1]) + omega_v / omega_r * (log(phi_v[2]) - log(phi_v
            [1])));
134 //enthalpy deviation
135     for i in 1:2 loop
136       del_h_l[i] = T_r_l * (z_l[i] - 1 - (b2[i] + 2 * b3[i] / T_r_l + 3 * b4[i] /
              T_r_l ^ 2) / (T_r_l * V_r_l[i]) - (c2[i] - 3 * c3[i] / T_r_l ^ 2) / (2 *
              T_r_l * V_r_l[i] ^ 2) + d2[i] / (5 * T_r_l * V_r_l[i] ^ 5) + 3 * E_l[i]);
137       del_h_v[i] = T_r_v * (z_v[i] - 1 - (b2[i] + 2 * b3[i] / T_r_v + 3 * b4[i] /
              T_r_v ^ 2) / (T_r_v * V_r_v[i]) - (c2[i] - 3 * c3[i] / T_r_v ^ 2) / (2 *
              T_r_v * V_r_v[i] ^ 2) + d2[i] / (5 * T_r_v * V_r_v[i] ^ 5) + 3 * E_v[i]);
138     end for;
139     del_h_lm = del_h_l[1] + omega_l / omega_r * (del_h_l[2] - del_h_l[1]);
140     del_h_vm = del_h_v[1] + omega_v / omega_r * (del_h_v[2] - del_h_v[1]);
141     Cpres_p[:] = zeros(3);
142     Hres_p[:] = zeros(3);
143     Sres_p[:] = zeros(3);
144     for i in 1:Nc loop
145       gma_c[i] = 1;
```

```
146         gmabubl_c[i] = 1;
147         gmadew_c[i] = 1;
148         philiqbubl_c[i] = 1;
149         phivapdew_c[i] = 1;
150       end for;
151       for i in 1:Nc loop
152         Pvap_c[i] = Simulator.Files.ThermodynamicFunctions.Psat(C[i].VP, T);
153       end for;
154       dphim_l = 1 / omega_r * (log(phi_l[2]) - log(phi_l[1]));
155       dphim_v = 1 / omega_r * (log(phi_v[2]) - log(phi_v[1]));
156     algorithm
157       for i in 1:Nc loop
158         for l in 1:Nc loop
159           dzcm_l[i, :] := -0.085 .* (omega[:] .- omega[i]);
160           dvcm_l[i, l] := 2 * sum(x_pc[2, :] .* (Vcij[:, l] .- Vcij[:, i]));
161           dtcm_l[i, l] := (2 * sum(x_pc[2, :] .* (Vcij[:, l] .^ n .* Tcij[:, l] -
                Vcij[:, i] .^ n .* Tcij[:, i])) .- n * V_cm_l ^ (n - 1) .* dvcm_l[i, l]
                .* T_cm_l) ./ V_cm_l ^ n;
162           dpcm_l[i, l] := P_cm_l * (dzcm_l[i, l] / Z_cm_l .+ dtcm_l[i, l] / T_cm_l
                .- dvcm_l[i, l] / V_cm_l);
163           dzcm_v[i, l] := -0.085 .* (omega[l] .- omega[i]);
164           dvcm_v[i, l] := 2. * sum(x_pc[3, :] .* (Vcij[:, l] - Vcij[:, i]));
165           dtcm_v[i, l] := (2. * sum(x_pc[3, :] .* (Vcij[:, l] .^ n .* Tcij[:, l] -
                Vcij[:, i] .^ n .* Tcij[:, i])) .- n * V_cm_v ^ (n - 1) .* dvcm_v[i, l]
                .* T_cm_v) ./ V_cm_v ^ n;
166           dpcm_v[i, l] := P_cm_v * (dzcm_v[i, l] / Z_cm_v .+ dtcm_v[i, l] / T_cm_v
                .- dvcm_v[i, l] / V_cm_v);
167         end for;
168       end for;
169       for i in 1:Nc loop
170         for l in 1:Nc loop
171           if l <> i then
172             sum1[i] := x_pc[2, l] * dtcm_l[i, l];
173             sum2[i] := x_pc[2, l] * dpcm_l[i, l];
174             sum3[i] := x_pc[2, l] * (omega[l] - omega[i]);
175             sum4[i] := x_pc[3, l] * dtcm_v[i, l];
176             sum5[i] := x_pc[3, l] * dpcm_v[i, l];
177             sum6[i] := x_pc[3, l] * (omega[l] - omega[i]);
178           end if;
179         end for;
180       end for;
181     equation
182       for i in 1:Nc loop
183         phi_liq[i] = exp(log(phi_lm) - del_h_lm / T * sum1[i] + (Z_cm_l - 1) /
              P_cm_l * sum2[i] - dphim_l * sum3[i]);
184         phi_vap[i] = exp(log(phi_vm) - del_h_vm / T * sum4[i] + (Z_cm_v - 1) /
              P_cm_v * sum5[i] - dphim_v * sum6[i]);
185       end for;
186       for i in 1:Nc loop
187         if phi_liq[i] == 0 or phi_vap[i] == 0 then
188           K_c[i] = 0;
189         else
190           K_c[i] = phi_liq[i] / phi_vap[i];
191         end if;
192       end for;
193     end LKPTest;
194
195     model Test
196       extends Modelica.Icons.Example;
197       import data = Simulator.Files.ChemsepDatabase;
198       parameter Integer Nc = 2;
199       parameter data.Nbutane but;
```

```
    parameter data.Nhexane hex;
    parameter data.GeneralProperties C[Nc] = {but, hex};
    LKP.ms S1(C = C, Nc = Nc) annotation(
      Placement(visible = true, transformation(origin = {-128, 14}, extent =
          {{-10, -10}, {10, 10}}, rotation = 0)));
  equation
    S1.F_p[1] = 100;
    S1.x_pc[1, :] = {0.5, 0.5};
    S1.P = 101325;
    S1.T = 300;
  end Test;
end LKP;
```

```
model LKPTest
 import Simulator.*;
import data = Simulator.Files.ChemsepDatabase;
parameter data.Acetone but;
parameter data.Aceticacid hex;
parameter Integer Nc=2;

parameter data.GeneralProperties comp[Nc]={but,hex};
//constants for simple fluid
parameter Real b1[2]={0.1181193,0.2026579}"Constants for the BWR-Lee_Kesler
    Equation";
parameter Real b2[2]={0.265728,0.331511}"Constants for the BWR-Lee_Kesler Equation
    ";
parameter Real b3[2]={0.154790,0.027655}"Constants for the BWR-Lee_Kesler Equation
    ";
parameter Real b4[2]={0.030323,0.203488}"Constants for the BWR-Lee_Kesler Equation
    ";
parameter Real c1[2]={0.0236744,0.0313385}"Constants for the BWR-Lee_Kesler
    Equation";
parameter Real c2[2]={0.0186984,0.0503618}"Constants for the BWR-Lee_Kesler
    Equation";
parameter Real c3[2]={0,0.016901}"Constants for the BWR-Lee_Kesler Equation";
parameter Real c4[2]={0.042724,0.041577}"Constants for the BWR-Lee_Kesler Equation
    ";
parameter Real d1[2]={0.0000155488,0.000048736}"Constants for the BWR-Lee_Kesler
    Equation";
parameter Real d2[2]={0.0000623689,0.00000740336}"Constants for the BWR-Lee_Kesler
     Equation";
parameter Real beta[2]={0.65392,1.226}"Constants for the BWR-Lee_Kesler Equation";
parameter Real gamma[2]={0.060167,0.03754}"Constants for the BWR-Lee_Kesler
    Equation";
parameter Real omega_r=0.397"acentric factor of N-octane";
parameter Real R=8.314472" Universal gas constant";
parameter Real T(unit = "K") = 300"input Temperature";
parameter Real P(unit = "Pa") = 101325" input Pressure";
parameter Real z[Nc](each min=0, each max=1) = {0.5, 0.5}"feed mole fraction";
parameter Real kij[Nc,2]={{1,1},{1,1}}" binary interaction parameter for the i-j
    mixture rule for the pseudo critical temperature";
Real T_c[Nc]=comp[:].Tc"Critical Temperature";
Real P_c[Nc]=comp[:].Pc"critical Pressure";
Real V_c[Nc]=comp[:].Vc"Critical Volume";
Real omega[Nc]=comp[:].AF"Acentric Factor";
Real omega_l"Acentric factor in liquid phase mixture";
Real omega_v"Acentric factor in vapour phase mixture";
Real x[Nc] (each min=0,each max=1 )={0.5,0.5}"Mole fraction in liquid phase";//
    ={0.3590478,0.6409522};
Real y[Nc](each min=0,each max=1 )={0.949496,0.050504}"Mole fraction in vapour
    phase";//={0.85280566,0.14719434};
```

```
36
37  Real V_cm_l"pseudo-critical volume in liquid phase mixture";
38  Real V_cm_v"pseudo-critical volume in vapour phase mixture";
39  Real T_cm_l"pseudo-critical Temperature in liquid phase mixture";
40  Real T_cm_v"pseudo-critical Temperature in vapour phase mixture";
41  Real P_cm_l"pseudo-critical pressure in liquid phase mixture";
42  Real P_cm_v"pseudo-critical pressure in vapour phase mixture";
43  Real Z_cm_l"pseudo critical compressibility factor in liquid phase mixture";
44  Real Z_cm_v"pseudo critical compressibility factor in vapour phase mixture";
45  Real T_r_l"Reduced Temperature in liquid phase mixture";
46  Real T_r_v"Reduced Temperature in vapour phase mixture";
47  Real P_r_l"Reduced Pressure in liquid phase mixture";
48  Real P_r_v"Reduced Pressure in vapour phase mixture";
49  Real B_l[2]"abbreviations in BWR-Lee-Kesler equation for functions depending on
        temperature only";
50  Real C_l[2]"abbreviations in BWR-Lee-Kesler equation for functions depending on
        temperature only";
51  Real D_l[2]"abbreviations in BWR-Lee-Kesler equation for functions depending on
        temperature only";
52  Real B_v[2]"abbreviations in BWR-Lee-Kesler equation for functions depending on
        temperature only";
53  Real C_v[2]"abbreviations in BWR-Lee-Kesler equation for functions depending on
        temperature only";
54  Real D_v[2]"abbreviations in BWR-Lee-Kesler equation for functions depending on
        temperature only";
55  Real V_r_l[2](each start=0.1)"Reduced volume in liquid phase mixture";
56  Real V_r_v[2](each start=1000)"Reduced volume in vapour phase mixture";
57  Real z_l[2]"compressibility factor in liquid phase(for simple and reference
        component)";
58  Real z_v[2]"compressibility factor in vapour phase(for simple and reference
        component)";
59  Real Z_l"compressibility factor in liquid phase";
60  Real Z_v"compressibility factor in vapour phase";
61  Real Tcij[Nc,Nc];
62  Real Vcij[Nc,Nc];
63  parameter  Real n=0.25"universal exponent in mixing rule for the pseudo critical
        temperature";
64  Real phi_l[2]"fugacity coefficient for liquid phase(for simple and reference fluid
        )";
65  Real phi_v[2]"fugacity coefficient for vapour phase(for simple and reference fluid
        )";
66  Real E_l[2];
67  Real E_v[2];
68  Real phi_lm"fugacity coefficient for liquid phase mixture";
69  Real phi_vm"fugacity coefficient for vapour phase mixture";
70
71  Real del_h_l[2]"isothermal enthalpy departure for liquid phase(for simple and
        reference fluid)";
72  Real del_h_v[2]"isothermal enthalpy departure for vapour phase(for simple and
        reference fluid)";
73  Real del_h_lm"isothermal enthalpy departure for liquid phase mixture";
74  Real del_h_vm"isothermal enthalpy departure for vapour phase mixture";
75
76  equation
77  //Mixing Rules
78  omega_l=sum({(x[i]*omega[i]) for i in 1:Nc});
79  omega_v=sum({(y[i]*omega[i]) for i in 1:Nc});
80
81  Tcij={{((T_c[i].*T_c[j])^0.5) .*kij[i,j] for i in 1:Nc}for j in 1:Nc} ;
82
83
84  Vcij={{(0.125*(V_c[i]^(1/3) + V_c[j]^(1/3))^3)/1000 for i in 1:Nc} for j in  1:Nc
```

```
    };

V_cm_l=sum({{(x[i]*x[j]*Vcij[i,j]) for i in 1:Nc} for j in 1:Nc});
V_cm_v=sum({{(y[i]*y[j]*Vcij[i,j]) for i in 1:Nc} for j in 1:Nc});

T_cm_l=(1/(V_cm_l^n))*sum({{(x[i]*x[j]*Vcij[i,j]^n *Tcij[i,j]) for i in 1:Nc} for
    j in 1:Nc});
T_cm_v=(1/(V_cm_v^n))*sum({{(y[i]*y[j]*Vcij[i,j]^n *Tcij[i,j]) for i in 1:Nc} for
    j in 1:Nc});

P_cm_l=((0.2905-(0.085*omega_l))*R*T_cm_l)/V_cm_l;
P_cm_v=((0.2905-(0.085*omega_v))*R*T_cm_v)/V_cm_v;

Z_cm_l=(P_cm_l*V_cm_l)/(R*T_cm_l);
Z_cm_v=(P_cm_v*V_cm_v)/(R*T_cm_v);

T_r_l=T/T_cm_l;
T_r_v=T/T_cm_v;

P_r_l=P/P_cm_l;
P_r_v=P/P_cm_v;


for i in 1:2 loop
B_l[i]=b1[i]-(b2[i]/T_r_l)-(b3[i]/(T_r_l^2))-(b4[i]/(T_r_l^3));
C_l[i]=c1[i]-(c2[i]/T_r_l)+(c3[i]/(T_r_l^3));
D_l[i]=d1[i]+(d2[i]/T_r_l);
B_v[i]=b1[i]-(b2[i]/T_r_v)-(b3[i]/(T_r_v^2))-(b4[i]/(T_r_v^3));
C_v[i]=c1[i]-(c2[i]/T_r_v)+(c3[i]/(T_r_v^3));
D_v[i]=d1[i]+(d2[i]/T_r_v);

(P_r_l*V_r_l[i])/T_r_l = 1+ (B_l[i]/V_r_l[i])+(C_l[i]/(V_r_l[i]^2))+(D_l[i]/(V_r_l
    [i]^5))+((c4[i]/((T_r_l^3)*(V_r_l[i]^2)))*(beta[i]+(gamma[i]/V_r_l[i]^2))*exp(-
    gamma[i]/V_r_l[i]^2));
(P_r_v*V_r_v[i])/T_r_v = 1+ (B_v[i]/V_r_v[i])+(C_v[i]/(V_r_v[i]^2))+(D_v[i]/(V_r_v
    [i]^5))+((c4[i]/((T_r_v^3)*(V_r_v[i]^2)))*(beta[i]+(gamma[i]/V_r_v[i]^2))*exp(-
    gamma[i]/V_r_v[i]^2));

z_l[i]=(P_r_l*V_r_l[i])/(T_r_l);
z_v[i]=(P_r_v*V_r_v[i])/(T_r_v);
end for;
Z_l=z_l[1] + ((omega_l/omega_r)*(z_l[2]-z_l[1]));
Z_v=z_v[1] + ((omega_v/omega_r)*(z_v[2]-z_v[1]));

//fugacity coefficient calculation
for i in 1:2 loop
E_l[i]= (c4[i]/(2*(T_r_l^3) *gamma[i]))*(beta[i]+1-((beta[i]+1+(gamma[i]/(V_r_l[i
    ]^2)))*exp(-gamma[i]/(V_r_l[i]^2))));
E_v[i]= (c4[i]/(2*(T_r_v^3) *gamma[i]))*(beta[i]+1-((beta[i]+1+(gamma[i]/(V_r_v[i
    ]^2)))*exp(-gamma[i]/(V_r_v[i]^2))));
phi_l[i]=exp(z_l[i]-1-(log(z_l[i]))+(B_l[i]/V_r_l[i])+(C_l[i]/(2*V_r_l[i]^2))+(D_l
    [i]/(5*V_r_l[i]^5))+E_l[i]);
phi_v[i]=exp(z_v[i]-1-(log(z_v[i]))+(B_v[i]/V_r_v[i])+(C_v[i]/(2*V_r_v[i]^2))+(D_v
    [i]/(5*V_r_v[i]^5))+E_v[i]);
end for;
phi_lm=exp(log(phi_l[1])+((omega_l/omega_r)*(log(phi_l[2])-log(phi_l[1]))));
phi_vm=exp(log(phi_v[1])+((omega_v/omega_r)*(log(phi_v[2])-log(phi_v[1]))));

//enthalpy deviation
for i in 1:2 loop
del_h_l[i]= T_r_l*(z_l[i]-1-((b2[i]+((2*b3[i])/T_r_l) + (3*b4[i]/T_r_l^2))/(T_r_l*
    V_r_l[i]))- ((c2[i]-((3*c3[i])/T_r_l^2))/(2*T_r_l*(V_r_l[i]^2)))+(d2[i]/(5*
```

```
    T_r_l*(V_r_l[i]^5)))+(3*E_l[i]));
135 del_h_v[i]= T_r_v*(z_v[i]-1-((b2[i]+((2*b3[i])/T_r_v) + (3*b4[i]/T_r_v^2))/(T_r_v*
    V_r_v[i]))- ((c2[i]-((3*c3[i])/T_r_v^2))/(2*T_r_v*(V_r_v[i]^2)))+(d2[i]/(5*
    T_r_v*(V_r_v[i]^5)))+(3*E_v[i]));
136 end for;
137 del_h_lm=del_h_l[1]+((omega_l/omega_r)*(del_h_l[2]-del_h_l[1]));
138 del_h_vm=del_h_v[1]+((omega_v/omega_r)*(del_h_v[2]-del_h_v[1]));
139 end LKPTest;
```