



Table of Contents

Section 3.1 - Volume Solvers

Cubic EoS

Statistical Associating Fluid Theory (SAFT)

Initial guesses

Implementation

Footnotes

Section 3.1 - Volume Solvers

Now we know how to choose an equation of state for different situations, we need to investigate how to obtain the fluid properties at a specified state. To start with, we will look at how to solve for the volume at a given pressure, temperature, and phase for a pure fluid. From there, we will see how this changes when the phase is not known beforehand, and when dealing with a multicomponent mixture.

Usually, we specify pressure and temperature (pT), which when solving for the volume corresponds to solving the equation

$$p(v, T_0) = p_0$$

for all values of v . These are considered the **volume roots** for our equation of state.

Cubic EoS

As you have seen in section 2, the general cubic equation of state can be written as

$$p = \frac{RT}{v - b} - \frac{a\alpha(T)}{(v + \delta_1 b)(v + \delta_2 b)}$$

where the values of δ_1, δ_2 are specific to each equation of state (van der Waals, Peng Robinson, Redlich Kwong), and the form of the **alpha function**, $\alpha(T)$, can be selected to match specific types of components or problems. In this workbook we will consider the van der Waals EoS, which is the case when

$$\delta_1 = \delta_2 = 0$$

$$\alpha(T) = 1$$

Giving us

$$p = \frac{RT}{v - b} - \frac{a}{v^2}$$

where

$$a = \frac{27}{64} \frac{(RT_c)^2}{p_c}$$
$$b = \frac{1}{8} \frac{RT_c}{p_c}$$

To solve this for the volume, we can leverage the fact you can rearrange it as a cubic equation in terms of the **compressibility factor**, Z .

To do this, we multiply the expression through by the denominators and substitute in the definition of Z .

$$Z = \frac{Pv}{RT}$$
$$pv^2(v - b) = RTv^2 - a(v - b)$$
$$pv^3 - (bp + RT)v^2 + av - ab = 0$$
$$Z^3 - \left(1 + \frac{bp}{RT}\right)Z^2 + \left(\frac{ap}{(RT)^2}\right)Z - \frac{abp^2}{(RT)^3} = 0$$

To express this cubic neatly we define 2 constants, A and B .

$$A = a \cdot \frac{p}{(RT)^2}$$
$$B = b \cdot \frac{p}{RT}$$

The cubic we then need to solve is

$$Z^3 - (1 + B)Z^2 + AZ - AB = 0$$

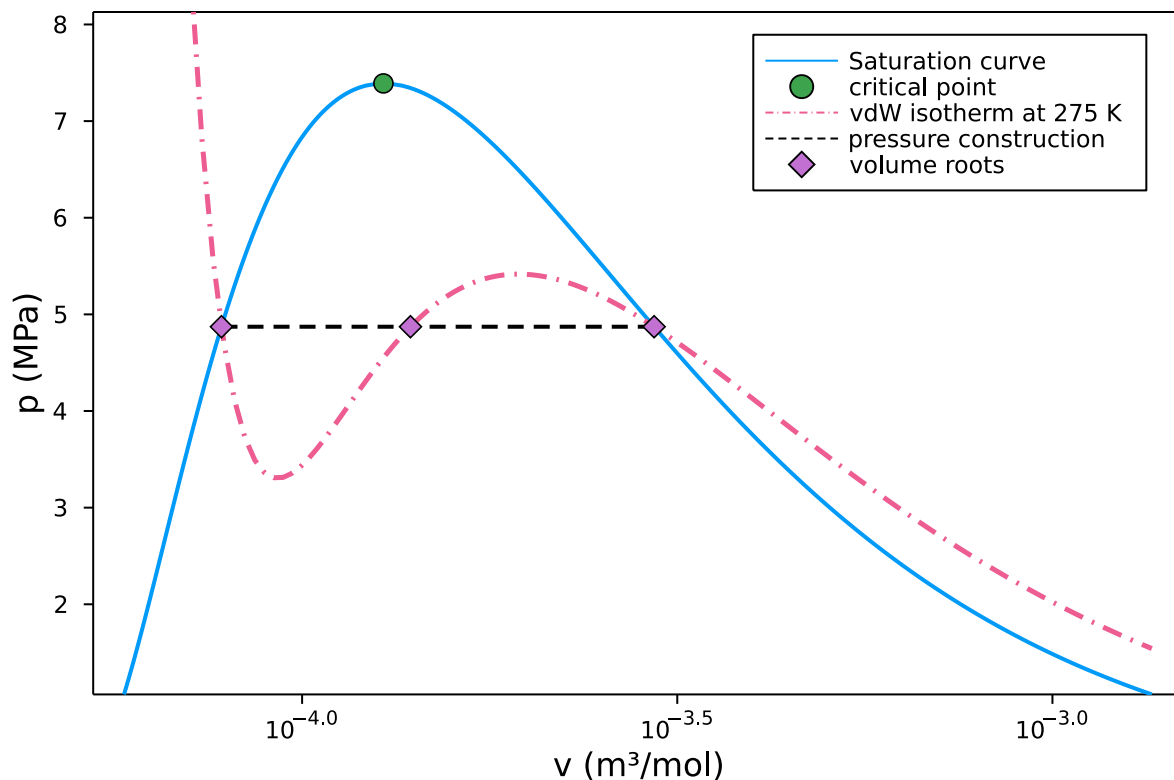
A similar result can be obtained for any cubic equation.

The three roots for the van der Waals equation of state can be seen below:

T =

 275 K

Cell deleted (UNDO)



Under saturated conditions, the middle root never has physical meaning. Note also the line marked **pressure construction**. For a pure saturated fluid, an isotherm has **constant temperature**. This is not typically captured by an equation of state, so when designing software, care should be taken that calculations of pressure are physical and correct.

Let's now solve the van der Waals equation for the volume roots and see if our answers make sense.

cubic_volume

```
cubic_volume(model::ABCubicModel, p, T)
```

Solves a cubic equation of state for all volume roots, real or complex. Returns a vector of all three roots.

```
• """
•     cubic_volume(model::ABCubicModel, p, T)
•
•     Solves a cubic equation of state for all volume roots, real or complex. Returns a
•     vector of all three roots.
•     """
•     function cubic_volume(model::ABCubicModel, p, T)
•         Tc, pc, _ = crit_pure(model)
•
•         a = 27/64 * (R*Tc)^2/pc # a parameter
•         b = 1/8 * (R*Tc)/pc # b parameter
•
•         # Rearranged cubic parameters
•         A = a*p/(R*T)^2
•         B = b*p/(R*T)
•
•         poly = [-A*B, A, -(1+B), 1.0] # Polynomial coefficients
•         Zvec = roots(poly) # Solve polynomial for Z
•         Vvec = Zvec.*(R*T/p) # Transform to volume
•         return Vvec
•     end
```

```
► [0.00026787-0.0im, 0.000152765+0.0im, 7.6368e-5+0.0im]
```

```
• begin
•     cubic_model = vdW(["carbon dioxide"])
•     p = 50e5
•     T = 273.15
•     Vvec = cubic_volume(cubic_model, p, T)
• end
```

| Root | V (m ³) |
|------|---------------------|
| 1 | 0.000268 |
| 2 | 0.000153 |
| 3 | 7.64e-5 |

Got it!

Good job!

Cell deleted ([UNDO](#))

Now we have that working, we can see we have 3 real roots. How do we know which one to choose? We know that the smallest root is *liquidlike*, the largest root is *vapourlike*, and that the middle root has no physical meaning. From our physical knowledge of hydrogen sulfide, we can tell that it should be a gas and so we should take the vapourlike root, but this isn't something we can apply rigorously or put into our code.

In many situations the cubic equation will have imaginary roots, which are always unphysical and should be discarded. However, most of the time a decision between real roots has to be made. To do this, we will introduce the **gibbs free energy**, as we know that the phase with the lowest gibbs free energy (or chemical potential) will be the stable phase at the given conditions.

To evaluate the chemical potential, we can use

```
VT_chemical_potential(model, V, T)
```

```
► [-2331.5, -2307.9, -2387.4]
```

```
• begin
•   μ(V) = VT_chemical_potential(cubic_model, V, T)
•
•   Vvec_real = real.(Vvec[abs.(imag.(Vvec)) .< eps()]) # Filter out volume roots
•               with imaginary components below machine precision
•   μvec = μ.(Vvec_real)
•   μ_show = [round(i[1], sigdigits=5) for i in μvec]
• end
```

| Root | μ | V |
|------|---------|----------|
| 1 | -2331.5 | 0.000268 |
| 2 | -2307.9 | 0.000153 |
| 3 | -2387.4 | 7.64e-5 |

We can see that our lowest chemical potential is given by our third root, showing this is the most stable root and that it is the correct volume for the system.

We can compare our answer for V to the result calculated by Clapeyron, using

```
volume(model, p, T)
```

```
V_cubic = 7.636801080730772e-5
```

```
• V_cubic = Vvec_real[findmin(μvec)[2]]
```

```
true
```

```
• V_cubic ≈ volume(cubic_model, p, T)
```

Cell deleted (UNDO)

Got it!

You converged within machine precision!

and see that we have chosen the correct root!

Statistical Associating Fluid Theory (SAFT)

With equations of state based off of SAFT (e.g. ogSAFT, SAFTVRMie), we have an expression for the residual helmholtz free energy. This is expressed as a sum of different contributions,

$$a^{\text{res}} = a^{\text{seg}} + a^{\text{chain}} + a^{\text{assoc}} .$$

As this is already implemented in Clapeyron, all we need to know to use these equations of state is that we have an expression

$$a^{\text{res}} = f(V, T) .$$

One way to obtain an expression we could solve for the volume at a specified pressure and temperature would be to take the partial derivative of a to express this as a nonlinear equation

$$\left(\frac{\partial a^{\text{res}}}{\partial V} \right)_T = -p .$$

This can then be rearranged to

$$f(V, T, p) = \left(\frac{\partial a^{\text{res}}(V, T)}{\partial V} \right)_T + p = 0 \quad (1)$$

giving us a non-linear root-finding problem.

An alternative approach is to begin with the definition of isothermal compressibility

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

which can be integrated to

$$\beta \cdot (p_2 - p_1) = \ln(V_1) - \ln(V_2)$$

$$\exp(\beta \cdot (p_2 - p_1)) = \frac{V_1}{V_2}$$

$$V_1 = V_2 \exp(\beta \cdot (p_2 - p_1)) .$$

If we take p_2 as the specification pressure, and p_1 a function of V_2 , then we have obtained a formula we can use to iterate to convergence.

$$V_{i+1} = V_i \exp(\beta(V_i) \cdot (p^{\text{spec}} - p(V_i)))$$

Where every variable is evaluated at the specification temperature, T^{spec} .

To increase numerical stability this is then moved to log-space

$$\ln V_{i+1} = \ln V_i + \beta(V_i) \cdot (p^{\text{spec}} - p(V_i)) . \quad (2)$$

In practice, equation (2) converges faster than directly solving equation (1). [1]

We now have a relation that will converge to a volume root of our equation via **successive substitution**, but how do we go about generating initial guesses?

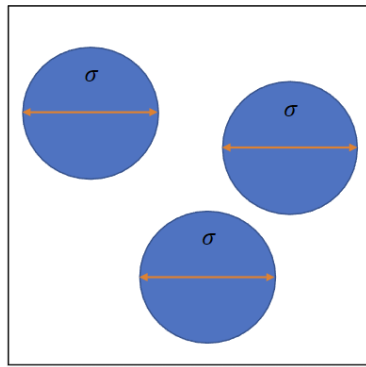
Initial guesses

To generate liquid-like initial guesses for SAFT equations, we're going to use a method based off of the packing fraction. This is defined as


$$\eta = \frac{\frac{3\pi}{3} \cdot \left(\frac{\sigma}{2}\right)^3 \cdot N_A}{V}$$

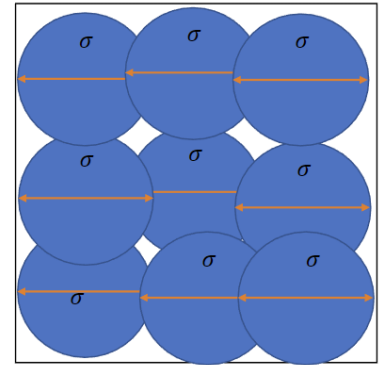
where σ is the segment size and N_A is Avogadro's number.

The packing fraction can visually be seen as the point at which all space in a given volume is taken up by fluid molecules:



$\eta \rightarrow 1$


An unphysical lower-bound



As we take the limit of the packing fraction to one,

$$1 = \frac{\frac{\pi}{6} \cdot N_A \cdot \sigma^3}{V}$$

Giving us the expression for our volume guess as

$$V_0^{\text{liq}} = \frac{\pi}{6} \cdot N_A \cdot \sigma^3 .$$

For the vapour-like initial guesses we can use the ideal gas equation

$$V^{\text{vap}} = \frac{nRT}{p} .$$

It would be possible to improve this by including further terms of the Virial equation, for example

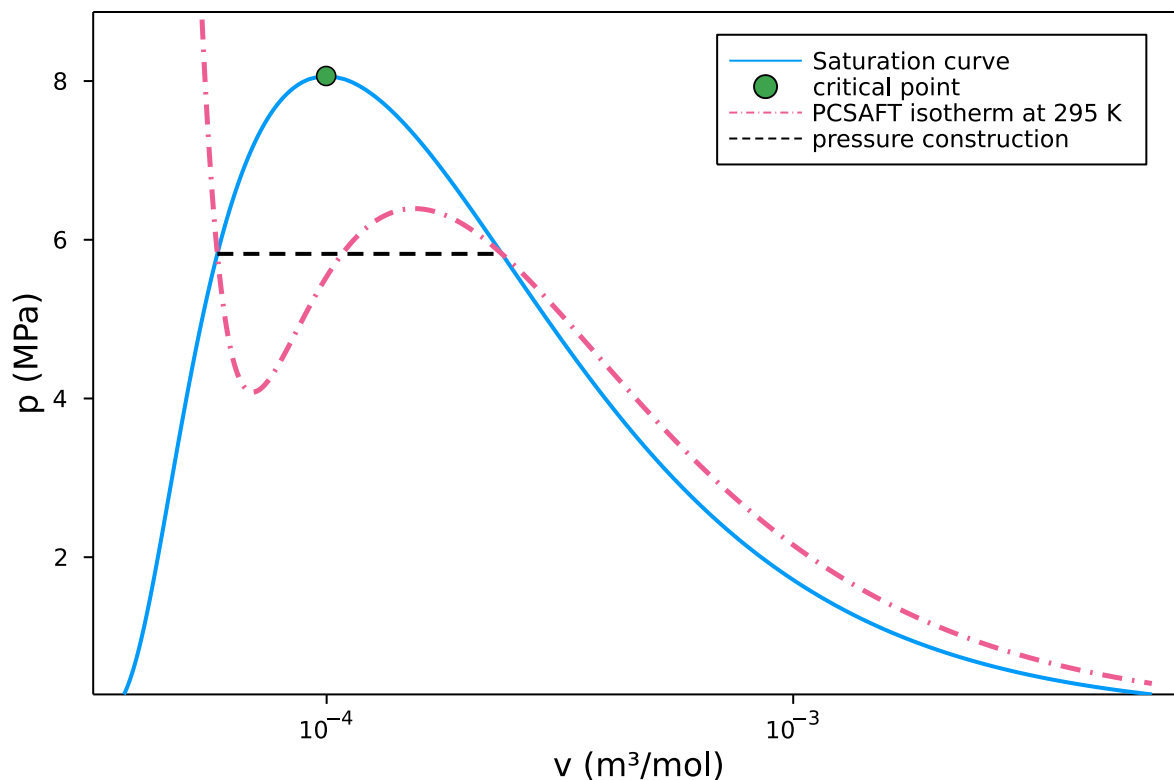
$$Z^{\text{vap}} = 1 + \frac{B(T)}{V}$$

but this isn't necessary for now, as the sequence defined by equation (2) generally converges very quickly.

T =



Cell deleted ([UNDO](#))



Implementation

To implement this, the functions we'll need are

```
 $\beta$  = VT_isothermal_compressibility(model, V, T)
p = pressure(model, V, T)
```

The `volume_guess` function for the liquid phase directly indexes the model object to obtain the sigma and segment values. Because only SAFT type models have these parameters, it is important we restrict our function to only accept the correct types. We do this with the `::SAFTModel` syntax in our function definition.

volume_guess

```
volume_guess(model, p, T, phase)
```

Generates initial guesses for the a volume solver. The liquid phase initial guess is based off of the packing fraction limit, and the vapour phase guess is based off of the ideal gas equation.

```
· """
·     volume_guess(model, p, T, phase)
·
· Generates initial guesses for the a volume solver. The liquid phase initial guess
· is based off of the packing fraction limit, and the vapour phase guess is based off
· of the ideal gas equation.
· """
· function volume_guess(model::SAFTModel, p, T, phase)
·     if phase == :liquid
·         # Extract parameters
·         σ = model.params.sigma.diagvalues[1]
·         seg = model.params.segment.values[1]
·         V0 = 1.25 * π/6 * N_A * σ^3 * seg
·     elseif phase == :vapour
·         V0 = R*T/p
·     else
·         @error "Invalid phase specification, $phase. Specify phase as either
·             ``:liquid`` or ``:vapour``"
·     end
·     return V0
· end
```

In this implementation of SAFT_volume, abstol and maxiters are **keyword arguments**. These are optional when calling the function, and have given default values. For example, you could either call

```
SAFT_volume(model, p, T, phase)
```

or for higher precision

```
SAFT_volume(model, p, T, phase; abstol=1e-10)
```

you can read more about keyword arguments [here](#).

SAFT_volume

```
SAFT_volume(model, p, T, phase; abstol=1e-9, maxiters=100)
```

Solves an equation of state for a volume root. The root converged to is chosen by the initial guess, which is specified by the phase argument. The phase can either be liquid, `:liquid`, or vapour, `:vapour`.

```
• """
•     SAFT_volume(model, p, T, phase; abstol=1e-9, maxiters=100)
•
•     Solves an equation of state for a volume root. The root converged to is chosen by
•     the initial guess, which is specified by the phase argument. The phase can either
•     be liquid, ``:liquid``, or vapour, ``:vapour``.
•     """
•     function SAFT_volume(model, p, T, phase; abstol=1e-9, maxiters=100)
•          $\beta(V) = \text{VT\_isothermal\_compressibility}(\text{model}, V, T)$ 
•          $p_1(V) = \text{pressure}(\text{model}, V, T)$ 
•
•         V = volume_guess(model, p, T, phase)
•         lnV = log(V)
•         Vtrack = [V]
•         norm = 1.0
•         iters = 0
•         while norm > abstol && iters < maxiters
•             d =  $\beta(V) * (p_1(V) - p)$  # Calculate the iterative step
•             lnV = lnV + d # Take the step
•
•             # Copy previous iteration and value
•             Vold = V
•             V = exp(lnV)
•             norm = abs(Vold - V)
•         end
•
•         if iters == maxiters
•             @warn "Volume iteration failed to converge in $maxiters iterations"
•         end
•         return V
•     end
```

Now our functions have been defined we can define our model, then call the function defined above to calculate the volume for a given phase.

```
4.752850833046656e-5
```

```
• begin
•     SAFT_model = PCSAFT(["carbon dioxide"])
•     V_SAFT = SAFT_volume(SAFT_model, p, T, :liquid)
• end
```

We can again compare our answer to Clapeyron

Cell deleted ([UNDO](#))

Got it!

Yay ♥

and we see that we've converged to the same answer!

Note that if we don't know *a-priori* which phase we should solve for, it's once again necessary to solve for both the liquid and vapour roots and evaluate the chemical potential to determine which is more stable.

Footnotes

[1]:

The recurrence relation formed by equation (2) turns out to be a Newton step towards the solution of equation (1).

Cell deleted ([UNDO](#))