MA348 Numerical Analysis, Thermodynamics

David Jefts

February 2, 2019

Introduction

The goal of this lab is to use various algorithms to estimate the root of a function. In this case, the function was van der Waal's equation of state, $(P + \frac{a}{v^2})(v - b) = RT$, where P is the Pressure in atmospheres, R is the Gas Constant for oxygen in atmospheres per mole-Kelvin, T is the temperature in Kelvin, v is the modal volume and $v = \frac{V}{n}$, V is the total volume, n is the number of moles of gas present, a is the measure of the average attraction between particles, and b is the volume excluded by a mole of particles. In many chemical engineering models, a very accurate modal volume of an atom or molecule, in this case oxygen, is required in order to properly construct containment apparatuses for these gases. van der Waal's equation of state is an expansion upon the classic Ideal Gas Law formula, $PV = nRT$. Using the relationship $v = \frac{V}{n}$, the Gas Law formula used for this lab is $Pv = RT$. In this lab, the values for R, a, and b are constant and known, T and P change between trials but are known, and v changes relative to the previous variables based on van der Waal's equation.

Theory-Analysis

The function for this lab is van der Waal's equation of state, $(P + \frac{a}{v^2})(v - b) = RT$, and the objective is to estimate roots for this function. v is the changing variable, essentially the 'x' value, so the function must be solved in terms of v:

$$f(v) = Pv^3 - (bP + RT)v^2 + av - ab = 0$$

This function serves as the main '$f(v)$' function for the remainder of this report. In addition, some of the algorithms used require the derivative and the second derivative of this function:

$$f'(v) = 3 \times Pv^2 - 2 \times (bP + RT)v + a = 0$$

$$f''(v) = 6 \times Pv - 2 \times (bP + RT) = 0$$

The Ideal Gas Law formula solved for v is:

$$v = \frac{RT}{P}$$

The only assumptions in this report are the oxygen Gas Constant values for R, a, and b. For this report, $R \approx 0.082054$, $a \approx 1.360$, and $b \approx 0.03183$. Additionally, the Fortran installation used to compute root values is only capable of representing 16 decimal places and is ineffective at representing extremely small numbers due to computer round-off and truncation errors.

Numerical Solution

This lab was solved using Fortran code to estimate the roots of the function and gnuplot to plot and tabulate the values. Multiple different methods were used, which as a group are colloquially called "open methods." This name comes from the fact that all of the methods work on an "open" set of numbers as opposed to the "bracketing" methods that work only on a set of numbers within a limited range.

Newton's Method (also known as the Newton-Raphson method is an approximation algorithm that uses the tangent line of a function to estimate the root. Starting with a given value, it iteratively uses the function $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ until a desired accuracy is reached. This method converges quadratically, making it a very desirable method for quickly calculating the root of a function and it is relatively easy to extend this method to high-order equations and functions. The drawbacks to this method are that the derivative of $f(x)$ has to be recalculated each iteration and the method will completely fail if $f'(x) = 0$ at any point during the iteration process.

The Secant Method is an approximation algorithm that uses the secant line of a function to estimate the root. Starting with a given value, it iteratively uses the function $x_{n+1} = x_n - \frac{f(x) \times (x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$ until a desired accuracy is reached. The Secant Method converges slightly slower than Newton's Method, on the order of $\approx$1.618, the Golden Ratio. This method can be implemented easier than the Newton Method since the derivative of $f(x)$ does not need to be calculated, however this method requires two initial points to start and it is not always guaranteed to converge.

The Modified Secant Method is another approximation algorithm that uses the secant line of a function to estimate the root. This method is a modification of The Secant Method detailed above, taking advantage of Newton's difference quotient function to estimate the derivative. It iteratively uses the function $x_{n+1} = x_n - \frac{f(x_n) \times \delta)}{f(x_n + \delta) - f(x_n)}$ until a desired accuracy is reached. The Modified Secant Method converges at the same rate as The Secant Method,

while removing the need to define and track two initial points. However it requires a definition of delta (δ) with constraints- if defined too large it may skip over the root, too small and it may never converge in a reasonable amount of time. For this purposes of this lab $\delta = 0.01$.

The Modified Newton's Method for Roots of Multiplicity is an approximation algorithm that uses the tangent line of a function and its second derivative to estimate the root. This method iteratively uses the function $x_{n+1} = x_n - \frac{f(x_n) \times f'(x_n)}{(f'(x_n))^2 - f(x) \times f''(x_n)}$ until a desired accuracy is reached. Like Newton's Method, this method converges quadratically. Additionally, this method works even if there is a Root of Multiplicity (i.e. the function has two roots of the same value such as $f(x) = x^2 - 9$. However calculating the second derivative of a function is not always easy.

For the comparison test using $P = 10$atm and $T = 100$K and starting numbers of $x_{n+1} = 3, x_n = 4$, and $x_{n-1} = 5$:

Newton's Method converged in 4 iterations
The Secant Method converged in 6 iterations
The Modified Secant Method converged in 5 iterations
The Modified Newton's Method for Roots of Multiplicity converged in 6 iterations

Results and Discussion

Firgure 1 in Appendix A is a graph of the error during each iteration of each algorithm, where each line represents one of the above-mentioned estimation methods, the x-axis is the number of iterations and the y-axis is the error. In this graph scenario $P = 10atm$ and $T = 300K$. This graph seems to indicate that Newton's Method both starts with a smaller error and converges towards the desired value faster. The other methods all converged upon the same value, but were slower, less efficient, and/or unnecessarily complicated (as in the case of the Modified Newton's Method for Roots of Multiplicity since this function does not have any roots with multiplicity). When compared with the Ideal Gas Law however, they all reached a Modal Volume about $0.2\frac{m^3}{mol}$. The cause of this discrepancy is most likely due to the corrections the van der Waal's equation makes to the Ideal Gas Law formula by observing and allowing for the fact that gas molecules attract each other

(noted by the variable a in his equation) and the volume of each individual mole of gas (noted by the variable b in his equation).

Conclusions

Newton's Method appears to be the most efficient method without sacrificing any error or simplicity, in the future that may not always be the case though, depending on the function, how it curves, and what its roots are. If this were to be done by hand the most accurate and efficient way would be Newton's Method, however the other methods do not converge at a significantly slower rate when using a tolerance of 10^{-6}. There are many other ways to optimize the estimation this function however, the methods of finding roots detailed in this report are just a handful from a plethora of various solvers and algorithms that accomplish the same task.

Appendix A

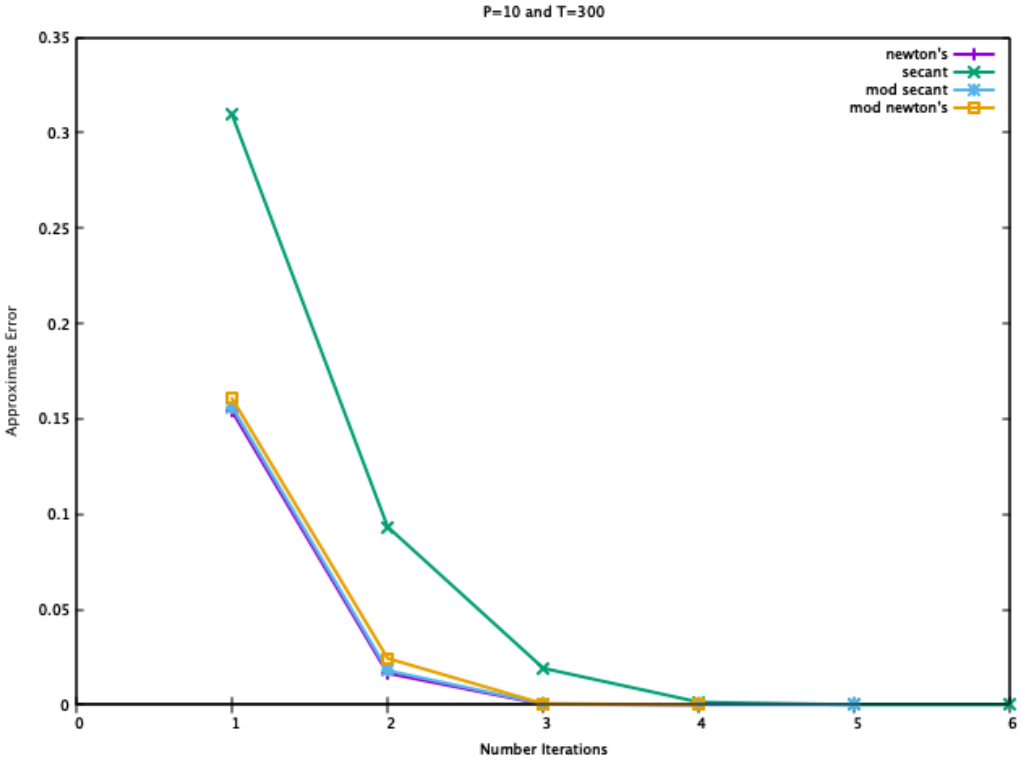

Figure 1: Graph of the error over time for each algorithm

Appendix B

Graphs of each combination of Pressure/Temperature. The green line represents the graph of the Ideal Gas Law at the given Pressure and Temperature (enumerated in the title of each graph) while the purple line represents the Newton's Method Estimation function on the interval. v is the modal volume estimated by Newton's Method.

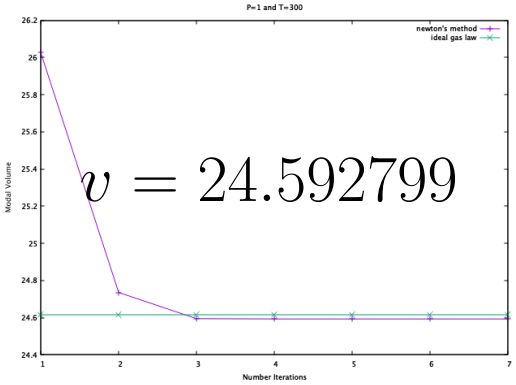

Figure 2: $P = 1atm$, $T = 300K$

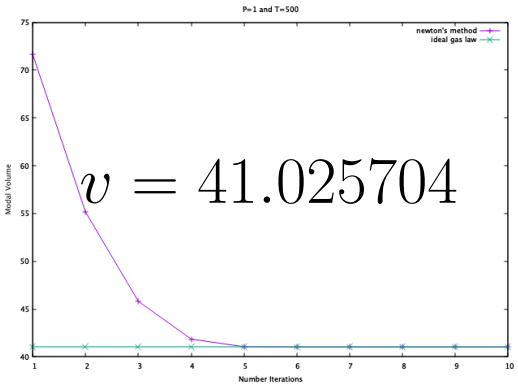

Figure 3: $P = 1atm$, $T = 500K$

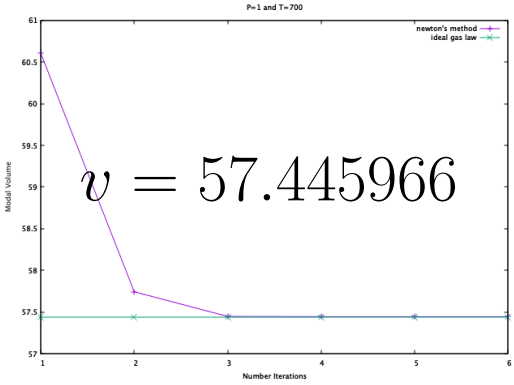

Figure 4: $P = 1atm$, $T = 700K$

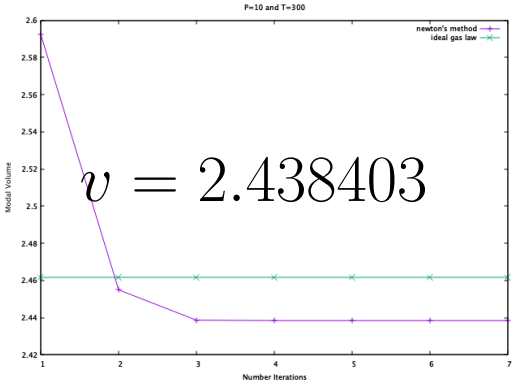

Figure 5: $P = 10atm$, $T = 300K$

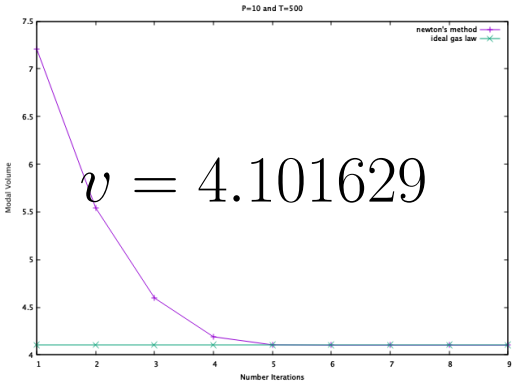

Figure 6: $P = 10atm$, $T = 500K$

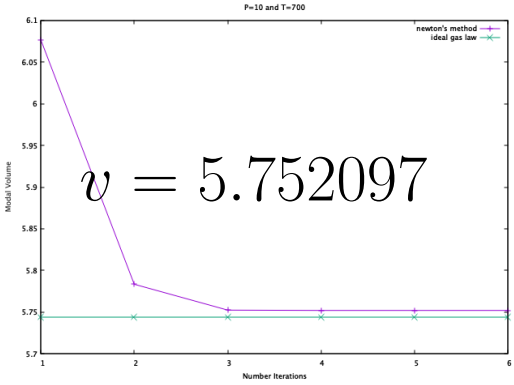

Figure 7: $P = 10atm$, $T = 700K$

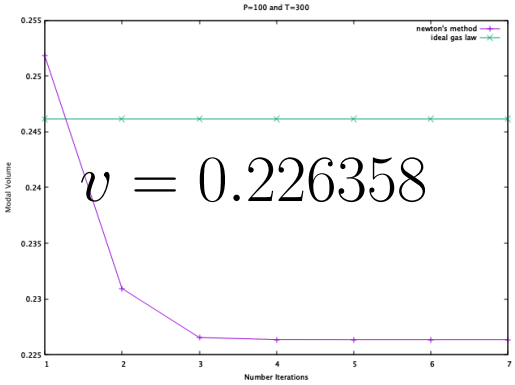

Figure 8: $P = 100atm$, $T = 300K$

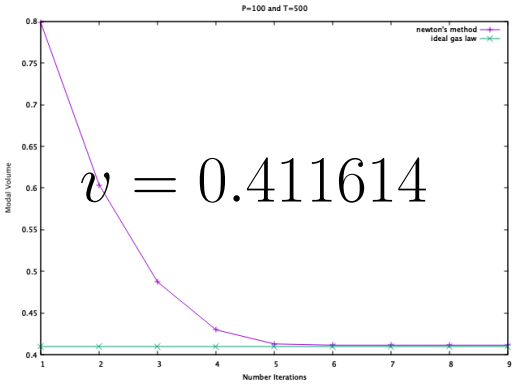

Figure 9: $P = 100atm$, $T = 500K$

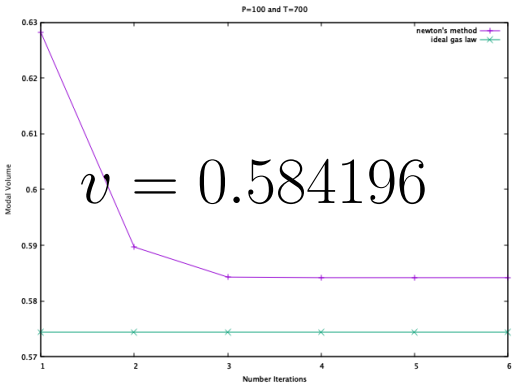

Figure 10: $P = 100atm$, $T = 700K$

Appendix C

```
program waals_equation
    implicit none
    doubleprecision R, a, b, true
    integer n, P, T
    doubleprecision x_oo, x_o, x_n, tol, err, r1, r2, delta
    doubleprecision step, func
    doubleprecision x_ioo, x_io, x_i, diff

    !INITIALIZE VARIABLES
    R = 0.082054    !atm/(mol*K)
    a = 1.360
    b = 0.03183
    P = 10          !atm
    T = 300         !K

    ! Do not know if this is that actual true value. This is just as exact as
    !!!! Fortran can get
    !!!Temp = 300
    !true = 24.592799
    true = 2.438403
    !true = 0.226358
    !!!Temp = 500
    !true = 41.025704
    !true = 4.101629
    !true = 0.411614
    !!!Temp = 700
    !true = 57.445966
    !true = 5.752097
    !true = 0.584196

    tol = 10.0**(-6)   !tolerance

    !initial estimate
    x_n =  3
    x_o =  4
    x_oo = 5
    delta = .01

    !possible P and T values
    !P = 1, 10, 100
    !T = 300, 500, 700
    P = 10; T = 300
    !estimate the roots, maximum of 500 iterations
    estimate : do n = 1, 1000
        !function:
        !waals(x_n, R, a, b, P, T)
        !ideal(r2, R, P, T) * r1)

        !!!Newton's Method
        x_n = x_o - (waals(x_o, R, a, b, P, T) / dwaals(x_o, R, a, b, P, T))
```

```
50          x_n = x_o - (waals(x_o, R, a, b, P, T) / dwaals(x_o, R, a, b, P, T))
51          err = abs(x_o - x_n) !approximate error
52          x_o = x_n
53          x_oo = x_o
54
55          !!!Secant Method
56          !x_n = x_o - ((waals(x_o, R, a, b, P, T) * (x_o - x_oo)) / (waals(x_o, R, a, b, P, T) - waals(x_oo, R, a, b, P, T)))
57          !x_oo = x_o
58          !x_o = x_n
59
60          !!!Modified Secant Method
61          !x_n = x_o - ((waals(x_o, R, a, b, P, T) * delta) / (waals(x_o + delta, R, a, b, P, T) - waals(x_o, R, a, b, P, T)))
62          !x_o = x_n
63
64          !!!Modified Newton's Method for Roots of Multiplicity
65          !x_n = (waals(x_o, R, a, b, P, T) * dwaals(x_o, R, a, b, P, T))
66          !x_n = x_n / ((dwaals(x_o, R, a, b, P, T))**2 - waals(x_o, R, a, b, P, T) * ddwaals(x_o, R, a, b, P, T))
67          !x_n = x_o - x_n
68          !x_o = x_n
69
70          !ideal gas law  values
71          x_i = ideal(x_i, R, P, T)
72
73          write(*, *) n, x_n, err
74          write(7, *) n, x_n
75          write(8, *) n, x_i
76
77          !determine if error is small "enough"
78          if(err < tol .and. n > 1) then
79              write(*, *) "What I found: ", x_n
80              !do not know the true value
81              !write(*, *) "Actual value: ", true
82              write(*, *) n, " iterations"
83              exit
84          end if
85      end do estimate
86
87      CALL SYSTEM('gnuplot script.sh')
88
89  contains
90
91      !van der Waal's equation of state
92      double precision FUNCTION waals(V, R, a, b, P, T)
93          IMPLICIT NONE
94          doubleprecision :: R, a, b, V
95          integer :: P, T
96          waals = P * V**3 - (b * P + R * T) * V**2 + a * V - a * b
97      END FUNCTION waals
```

```
99      double precision FUNCTION dwaals(V, R, a, b, P, T)
100         IMPLICIT NONE
101         doubleprecision :: R, a, b, V
102         integer :: P, T
103         dwaals = 3 * P * V**2 - 2 * (b * P + R * T) * V + a
104     END FUNCTION dwaals
105
106     double precision FUNCTION ddwaals(V, R, a, b, P, T)
107         IMPLICIT NONE
108         doubleprecision :: R, a, b, V
109         integer :: P, T
110         ddwaals = 6 * P * V - 2 * (b * P + R * T)
111     END FUNCTION ddwaals
112
113     !Ideal Gas Law
114     double precision FUNCTION ideal(V, R, P, T)
115         IMPLICIT NONE
116         doubleprecision :: R, V
117         integer :: P, T
118         ideal = (R * T) / P
119     END FUNCTION ideal
120
121 end program waals_equation
```

```
#!/usr/bin/gnuplot

set title "P=100 and T=700"

set xzeroaxis linestyle 8
set xtics axis 1
set xlabel "Number Iterations"
set ylabel "Modal Volume"
plot "fort.7" title "newton's method" with linespoints, "fort.8" title "ideal gas law" with linespoints

### PLOT 4 GRAPHS AT ONCE ###
#plot [0:] "fort.7" title "newton's" lw 2 with linespoints, \
#    "fort.8" title "secant" lw 2 with linespoints, \
#    "fort.9" title "mod secant" lw 2 with linespoints, \
#    "fort.10" title "mod newton's" lw 2 with linespoints, \

#set table "plot.tex"
#plot "fort.7"
#unset table

pause -1
```