Formosa Journal of Computer and Information Science (FJCIS)
Vol.2, No.1, 2023:51-94

Implementing Newton-Raphson Method Algorithm in 6 Programming Languages' Solution to Van Der Waals Equation of State

Abdulhalim Musa Abubakar[1*], Eva Schieferstein[2], Volodymyr Kutarov[3], Muhammad Tayyab Bilal[4], Chantawan Noisri[5], Bello Iliyasu[6]
[1]Department of Chemical Engineering, Faculty of Engineering, Modibbo Adama University, Adamawa State, Nigeria
[2]Fraunhofer Institute UMSICHT, Germany
[3]Research Institute of Physics, I. I. Mechnikov National University, Ukraine
[4]Computer Science Department, University of Agriculture Faisalabad Punjab, Pakistan
[5]Department of Mathematics and Statistics, Faculty of Science, Thaksin University, Phatthalung, Thailand
[6]Department of Mathematics, Modibbo Adama University, Adamawa State, Nigeria
Corresponding Author: Abdulhalim Musa Abubakar
abdulhalim@mau.edu.ng

ARTICLEINFO

Keywords: Van Der Waals Equation of State, Newton Method, Programming Language, Numerical Methods, Java

Received: 1 February
Revised : 17 February
Accepted: 17 March

ABSTRACT

Newton's method can be implemented for the Van der Waals equation of state (VDW EOS) and its derivatives. Application of programming language in chemical engineering is not new but however limited in practice, as it is not included in the curriculum of students of tertiary institutions across the globe. Finding roots through iteration for the VDW EOS can be described as interesting and simple using C++, FORTRAN, R, MATLAB, Java and Python languages; even though a perfect programming language does not exist. In this work, the coding syntax in the six languages employed to solve parameters in VDW EOS differs in terms of simplicity and user-friendliness. Notwithstanding, all of them proves to be an adaptable software, capable of solving chemical engineering problems leading to nonlinear relations

DOI: https://doi.org/10.55927/fjcis.v2i1.3953
ISSN-E: 2830-3040
https://journal.formosapublisher.org/index.php/fjcis

INTRODUCTION

Engineers attach much significance to numerical analysis to solve nonlinear analytical models by employing data from empirical procedures (Bect et al., 2016; Djamila, 2017; Hossain, 2022). Their task is made easier utilizing free or commercial software programs like Minitab, Excel, Mathematica, POLYMATH, Maple and MATLAB. Now, engineers had moved a step further to consult programming language packages, such as MATLAB, R language, C++, Python, FORTRAN and Java to do the same job. In chemical engineering, for instance, nonlinear/cubic equation of state (EOS) like the Van der Waals (VDW) EOS, including complex simulations, can be simplified and solved using any programming language (David, 2015; Hawick, 2011; Terrel, 2011; Vasudevan et al., 2019). Cubic EOS like VDW EOS is best solved using Newton-Raphson method using programming languages due to its rapid rate of convergence to the root, single initial guess, easy programming effort and accuracy (Bakari et al., 2016; Balaji & Seader, 1995). Previously, Abubakar & Mustapha (2021), Shukla & Singh (2022), Nasri & Binous (2009) and Sun et al. (2011) used one of C++, MATLAB and Python programs to solve either, VDW, Peng-Robinson, Redlich-Kwong or Soave-Redlich-Kwong EOS with simple codes. This were carried out after Kassim and Cadbury concluded that programming should be included in undergraduate chemical engineering curriculum since 1996 (Andika & Putra, 2022; Teles et al., 2018). Ever since, a sporadic rise in coding in the field, especially the one that relates to mathematical methods was witnessed (Ahuja, 2019; Hinsen, 2013; Salamanca, 2020).

VDW EOS is an approximate prediction or an extension of the ideal gas law applied in the study of gases and liquids molecules interaction (Garces, 2015). As a PhD. Thesis, Johannes Diderik Van der Waals created this equation that points to the existence of a gas-liquid transition as well as critical points, in 1873 (Johnston, 2014; J. Tian & Gui, 2003). VDW equation is simple and flexible as shown in Equation 1 and 2 in molar volume and volume versions of the equation (Kutarov & Schieferstein, 2020; Saggion et al., 2019).

$$\left(P + \frac{a}{V_m^2}\right)(V_m - b) = RT \qquad (1)$$

$$\left(P + \frac{an^2}{V^2}\right)(V - nb) = nRT \qquad (2)$$

Where, P, V, T stands for pressure, volume and temperature, n = number of moles, R = molar gas constant, $V_m = \frac{V}{n}$ = molar volume (density) and a, b = VDW constants. In Equation (1), $P + \frac{a}{V_m^2}$ determines the intermolecular forces while $(V_m - b)$ determines the effective volume calculated from non-ideal gases (Frank & GroBmann, 2006; Garces, 2015). A gas described by the VDW equation behaves similar to an ideal gas in the limit of large V_m. At large V_m, molecules will be far apart, with negligible intermolecular interactions. Moreover, the volume

occupied by molecules will be negligible in comparison to the V_m. In the limit of large pressure, the compressibility factor z >1; hence, the gas shows deviation from ideal behavior. Also, in Equation (1), the term, $P + \frac{a}{V_m^2}$ corresponds to ideal gas pressure; hence, $P_{real} < P_{ideal}$. So, when pressure is increased drastically, Equation (2) becomes: $\left(P + \frac{an^2}{V^2}\right)(V) = RT$, as the molecules of the gases are forced to come close to each other, thereby decreasing the volume occupied by the gases. That's the reason the effective volume constant, b, was ignored (Prodanov, 2022).

VDW constants are functions of critical temperature and pressure, expressed as shown in Equation (3) and (4) and found in special tables for different gases (see Appendix).

$$a = \frac{27}{64}\frac{R^2 {T_C}^2}{P_C} \quad (3)$$

$$b = \frac{RT_C}{8P_C} \quad (4)$$

The temperature below which compression leads to the formation of the liquid phase and above which the fluid remains in the gaseous state or above which the gas cannot become liquid, regardless of the applied pressure is called critical temperature, T_C (Saggion et al., 2019). The critical pressure, Pc, is defined as the pressure above which liquid and gas cannot coexist at any temperature. Since 'a' is measure of the intermolecular attractions, gases with higher 'a' has greater attraction; while 'b', which is the co-volume, can be referred to as the four-fold volume of the gas particles (Frank & GroBmann, 2006). These two constants characterize a real gas and are independent of temperature. Many years after, the VDW EOS has seen several modifications (Boynton & Bramley, 1922; Kontogeorgis et al., 2019; Papari et al., 2011; Rault, 2019; Shrab, 2004; J. Tian & Gui, 2003; J. X. Tian & Gui, 2018).

Objective of this work is to study the basis of programming with R, C++, MATLAB, Java, FORTRAN and Python softwares in order to implement a comprehensible code solution to VDW EOS. Concerns would be, a way to solve P, V, T, n and V_m, with special emphasis on the solution of n, V and V_m, because they require a series of iteration before arriving at their roots. Furthermore, VDW EOS, would hence be manipulated or re-written in-terms of any of the relevant parameters stated and to find the derivative expression (where applicable) so as to iterate them using Newton-Raphson method by codes. The work would also involve, a quick validation using a sample gas problem in the six programmed syntax selected. In addition, the C++ and Java versions of the source code would be translated into a comprehensive flowchart. Before this, a mini-review of Java, Python, MATLAB, FORTRAN, C++ and R programming languages was carried out.

1. Programming Languages

Needs for the creation of interactive networked programs was the sole aim of designing a strikingly all-purpose programming language called Java in 1995 by Sun Microsystems, Inc. (Moreira et al., 2000; Scheiber, 2007; Schildt, 2022). Java's progressive pool of skillful programmers/software developers in academia and industry or scientific and engineering arenas, is due to its suitability as a strong programming language; because of its security, cross-platform portability, simple object semantics, absence of pointers and ability to vigorously check array accesses to circumvent common bugs (Blount & Chatterjee, 1999; Farrell, 2022; Moreira et al., 2000). Long ago, Java was observed to be 500 times slower than Fortran and C++ programs, a dilemma that is now a thing of the past (Boisvert et al., 2001). Further progress led to an open-source software tool called Easy Java Simulations (EJS) which translates core simulation algorithm entered to a Java code. Example is a series of three stirred tank system's dynamic behaviors and control modeled using Java and showcased by Salcedo-Diaz et al. (2006). Here, few among several Java packages would be selected to solve specific numerical problems using Newton-Raphson method, as previously demonstrated in the literature (Bishop & Bishop, 2000; Li, 2022; Scheiber, 2007).

All kinds of engineering and scientific tasks can be run using the Python programming language, making it the most in-demand skill by employers (Gor, 2021). As a unique feature, setting aside memory and declaring variables are not necessary before use as codes are run without compiling in Python (Pine, 2019). Python is free, easy to learn and use, with libraries such as Pandas, NumPy, SciPy and Matplotlib for speedy numerical computation (Baptista, 2021; Gor, 2021; Hart & Laird, 2014; Pandey et al., 2020; Pine, 2019). However, speed is an issue when it comes to large-scale simulation and difficult applications, having less comparative advantage to FORTRAN and C++ (Pine, 2019). Solofsson et al. (2019) and Wang & Dowling (2022) demonstrated how a model-base tool in Python can be used for design of experiments – maximizing statistics gain from experiments while lessening time and resource costs; just in physical chemistry lab described by Marie (2022). A web platform called Google Colab enables the running of Python codes using Google's infrastructure (Baptista, 2021). Gor (2021) and Baptista (2021) previously worked on how to introduce Python to undergraduate chemical engineering students in their respective institutions to solve problems in fluid flow, thermodynamics, reactor design and kinetics. In chemical engineering (Hernandez & Martin, 2019), modeling and optimization as demonstrated by Hart & Laird (2014), calculation of interfacial properties of pure fluids showcased by Mejia et al. (2021), a Python program by Shukla & Singh (2022) to solve Peng-Robinson and Redlich-Kwong EOS, and HidroUFF Density Calculator software designed based on the Python language by de Sousa

et al. (2021), among others (Albrecht, 2021) are typical applications. Kiusalaas (2013) previously provided a simple Python3 code syntax to solve cubic polynomials using Newton's method. Python was developed by Guido van Rossum in 1989 (Nguyen, 2019; Zehra et al., 2020).

Mathematical and technical computing in engineering fields can be carried out using readily available tools provided by MATLAB developed by MathWorks.Com (Asadi, 2022; Mehtre & Pal, 2019). The tools are housed in 3 windows, namely, the command, graphics and edit windows used to model and simulate any physical problem in the engineering domain (Al-Malah, 2014; Mehtre & Pal, 2019). Though comprehensive, it could be found cumbersome by beginners running the program in any of Windows, Unix or Macintosh environments. Countless use of MATLAB in chemical engineering analysis has been reported in previous research (Yeo, 2017). Solution to numerical methods problems (Hossain, 2022), for example, has since been demonstrated in Peng-Robinson and Soave-Redlich-Kwong solutions to EOS by Nasri & Binous (2009) and Sun et al. (2011) respectively using MATLAB. This can be achieved using the routine '*fsolve*' in MATLAB to solve general systems of nonlinear equations; as seen in generic syntax for MATLAB coding of the Newton-Raphson method given by Rose (2017).

FORTRAN programming language, since its inception in 1957 has been used to develop many computer models (Mak & Taheri, 2022; Ott et al., 2020). The software development transitioned from FORTRAN 77, 90/95, 2003, 2010 to FORTRAN 2018 (Bose, 2019; Johnson et al., 2019). Additional tools to assess FORTRAN codes' quality and upgrade them, despite its high-level array support, portability, stability, predictable and controllable performance, longevity, low runtime overhead, productivity and ease of use has been developed (Garcia-Rodriguez et al., 2016; Mak & Taheri, 2022). Such act is termed 'refactoring', and is meant to improve quality and readability and increase speed and performance. FORTRAN is not obsolete as it is earlier believed, looking at over 14000 citations mentioning FORTRAN 77 alone in Google Scholar between 2011-2022 (Kedward et al., 2022; Mak & Taheri, 2022). In some chemical engineering text, Euler method of solving differential equations was applied to solve problems in material balances and control (Bose, 2019; Pao, 2018). Furthermore, Tholl (2010) wrote on chemical process design analysis and simulation with FORTRAN programs. Siddiqui & Ahmad (2020) discussed the theory governing Newton's method and its algorithm for solving actual data using FORTRAN program. Also, Shafeek & Karunarathne (2018), created a novel compiler capable of translating a source-code from a programming language to flowcharts.

In 1979, Bjarne Stroustrup created the 'C with class', otherwise called the C language, which was later renamed to C++ in 1983 and released in 1985 (Nguyen, 2019; Zehra et al., 2020; Zheng et al., 2019). It is a case-sensitive language capable of deployment in scientific and engineering computing (Bergmann, 2021; Johnson et al., 2019). It can be used to program automated teller machine (ATM) transactions, linear programming problems, numerical analysis, counters, student grading system, matrix solutions, game applications and scientific software development (Adams, 2021; Passos, 2009). In chemistry (Rassokhin, 2020), mathematics (Bandele & Adekunle, 2015; Scheinerman, 2006), engineering science (Nyhoff, 2012) and chemical engineering (Abubakar & Mustapha, 2021; Kadam, 2013; Kapuno, 2008; Tang & Wang, 2019), specific areas C++ can be absolved into solving problems in those fields were previously practicalized. For instance, how to command C++ to execute the partial derivative of functions in chemical engineering thermodynamics, how to find molar volume from four cubic EOS and how to execute simplex cost optimization algorithm in chemical engineering production companies, were clearly illustrated by Bell et al. (2022), Abubakar & Mustapha (2021) and Abubakar et al. (2021), respectively. In several other fields of study, Allen & Vahid (2020), Nguyen (2019), Winkel & Bella (2018), Finkel et al. (1994), Kamarudin et al. (2022), Lee & Phillips (1998), Xiduo et al. (2020) and Jarvinen & Ala-Mutka (2004), after analyzing the significance of C++, propose it for teaching and learning in high schools and universities of their respective countries. However, system failure, memory corruption and difficulty debugging pointers are major problems associated with C++ that makes Python by far an easier programming language for beginners (Ateeq et al., 2014; Meltzer, 2021; Nguyen, 2019; Zehra et al., 2020).

Data analysis and visualization, graphics and statistical computing (e.g., classical statistical tests, classification, linear and nonlinear modelling, clustering rate and time series analysis, among others) can be conducted in R programming language (Blagoev, 2018; Schauber, 2015). It is a free, portable and open source software environment that works absolutely well in OS X, Windows and Linux (Evans, 2014; Ozgur et al., 2017; Schauber, 2015). It was created in 1991 by Ross Ihaka and Robert Gentleman and used by various companies including Astra, Baxter Healthcare Corporation, Merck, AT & T Labs, Google and several others (Blagoev, 2018; Ozgur et al., 2017; Peng, 2018). In the field, data scientists, statisticians, geneticists, and biostatisticians find the programming language worthy of adoption (Chan, 2018; Evans, 2014). But to algebraically integrate a function, precision graphics and solve optimization and partial differential equations, R appears to be an unsuitable programming platform (Evans, 2014). Apart from this author who also described the application of R in chemical engineering (Abubakar et al., 2023), no published manuscript had described with

practical example, its application in chemical engineering fields; though, Diaz-Bejarano et al. (2019) previously propose it for undergraduate teaching in the field. The Department of Statistics and Mathematics in many Western European Universities and the United States, had since began teaching R language to their ward (Blagoev, 2018; Braun, 2021).

METHODOLOGY

Six programming languages, namely, Java, FORTRAN, Python, R, C++ and MATLAB were installed to a personal laptop computer. Their respective model and versions are as shown in Table 1.

Table 1. Software Version Used

Programming Software	**Version**
C++	Dev C++ 5.11 TDM-GCC 4.9.2
Python	Python 3.11.1 Pip Bootstrap (64-bit)
FORTRAN	FTN95 Personal Edition
MATLAB	MATLAB R2022b: Online Version
Java	Java 8 Update 351: October 18, 2022 Release
R	R version 4.2.2 (2022-10-31 ucrt)

Similarities and differences (Ozgur et al., 2017; Sudhaka, 2018) as regard the use of the listed programming source codes was studied in order to execute the task. VDW EOS as shown in Equation (1) and (2) were manipulated into different forms, that would allow the calculation of a particular parameter in question. Problem 1 was selected out of six sample problems that can be solved using VDW EOS as shown in Table 2.

Table 2. Typical Problem Set for Solution Using EOS

Number	**Problems**
Problem 1	1.00 mol of ammonia (NH_3) fills a 7.00 litre bottle at 350K. What does the VDW equation predict that the pressure will be? For ammonia: a = 4.1969 $\frac{atm\ L^2}{mol^2}$ and b = 0.0374 L/mol
Problem 2	Calculate the pressure exerted by 0.4891 mol of N_2 in a 1 L container at 27℃. Compare the results with the ideal gas equation.
Problem 3	Calculate the temperature of a container with 10.76 atm pressure exerted by 1.502 mol of CO_2 in a 3.5 L.
Problem 4	A sample of 7.50 kg gaseous oxygen fills a 100 L flask at 289℃. What is the pressure of the gas, calculated from the VDW EOS?
Problem 5	Estimate the molar volume of CO_2 at 500K and 100 atm by treating it as a VDW gas with constants a = 3.64 atm L^{-2} mol^{-2} and b = 4.267×10^{-2} $Lmol^{-1}$.
Problem 6	Use the VDW equation and the ideal gas equation to calculate the volume of 1 mol of neon at a pressure of 500 atm and a temperature of 355K. Explain why the two are different.

Problem 1 was first solved manually using method described by Bamdad (2004), to have a glimpse of all the variables outcome or specific values. The known parameters were then used in the programming language to solve iteratively, declared unknown variable in the manipulated functions, as way of confirming the validity of the programmed codes.

A summarized flowchart for the C++ program written was generated using Code to Flowchart version 2.0, released in March 31, 2013. A flowchart for iterative number of moles estimation for the Java source code was also produced.

RESULTS AND DISCUSSION

1. Equation Customization

Different representation of the VDW EOS after manipulating Equation (1) and (2) is shown in Table 3, in form of Equation (5) to (14).

Table 3. Functions (f), Derivative Functions (f') and Other Customized Parameters in VDW EOS

Variables	Equation Number
Number of Moles (n):	
$f(n) = n^3 - \left(\frac{V}{b}\right)n + \left[\frac{P}{ab}\left(b + \frac{RT}{P}\right)V^2\right]n - \frac{PV^3}{ab} = 0$	(5)
$f'(n) = 3n^2 - 2\left(\frac{V}{b}\right)n + \frac{P}{ab}\left(b + \frac{RT}{P}\right)V^2$	(6)
Molar Volume (V_m):	
$f(V_m) = V_m^3 - \left(b + \frac{RT}{P}\right)V_m^2 + \left(\frac{a}{P}\right)V_m - \frac{ab}{P} = 0$	(7)
$f'(V_m) = 3V_m^2 - 2\left(b + \frac{RT}{P}\right)V_m + \frac{a}{P}$	(8)
Volume (V):	
$f(V) = V^3 - \left(nb + \frac{nRT}{P}\right)V^2 + \left(\frac{an^2}{P}\right)V - \frac{an^3b}{P} = 0$	(9)
$f'(V) = 3V^2 - 2\left(nb + \frac{nRT}{P}\right)V + \frac{an^2}{P}$	(10)
Temperature:	
$T = \frac{\left(P + \frac{an^2}{V^2}\right)(V - nb)}{nR}$	(11)
$T = \frac{\left(P + \frac{a}{V_m^2}\right)(V_m - b)}{R}$	(12)
Pressure:	
$P = \frac{nRT}{V - nb} - \frac{an^2}{V^2}$	(13)
$P = \frac{RT}{V_m - b} - \frac{a}{V_m^2}$	(14)

Unlike the customized versions for temperature and pressure, in terms of both V and V_m in Equation (11) to (14), others can only be functionalized in terms of the desired variables (i.e., n, V and V_m), as indicated in Equation (5) to (10). Thus, the functions being nonlinear, can only be solved using a suitable numerical method. Newton-Raphson method, as given in Equation (15) requires the finding and utilization of the derivative of the functions shown in Table 3 to compute values of the unknown in successive iteration, till a repeat is witnessed – which is then declared as the root of the function equal to 0.

$$x_{i+1} = x_i - \frac{f(x_i)}{f\prime(x_i)} \qquad (15)$$

Where, i = 0, 1, 2, … (number of iteration) and x = unknown variable (i.e., n, V and V_m, in this case). Procedure starts by guessing an initial value called x_0. It is recommended to calculate the initial parameter from the ideal gas law ($PV = nRT$ or $PV_m = RT$) when dealing with EOS. In this case, initial guess for functions with independent variables, n, V and V_m are given by Equation (16).

$$\left.\begin{aligned} n_0 &= \frac{PV}{RT} \\ V_0 &= \frac{nRT}{P} \\ V_{m_0} &= \frac{RT}{P} \end{aligned}\right\} \qquad (16)$$

2. Solution to Problem 1

Combinations of Equation (1) through (16) was used to complete the remaining properties of NH_3 in the bottle described by Problem 1. Hence, Table 4 represents a complete assay of the problem under consideration, which can then be used as a reference input in the respective programming languages picked.

Table 4. Known and Calculated Parameters in Problem 1

Parameter	**Value**	**Parameter**	**Value**
R =	0.08206 $\frac{L\,atm}{mol\,K}$	$V_{m_{ideal}} = V_{m_0}$=	7.1092 L
P =	4.0394 atm	n =	1 mol
T =	350 K	$n_{ideal} = n_0$ =	0.9845 mol
T_C =	405.5 K	a =	4.1969 $\frac{atm\,L^2}{mol^2}$
P_C =	111.3 atm	b =	0.0374 L/mol
V =	7 L	MW =	17 Kg/ mol
$V_{ideal} = V_0$ =	7.1092 L	m =	17 kg
V_m=	7 L		

Where, MW = Molecular Weight and $m = n \times MW$ = Mass of NH_3

R was obtained from ideal gas equation for a unit mol at 1 atm, 273K and 22.4 L standard conditions. Critical conditions, T_C and P_C, for NH_3 was obtained from the Appendix. This can be used to calculate the VDW constants using Equation (3) and (4), if not provided or directly sought from VDW constants tables in the Appendix.

3. FORTRAN Newton-Raphson Implementation

In order to solve for volume using FORTRAN, the volume function and its derivative was declared 'f' and 'g' within the program syntax, as shown in Figure 1.

```
PROGRAM vdw
IMPLICIT NONE

REAL::f,g,ea=6,es=1,v1,v0
INTEGER::c=0

PRINT *, "VAN DER WAALS EQUATION OF STATE"
WRITE(*,*)
PRINT *, " By Engr. Abdulhalim M. A."
WRITE(*,*)
PRINT *, "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
PRINT *, "VOLUME CALCULATION USING NEWTON-RAPHSON METHOD"
PRINT *, "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
WRITE(*,*)

PRINT *,'Initial Volume Approximation  = ?'
READ(*,*)v0                  ! Hint: Calculate Initial Value from
WRITE(*,*)                   !       Ideal Gas Equation

DO WHILE(ea>es)
v1=v0-((f(v0))/(g(v0)))      ! Newton's Equation
ea=abs(((v1-v0)/v1)*100)     ! Absolute Error
v0=v1
c=c+1                        ! Number of Iterations
IF(c>50) EXIT
PRINT *,'CURRENT ROOT IS',V1
END DO

PRINT *,'FINAL ROOT IS',V1

END PROGRAM
REAL function f(v1)
REAL::v1

f=(v1**3)-7.147585603*(v1**2)+1.039002295*v1-0.038828778
                             ! Function of Volume
RETURN
END FUNCTION

REAL function g(v2)
REAL::v2

g=3*(v2**2)-2*7.147585603*v2+1.039002295
                             ! Derivative of Volume
RETURN
END FUNCTION
```

Figure 1. FORTRAN Source Code for Finding Volume in VDW EOS

Functions 'f' and 'g' in the source code are simply Equations (9) and (10). For simplification, the coefficients of V in both functions were replaced with a real value. Approximating the coefficients is not encouraged here, in order to avoid errors that might result or convergence difficulty. FORTRAN uses 'PRINT' or 'WRITE' and 'READ' to display text or results and request inputs, respectively. Figure 2 is the FORTRAN output window, requesting an initial estimate of V to kick-start the computation.

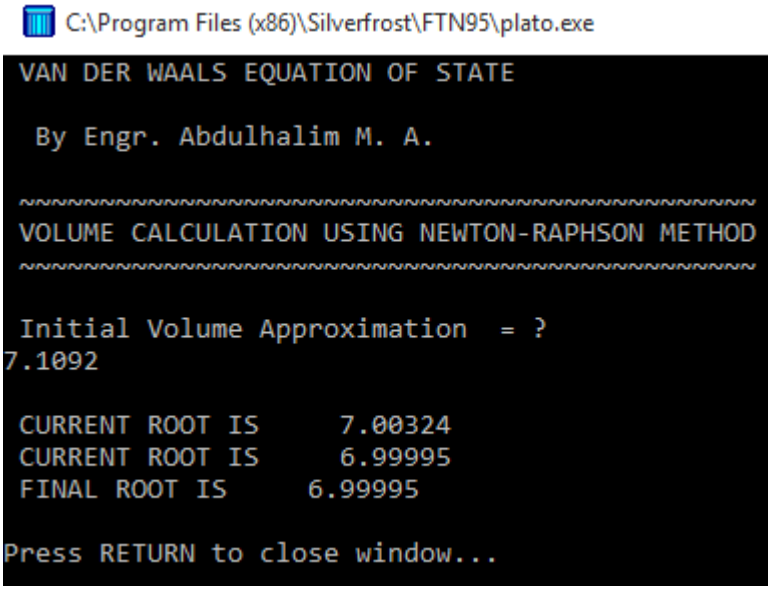


Figure 2. Root of the Volume Function in FORTRAN Output Window

A repeat in value of the root in any numerical calculations using Newton's Method points to the arrival at a solution or convergence. In 3 iterations, V ≅ 7.0 L, starting with V_0 = 7.1092 L from ideal gas law, as shown in Figure 2.

The same principle followed to obtain V was repeated to determine 'n' using Equations (5) and (6), as shown in Figure 3.

```
PROGRAM vdw
IMPLICIT NONE

REAL::f,g,ea=6,es=1,n1,n0
INTEGER::c=0

PRINT *, "VAN DER WAALS EQUATION OF STATE"
WRITE(*,*)
PRINT *, " By Engr. Abdulhalim M. A."
WRITE(*,*)
PRINT *, "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
PRINT *, "MOLES CALCULATION USING NEWTON-RAPHSON METHOD"
PRINT *, "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
WRITE(*,*)

PRINT *,'Initial Number of Moles Approximation  = ?'
READ(*,*)n0                    ! Hint: Calculate Initial Value from
WRITE(*,*)                     !       Ideal Gas Equation

DO WHILE(ea>es)
n1=n0-((f(n0))/(g(n0)))        ! Newton's Equation
ea=abs(((n1-n0)/n1)*100)       ! Absolute Error
n0=n1
c=c+1                          ! Number of Iterations
IF(c>50) EXIT
PRINT *,'CURRENT ROOT IS',N1
END DO

PRINT *,'FINAL ROOT IS',N1

END PROGRAM
REAL function f(n1)
REAL::n1

f=(n1**3)-187.309938*(n1**2)+9019.899913*n1-8833.654173
                               ! Function of Number of Moles
RETURN
END FUNCTION

REAL function g(n2)
REAL::n2

g=3*(n2**2)-2*187.309938*n2+9019.899913
                               ! Derivative of Number of Moles
RETURN
END FUNCTION
```

Figure 3. Source Code in FORTRAN Environment to Execute Number of Moles in VDW EOS

Similarly, 'f' and 'g' were used to represent the function of the number of moles and its derivative, respectively, while replacing the coefficients of the independent variable with real values. Figure 4 produces the answer to the mole calculation, starting with $n_0 = \frac{PV}{RT}$, calculated using their values, given in Table 4.

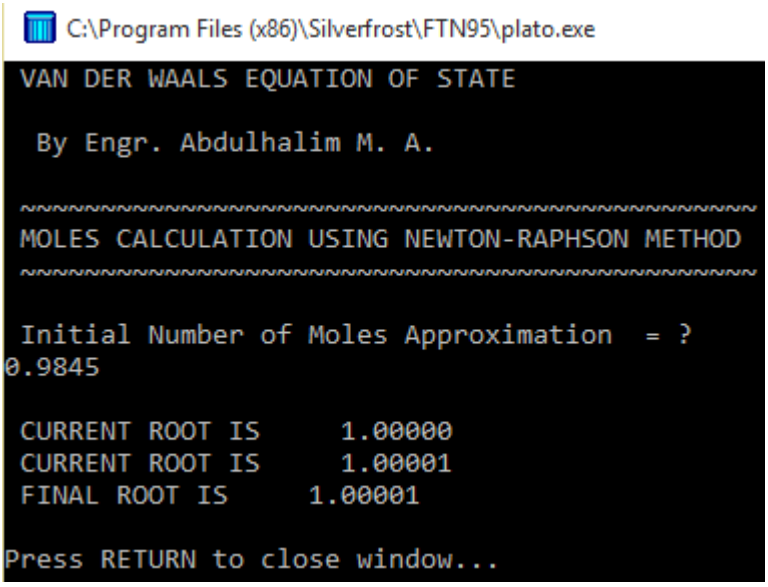


Figure 4. Iterations for Number of Moles Calculation in FORTRAN Result Window

Where the basis is 'per mole', Equation (7) and (8) is most suitable to be applied. Functions of molar function and its derivative were interpreted in FORTRAN using the equations, same way it was presented when calculating 'n' and 'V'. FORTRAN source code for the solution of V_m in Figure 5 is same with Figure 1, because n = 1 mole.

```
PROGRAM vdw
IMPLICIT NONE

REAL::f,g,ea=6,es=1,mv1,mv0
INTEGER::c=0

PRINT *, "VAN DER WAALS EQUATION OF STATE"
WRITE(*,*)
PRINT *, " By Engr. Abdulhalim M. A."
WRITE(*,*)
PRINT *, "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
PRINT *, "MOLAR VOLUME CALCULATION USING NEWTON-RAPHSON METHOD"
PRINT *, "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
WRITE(*,*)

PRINT *,'Initial Molar Volume Approximation  = ?'
READ(*,*)mv0                    ! Hint: Calculate Initial Value from
WRITE(*,*)                     !       Ideal Gas Equation

DO WHILE(ea>es)
mv1=mv0-((f(mv0))/(g(mv0)))      ! Newton's Equation
ea=abs(((mv1-mv0)/mv1)*100)     ! Absolute Error
mv0=mv1
c=c+1                           ! Number of Iterations
IF(c>50) EXIT
PRINT *,'CURRENT ROOT IS',MV1
END DO

PRINT *,'FINAL ROOT IS',MV1

END PROGRAM
REAL function f(mv1)
REAL::mv1

f=(mv1**3)-7.147585603*(mv1**2)+1.039002295*mv1-0.038828778
                               ! Function of Molar Volume
RETURN
END FUNCTION

REAL function g(mv2)
REAL::mv2

g=3*(mv2**2)-2*7.147585603*mv2+1.039002295
                               ! Derivative of Molar Volume
RETURN
END FUNCTION
```

Figure 5. Implementing Molar Volume Calculation Using FORTRAN Source Code

In all the FORTRAN codes, limit of iterations ('c') is put at 49, even though, number of iterations doesn't go beyond 10 in majority of nonlinear numerical analysis. Newton's method is known for its rapid convergence to the root (Figure 6), and is clearly observed here, as the 3 FORTRAN solutions gives the roots in question in just 3 iterations.

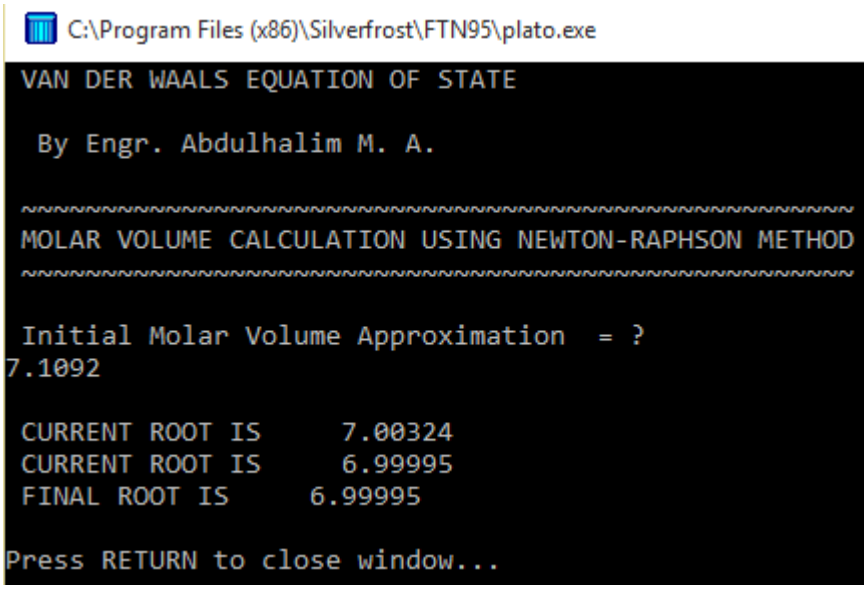


Figure 6. Stepwise Root Computation of the Molar Volume Function After FORTRAN Code Execution from VDW EOS

Apart from guessing the ideal gas values as demonstrated in Figure 2, 4, and 6, any other starting value not too far from the root would converge, provided the tolerance limit is not exceeded as declared in all FORTRAN codes.

Gas Properties Calculation Using C++

Figure 7 is best among all the source code solution provided in this work, as it is able to calculate all gas properties without any preliminary calculations.

VDW EOS.cpp

```cpp
/* Iterative Volume Computation by Newton's Method for Cubic EOS */
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;
// VANDER WAALS EOS-FUNCTION OF MOLAR VOLUME
#define f(MV) MV*MV*MV-SE1*MV*MV+(aVDW_1/P)*MV-(aVDW_1*bVDW_1)/P
#define d(MV) 3*MV*MV-2*SE1*MV+aVDW_1/P
// VANDER WAALS EOS-FUNCTION OF NUMBER OF MOLES
#define x(n) n*n*n-SE2*n*n+SE3*n-SE4
#define y(n) 3*n*n-2*SE2*n+SE3
// VANDER WAALS EOS-FUNCTION OF VOLUME
#define g(V) V*V*V-SE5*V*V+SE6*V-SE7
#define h(V) 3*V*V-2*SE5*V+SE6
int main()
{
    int option, step=1, N;
    float R, Tc, Pc, T, P, V_ideal, MV_ideal, aVDW_1, bVDW_1, SE1, SE2;
    float N_ideal, aVDW_2, bVDW_2, w, V, n, SE3, SE4;
    float aVDW_P, bVDW_P, aVDW_T, bVDW_T, MV, aVDW_V, bVDW_V, SE5, SE6, SE7;
    float V1, MV1, f0, f1, d0, d1, e, n1, x0, x1, y0, y1, g0, g1, h0, h1;

    cout<<"\n"<<"SOLUTION TO VANDER WAALS EOS PROBLEM"<<"\n"<<endl;
    cout<<"Calculation Option..."<<"\n"<<endl;
    cout<<setw(13)<<"(1) "<<"Molar Volume (MV))"<<endl;
    cout<<setw(13)<<"(2) "<<"Pressure (P)"<<endl;
    cout<<setw(13)<<"(3) "<<"Temperature (T)"<<endl;
    cout<<setw(13)<<"(4) "<<"Number of Moles (n)"<<endl;
    cout<<setw(13)<<"(5) "<<"Volume (V)"<<endl;
    cout<<"\n"<<"Note: Pc = Critical Pressure, Tc = Critical Temperature,";
    cout<<"\n"<<"      and R = Molar Gas Constant"<<endl;
    cout<<"\n"<<"Instruction: To select an option, type '1', '2', '3', '4' or '5'"<<endl;
    cout<<endl<<"Choose Option: ";
    cin>>option;
    switch(option)
    {
        case 1:
            cout<<"\n\t"<<"Enter the Gas Properties:"<<endl<<endl;
            cout<<setw(20)<<"T = ";
            cin>>T;
            cout<<setw(20)<<"P = ";
            cin>>P;
            cout<<setw(20)<<"Tc = ";
            cin>>Tc;
            cout<<setw(20)<<"Pc = ";
            cin>>Pc;
            cout<<setw(20)<<"R = ";
            cin>>R;
            aVDW_1=(27*pow(R,2)*pow(Tc,2))/(64*Pc);
            bVDW_1=(R*Tc)/(8*Pc);
            SE1=bVDW_1+(R*T)/P;
            cout<<endl<<"*************************"<<endl;
            cout<<"NEWTON-RAPHSON METHOD";
            cout<<endl<<"*************************"<<endl;
            cout<<setprecision(4)<<fixed;
        /* INPUTS */
            cout<<"\t"<<"Guess value, MV_ideal = ";
            cin>>MV_ideal;
            cout<<"\t"<<"Specify Tolerable Error: ";
            cin>>e;
            cout<<"\t"<<"Maximum Iteration = ";
            cin>>N;
            cout<<"\n"<<setw(5)<<"Iteration"<<setw(7)<<"MV"<<setw(10)<<"f(MV)"<<setw(11)<<"f'(MV)
                do
                {
                    f0=f(MV_ideal);     // = f(V)
                    d0=d(MV_ideal);     // = f'(V)
                    if(d0==0.0)
                    {
                        cout<<"Mathematical Error";
                        exit(0);
                    }
```

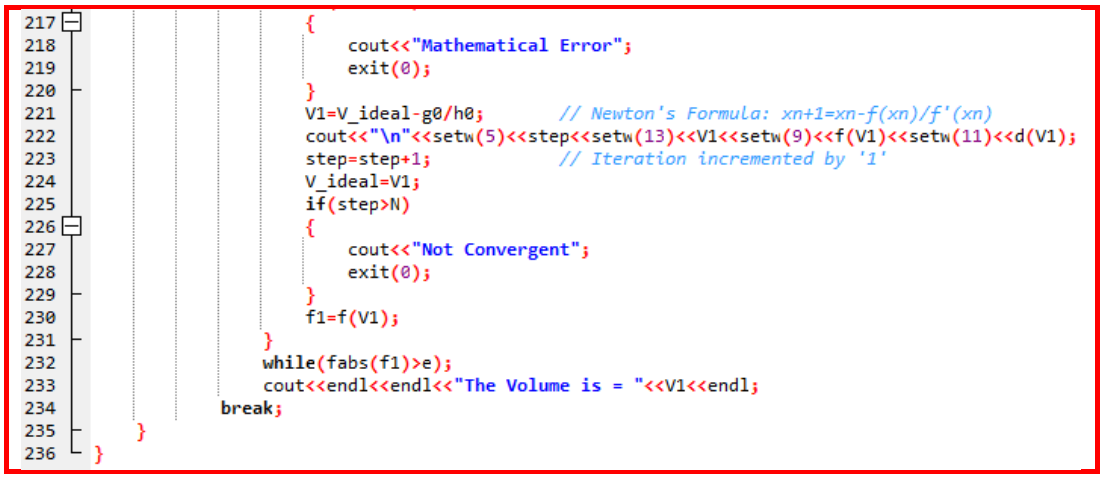
```
                    MV1=MV_ideal-f0/d0;     // Newton's Formula: xn+1=xn-f(xn)/f'(xn)
                    cout<<"\n"<<setw(5)<<step<<setw(13)<<MV1<<setw(9)<<f(MV1)<<setw(11)<<d(MV1);
                    step=step+1;            // Iteration incremented by '1'
                    MV_ideal=MV1;
                    if(step>N)
                    {
                        cout<<"Not Convergent";
                        exit(0);
                    }
                    f1=f(MV1);
                }
                while(fabs(f1)>e);
                cout<<endl<<endl<<"The Molar Volume is = "<<MV1<<endl;
            break;

        case 2:
            cout<<"\n\t"<<"Enter the Gas Properties:"<<endl<<endl;
            cout<<setw(20)<<"T = ";
            cin>>T;
            cout<<setw(20)<<"MV = ";
            cin>>MV;
            cout<<setw(20)<<"Tc = ";
            cin>>Tc;
            cout<<setw(20)<<"Pc = ";
            cin>>Pc;
            cout<<setw(20)<<"R = ";
            cin>>R;
            aVDW_P=(27*pow(R,2)*pow(Tc,2))/(64*Pc);
            bVDW_P=(R*Tc)/(8*Pc);
            P=(R*T)/(MV-bVDW_P)-aVDW_P/pow(MV,2);
                cout<<endl<<endl<<"Pressure, P = "<<P<<endl;
                cout<<endl<<"Constants: a = "<<aVDW_P<<" & b = "<<bVDW_P<<endl;
            break;

        case 3:
            cout<<"\n\t"<<"Enter the Gas Properties:"<<endl<<endl;
             cout<<setw(20)<<"MV = ";
             cin>>MV;
             cout<<setw(20)<<"P = ";
             cin>>P;
             cout<<setw(20)<<"Tc = ";
             cin>>Tc;
             cout<<setw(20)<<"Pc = ";
             cin>>Pc;
             cout<<setw(20)<<"R = ";
             cin>>R;
             aVDW_T=(27*pow(R,2)*pow(Tc,2))/(64*Pc);
             bVDW_T=(R*Tc)/(8*Pc);
             T=((P+aVDW_T/pow(MV,2))*(MV-bVDW_T))/R;
                 cout<<endl<<endl<<"Temperature, T = "<<T<<endl;
                 cout<<endl<<"Constants: a = "<<aVDW_T<<" & b = "<<bVDW_T<<endl;
             break;

         case 4:
             cout<<"\n\t"<<"Enter the Gas Properties:"<<endl<<endl;
             cout<<setw(20)<<"V = ";
             cin>>V;
             cout<<setw(20)<<"P = ";
             cin>>P;
             cout<<setw(20)<<"T = ";
             cin>>T;
             cout<<setw(20)<<"Tc = ";
             cin>>Tc;
             cout<<setw(20)<<"Pc = ";
             cin>>Pc;
             cout<<setw(20)<<"R = ";
             cin>>R;
             aVDW_2=(27*pow(R,2)*pow(Tc,2))/(64*Pc);
             bVDW_2=(R*Tc)/(8*Pc);
             SE2=V/bVDW_2;
             SE3=(P/(aVDW_2*bVDW_2))*(bVDW_2+(R*T/P))*pow(V,2);
             SE4=(P/(aVDW_2*bVDW_2))*pow(V,3);
```

```
145             cout<<endl<<"************************"<<endl;
146             cout<<"NEWTON-RAPHSON METHOD";
147             cout<<endl<<"************************"<<endl;
148             cout<<setprecision(4)<<fixed;
149         /* INPUTS */
150             cout<<"\t"<<"Guess value, N_ideal = ";
151             cin>>N_ideal;
152             cout<<"\t"<<"Specify Tolerable Error: ";
153             cin>>e;
154             cout<<"\t"<<"Maximum Iteration = ";
155             cin>>N;
156             cout<<"\n"<<setw(5)<<"Iteration"<<setw(7)<<"n"<<setw(10)<<"f(n)"<<setw(11)<<"f'(n)";
157                 do
158                 {
159                     x0=x(N_ideal);      // = f(n)
160                     y0=y(N_ideal);      // = f'(n)
161                     if(y0==0.0)
162                     {
163                         cout<<"Mathematical Error";
164                         exit(0);
165                     }
166                     n1=N_ideal-x0/y0;       // Newton's Formula: xn+1=xn-f(xn)/f'(xn)
167                     cout<<"\n"<<setw(5)<<step<<setw(13)<<n1<<setw(9)<<x(n1)<<setw(11)<<y(n1);
168                     step=step+1;            // Iteration incremented by '1'
169                     N_ideal=n1;
170                     if(step>N)
171                     {
172                         cout<<"Not Convergent";
173                         exit(0);
174                     }
175                     x1=x(n1);
176                 }
177                 while(fabs(x1)>e);
178                 cout<<endl<<endl<<"Number of moles, n = "<<n1<<endl;
179             break;
180
181     case 5:
182         cout<<"\n\t"<<"Enter the Gas Properties:"<<endl<<endl;
183         cout<<setw(20)<<"T = ";
184         cin>>T;
185         cout<<setw(20)<<"P = ";
186         cin>>P;
187         cout<<setw(20)<<"n = ";
188         cin>>n;
189         cout<<setw(20)<<"Tc = ";
190         cin>>Tc;
191         cout<<setw(20)<<"Pc = ";
192         cin>>Pc;
193         cout<<setw(20)<<"R = ";
194         cin>>R;
195         aVDW_V=(27*pow(R,2)*pow(Tc,2))/(64*Pc);
196         bVDW_V=(R*Tc)/(8*Pc);
197         SE5=n*bVDW_V+(n*R*T)/P;
198         SE6=aVDW_V*pow(n,2)/P;
199         SE7=aVDW_V*bVDW_V*pow(n,3)/P;
200         cout<<endl<<"************************"<<endl;
201         cout<<"NEWTON-RAPHSON METHOD";
202         cout<<endl<<"************************"<<endl;
203         cout<<setprecision(4)<<fixed;
204     /* INPUTS */
205         cout<<"\t"<<"Guess value, V_ideal = ";
206         cin>>V_ideal;
207         cout<<"\t"<<"Specify Tolerable Error: ";
208         cin>>e;
209         cout<<"\t"<<"Maximum Iteration = ";
210         cin>>N;
211         cout<<"\n"<<setw(5)<<"Iteration"<<setw(7)<<"V"<<setw(10)<<"f(V)"<<setw(11)<<"f'(V)";
212             do
213             {
214                 g0=g(V_ideal);      // = f(V)
215                 h0=h(V_ideal);      // = f'(V)
216                 if(h0==0.0)
```



```
217                 {
218                     cout<<"Mathematical Error";
219                     exit(0);
220                 }
221                 V1=V_ideal-g0/h0;       // Newton's Formula: xn+1=xn-f(xn)/f'(xn)
222                 cout<<"\n"<<setw(5)<<step<<setw(13)<<V1<<setw(9)<<f(V1)<<setw(11)<<d(V1);
223                 step=step+1;            // Iteration incremented by '1'
224                 V_ideal=V1;
225                 if(step>N)
226                 {
227                     cout<<"Not Convergent";
228                     exit(0);
229                 }
230                 f1=f(V1);
231             }
232             while(fabs(f1)>e);
233             cout<<endl<<endl<<"The Volume is = "<<V1<<endl;
234         break;
235     }
236 }
```

Figure 7. Multiple Option C++ Program to Execute VDW EOS for P, V, T, n, and V_m Parameters

Alphabet symbols, 'f', 'd'; 'x', 'y'; and 'g', 'h' in the C++ source code, represents function and derivative of the molar volume, number of moles, and volume respectively. SE, standing for short expressions, was used to represent the coefficients of the variables to be computed in the functions. Table 5 depicts all the short expressions and their values (coefficients) as used earlier in FORTRAN, C++ and subsequent programming (in Python Java, MATLAB and R).

Table 5. Values of the Nonlinear Functions' Coefficients

Coefficient	Expression	Value
SE1	$b + \frac{RT}{P}$	7.14758560346
SE2	$\frac{V}{b}$	187.30993802315
SE3	$\frac{P}{ab}\left(b + \frac{RT}{P}\right)V^2$	9019.89991305033
SE4	$\frac{PV^3}{ab}$	8833.65417278678
SE5	$nb + \frac{nRT}{P}$	7.14758560346
SE6	$\frac{an^2}{P}$	1.03900229550
SE7	$\frac{an^3b}{P}$	0.03882877836

Users of the C++ code, when run, are asked to choose the gas parameter to compute, as specified in the 'switch' programming loops. It further requests

the unknown parameters of the gas, initial guess, maximum numberofiterationsand tolerable error as shown in Figure 8.Z

```
SOLUTION TO VANDER WAALS EOS PROBLEM

Calculation Option...

        (1) Molar Volume (MV))
        (2) Pressure (P)
        (3) Temperature (T)
        (4) Number of Moles (n)
        (5) Volume (V)

Note: Pc = Critical Pressure, Tc = Critical Temperature,
      and R = Molar Gas Constant

Instruction: To select an option, type '1', '2', '3', '4' or '5'

Choose Option: 4

        Enter the Gas Properties:

               V = 7
               P = 4.0394
               T = 350
              Tc = 405.5
              Pc = 111.3
               R = 0.08206

*************************
NEWTON-RAPHSON METHOD
*************************
        Guess value, N_ideal = 0.9845
        Specify Tolerable Error: 0.001
        Maximum Iteration = 10

Iteration      n      f(n)      f'(n)
    1       1.0000  -0.0439  8648.2783
    2       1.0000   0.0000  8648.2764

Number of moles, n = 1.0000

--------------------------------
Process exited after 54.25 seconds with return value 0
```

Figure 8. Number of Moles Determined After Choosing a Calculation Option '4' in C++

Number of moles, its function and derivative values per iterations are tabulated (Figure 8) according to the instruction passed to C++ through the codes. When the root is found, calculation stops and the root is declared. To choose another option, users must return to the code environment and re-run the program. The molar volume, as shown in Figure 9 was obtained in just 2 iterations.

```
SOLUTION TO VANDER WAALS EOS PROBLEM

Calculation Option...

        (1) Molar Volume (MV))
        (2) Pressure (P)
        (3) Temperature (T)
        (4) Number of Moles (n)
        (5) Volume (V)

Note: Pc = Critical Pressure, Tc = Critical Temperature,
      and R = Molar Gas Constant

Instruction: To select an option, type '1', '2', '3', '4' or '5'

Choose Option: 1

        Enter the Gas Properties:

              T = 350
              P = 4.0394
             Tc = 405.5
             Pc = 111.3
              R = 0.08206

*************************
NEWTON-RAPHSON METHOD
*************************
        Guess value, MV_ideal = 7.1092
        Specify Tolerable Error: 0.001
        Maximum Iteration = 10

Iteration     MV     f(MV)     f'(MV)
    1       7.0032   0.1580    48.0625
    2       7.0000   0.0002    47.9715

The Molar Volume is = 7.0000

--------------------------------
Process exited after 46.32 seconds with return value 0
Press any key to continue . . .
```

Figure 9. Molar Volume Determined After Choosing a Calculation Option '1' in C++

A situation in which the root is not arrived at (not convergent), the value in the last iteration could be the root or better still, it is advised to increase the number of iterations. A summary translation of the C++ source code in Figure 7 is depicted in block diagram in Figure 10.

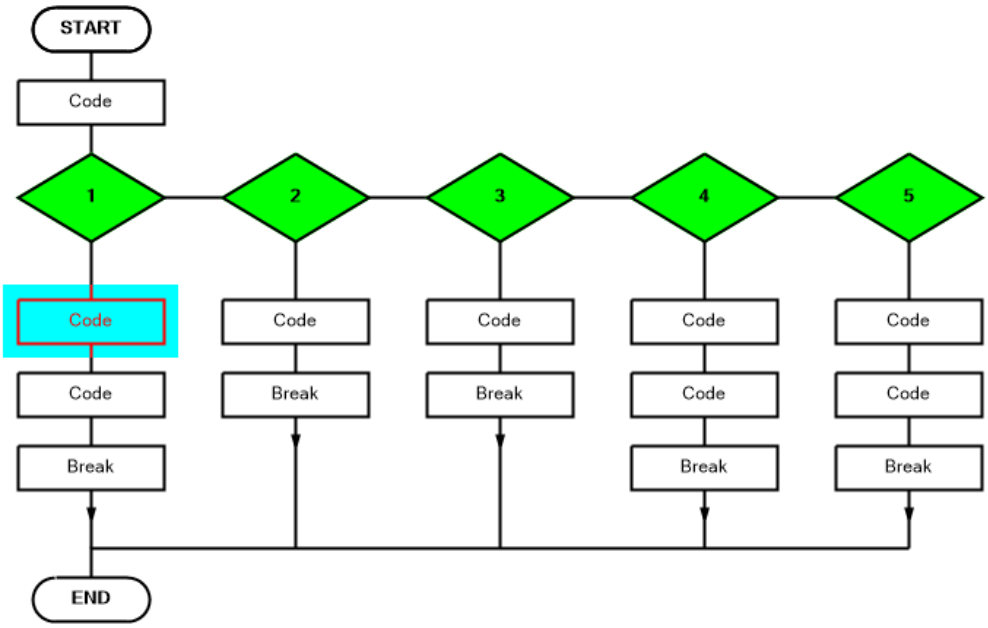


Figure 10. Block Summary Flowchart of the C++ Syntax for Solving Newton's Method

Once an option is selected (see rhombus shape in Figure 10), the other options are ignored. C++ would only be concern with the execution of the source code within the option loop, which is put to a halt by the 'break' statement, also shown in Figure 7. Within the selected option, series of steps followed to execute Newton's algorithm is similar to the general flowchart illustrated in Figure 11.

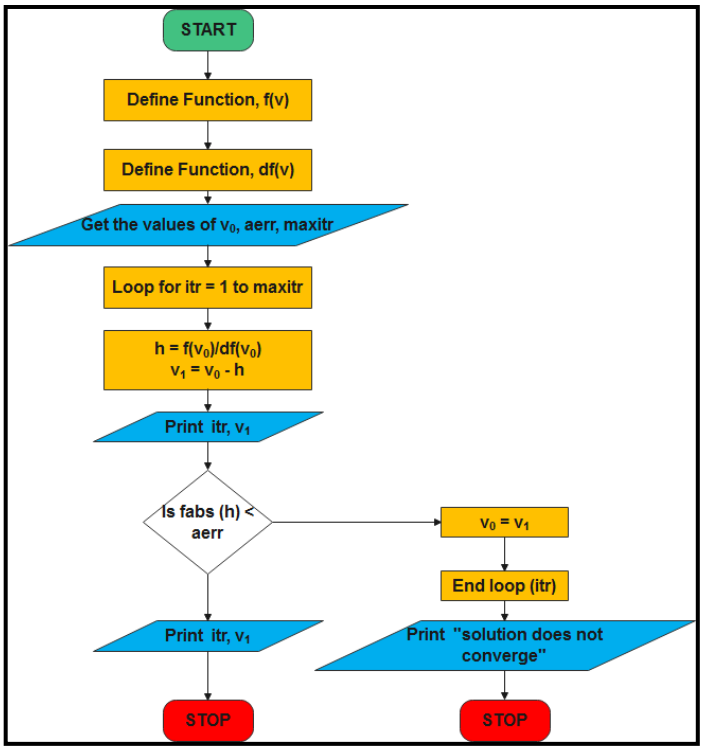


Figure 11. Newton-Raphson Method Flowchart Showing the Iterative Steps

Not only C++, the algorithm and flowchart in Figure 11 can be used to write source code for Newton's method in any high-level programming language. Note that fabs( ) function in C++ returns the absolute value of the argument. Other abbreviations like 'itr' = iteration or 'step', 'maxit' = maximum iteration or 'N', and 'aerr' = absolute error.

PYTHON Algorithm for Newton's Method

SEs given in Table 5 helped in defining the coefficients in the functions to estimate the volume in the PYTHON program, shown in Figure 12.

```
# Defining Volume Function
def f(v):
    return v**3-7.147585603*(v**2)+1.039002295*v-0.038828778
# Defining Derivative Volume Function
def g(v):
    return 3*(v**2)-2*7.147585603*v+1.039002295
# Implementing Newton Raphson Method
def newtonRaphson(v0,e,N):
    print('\n\n*** VAN DER WAALS EOS SOLUTION WITH NEWTON RAPHSON METHOD ***')
    print('\n\n*** VOLUME CALCULATION ***')
    step = 1
    flag = 1
    condition = True
    while condition:
        if g(v0) == 0.0:
            print('Divide by zero error!')
            break
        v1 = v0 - f(v0)/g(v0)
        print('Iteration-%d, v1 = %0.6f and f(v1) = %0.6f' % (step, v1, f(v1)))
        v0 = v1
        step = step + 1
        if step > N:
            flag = 0
            break
        condition = abs(f(v1)) > e
    if flag==1:
        print('\nRequired root is V = %0.8f' % v1)
    else:
        print('\nNot Convergent.')
# Input Section
v0 = input('Enter Initial Volume Guess: ')
e = input('Tolerable Error: ')
N = input('Maximum Step: ')
# Converting n0 and e to float
v0 = float(v0)
e = float(e)
# Converting N to integer
N = int(N)

# Starting Newton Raphson Method
newtonRaphson(v0,e,N)
```

Figure 12. PYTHON Newton's Method Volume Iteration Source Code

The volume and its function after every iteration were kept in 6 decimal places (d.p.), defined as "%0.6f" in PYTHON while the final root is kept at 8 d.p., as shown in Figure 13.

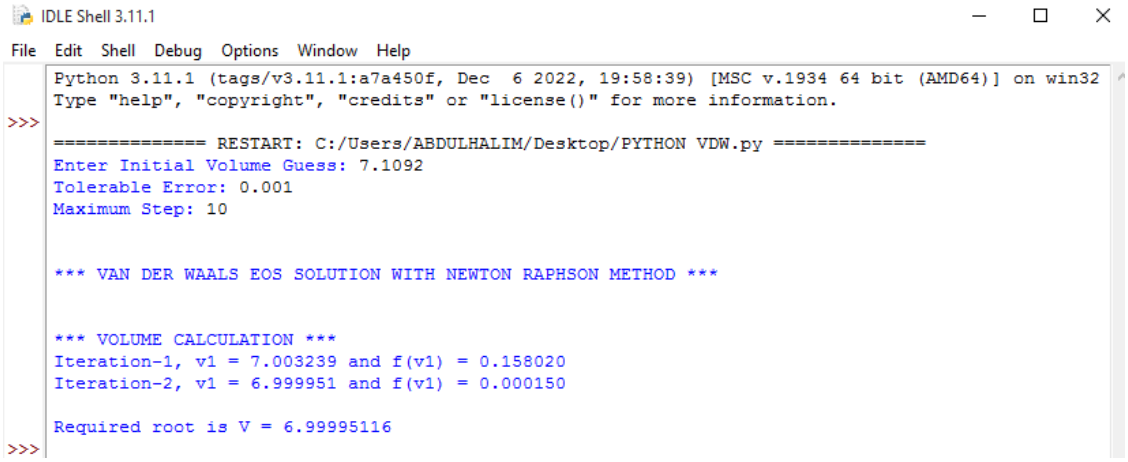


Figure 13. PYTHON Execution of VDW EOS Root – Volume Computation

Just like the C++ program written, the user is allowed to specify the initial guess, tolerable error and maximum step. However, the written FORTRAN source code is limited to requesting the initial value only without any other parameter, and less flexible compared to Python source. Furthermore, the PYTHON source code can only be used for a particular problem – which means the user had to calculate and replace the SEs (coefficients) for every problem given, and makes the C++ more user-friendly in this case. Number of moles can be calculated for Problem 1 using the same Python coding technique (check Figure 14).

```
# Defining the Mole Function
def f(n):
    return n**3-187.309938*(n**2)+9019.899913*n-8833.654173
# Defining Derivative Mole Function
def g(n):
    return 3*(n**2)-2*187.309938*n+9019.899913
# Implementing Newton Raphson Method
def newtonRaphson(n0,e,N):
    print('\n\n*** VAN DER WAALS EOS SOLUTION WITH NEWTON RAPHSON METHOD ***')
    print('\n\n*** NUMBER OF MOLES CALCULATION ***')
    step = 1
    flag = 1
    condition = True
    while condition:
        if g(n0) == 0.0:
            print('Divide by zero error!')
            break
        n1 = n0 - f(n0)/g(n0)
        print('Iteration-%d, n1 = %0.6f and f(n1) = %0.6f' % (step, n1, f(n1)))
        n0 = n1
        step = step + 1
        if step > N:
            flag = 0
            break
        condition = abs(f(n1)) > e
    if flag==1:
        print('\nRequired root is n = %0.8f' % n1)
    else:
        print('\nNot Convergent.')
# Input Section
n0 = input('Enter Initial Number of Moles Guessed: ')
e = input('Tolerable Error: ')
N = input('Maximum Step: ')
# Converting n0 and e to float
n0 = float(n0)
e = float(e)
# Converting N to integer
N = int(N)

# Starting Newton Raphson Method
newtonRaphson(n0,e,N)
```

Figure 14. Number of Moles Iteration Source Code Implementation of Newton's Method in PYTHON

Power of a variable can be expressed using the symbol ** in PYTHON and FORTRAN as demonstrated in Figure 5 and 14. With this, relevant variables were replaced to prompt the program to calculate the number of moles, as shown in Figure 15.

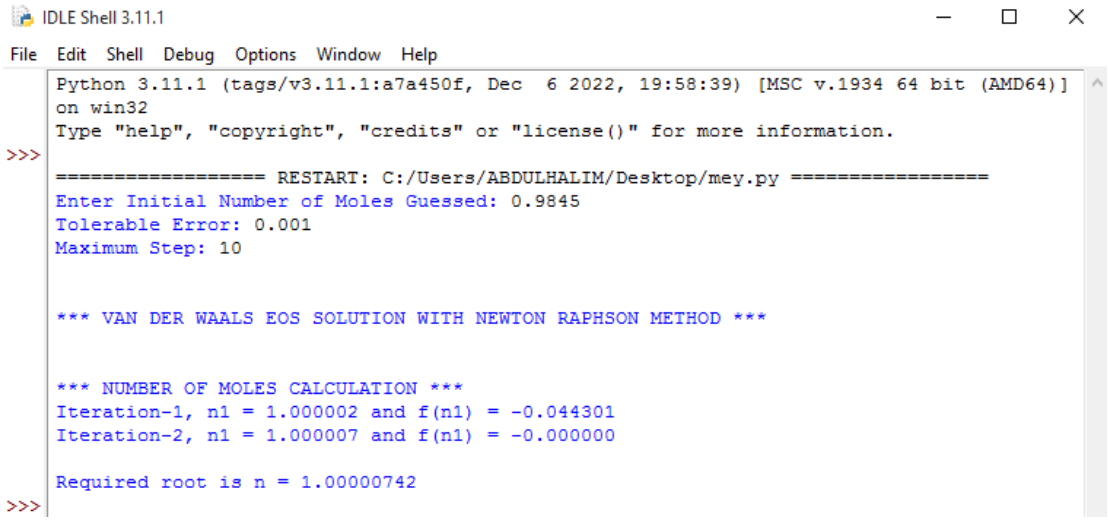


Figure 15. Number of Moles as Calculated Using PYTHON Source Code

Figure 16 depicts the PYTHON source code for approximating the roots of the VDW molar volume function.

```
# Defining Molar Volume Function
def f(mv):
    return mv**3-7.147585603*(mv**2)+1.039002295*mv-0.038828778
# Defining Derivative Molar Volume Function
def g(mv):
    return 3*mv**2-2*7.147585603*mv+1.039002295
# Implementing Newton Raphson Method
def newtonRaphson(mv0,e,N):
    print('\n\n*** VAN DER WAALS EOS SOLUTION WITH NEWTON RAPHSON METHOD ***')
    print('\n\n*** MOLAR VOLUME CALCULATION ***')
    step = 1
    flag = 1
    condition = True
    while condition:
        if g(mv0) == 0.0:
            print('Divide by zero error!')
            break
        mv1 = mv0 - f(mv0)/g(mv0)
        print('Iteration-%d, mv1 = %0.6f and f(mv1) = %0.6f' % (step, mv1, f(mv1)))
        mv0 = mv1
        step = step + 1
        if step > N:
            flag = 0
            break
        condition = abs(f(mv1)) > e
    if flag==1:
        print('\nRequired root is MV = %0.8f' % mv1)
    else:
        print('\nNot Convergent.')
# Input Section
mv0 = input('Enter Initial Molar Volume Guessed: ')
e = input('Tolerable Error: ')
N = input('Maximum Step: ')
# Converting mv0 and e to float
mv0 = float(mv0)
e = float(e)
# Converting N to integer
N = int(N)

# Starting Newton Raphson Method
newtonRaphson(mv0,e,N)
```

Figure 16. Newton's Method PYTHON Syntax for Molar Volume Iteration

The calculated value of the molar volume after running codes in Figure 16 is shown in Figure 17.

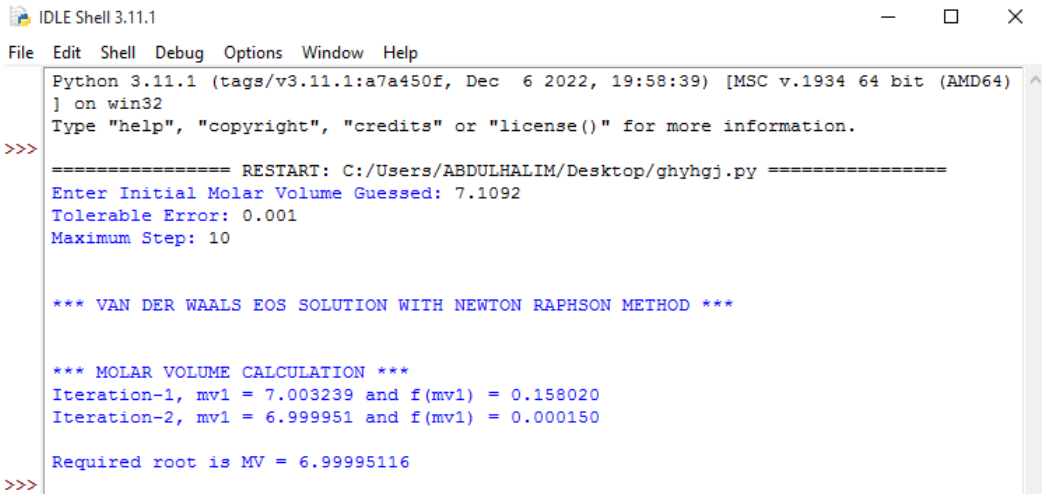


Figure 17. Results Window for Molar Volume Approximation Using Python

JAVA Execution of VDW EOS using Newton's Method

Unlike in former programs written in this work, the initial guess is declared within the program, while only the result is displayed after running the Java codes, as shown in Figure 18 and 19.

Execute | Beautify | Share Source Code Help

```java
// Java Program for Implementing Newton-Raphson Method to Solve
// Van der Waals (VDW) Equation of State (EOS)
public class VDW {
    static final double EPSILON = 0.001;
    // Volume Function: v*v*v-7.147585603*v*v+1.039002295*v-0.038828778
    static double func(double v)
    {
        return v*v*v-7.147585603*v*v+1.039002295*v-0.038828778;
    }
    // Volume Function Derivative: 3*v*v-2*7.147585603*v+1.039002295
    static double derivFunc(double v)
    {
        return 3*v*v-2*7.147585603*v+1.039002295;
    }
    // Function to find the root
    static void newtonRaphson(double v)
    {
        double h = func(v) / derivFunc(v);
        while (Math.abs(h) >= EPSILON)
        {
            h = func(v) / derivFunc(v);
    // Newtons Method Equivalent: v(i+1) = v(i) - f(v) / f'(v)
            v = v - h;
```

```
24          }
25          System.out.print("Value of the"
26                  + " root is V = "
27                  + Math.round(v * 100.0) / 100.0);
28      }
29      // Driver Code
30      public static void main (String[] args)
31      {
32          // Assumed Initial Values
33          double v0 = 7.1092;
34          newtonRaphson(v0);
35      }
36  }
37  // Code Amended by Engr. Abdulhalim Musa Abubakar
```

Figure 18. Volume Implementation Using Java Source Code

Result

CPU Time: 0.19 sec(s), Memory: 33596 kilobyte(s)

```
Value of the root is V = 7.0
```

compiled and executed in 1.276 sec(s)

Figure 19. Volume Result After Running the Java Program

In Figure 20, Java does not allow the use of 'mv' (two letters), to represent molar volume in the function as used in PYTHON, C++ and FORTRAN. Instead 'd', a single alphabet was made to stand for molar volume.

Execute | Beautify | Share Source Code Help

```java
// Java Program for Implementing Newton-Raphson Method to Solve
// Van der Waals (VDW) Equation of State (EOS)
public class VDW {
    static final double EPSILON = 0.001;
    // Nolar Volume Function: d*d*d-7.147585603*d*d+1.039002295*d-0
        .038828778
    static double func(double d)
    {
        return d*d*d-7.147585603*d*d+1.039002295*d-0.038828778;
    }
    // Molar Volume Function Derivative: 3*d*d-2*7.147585603*d+1.039002295
    static double derivFunc(double d)
    {
        return 3*d*d-2*7.147585603*d+1.039002295;
    }
    // Function to find the root
    static void newtonRaphson(double d)
    {
        double h = func(d) / derivFunc(d);
        while (Math.abs(h) >= EPSILON)
        {
            h = func(d) / derivFunc(d);
    // Newtons Method Equivalent: d(i+1) = d(i) - f(d) / f'(d)
            d = d - h;
        }
        System.out.print("Value of the"
                + " root is MV = "
                + Math.round(d * 100.0) / 100.0);
    }
    // Driver Code
    public static void main (String[] args)
    {
        // Assumed Initial Values
        double d0 = 7.1092;
        newtonRaphson(d0);
    }
}
// Code Amended by Engr. Abdulhalim Musa Abubakar
```

Figure 20. Molar Volume Implementation Using Java Source Code

As earlier stated, the molar volume and the volume are same, since n = 1, as shown in Figure 21.

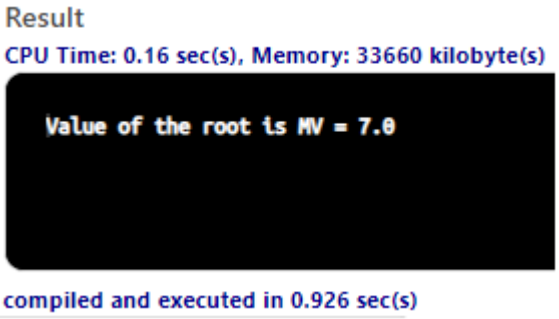


Figure 21. Molar Volume Result After Running the Java Program

Just like Figure 18 and 20, in Figure 22, 'h' was made to stand for the ratio of the function and its derivative

Execute | Beautify | Share | Source Code | Help

```
1  // Java Program for Implementing Newton-Raphson Method to Solve
2  // Van der Waals (VDW) Equation of State (EOS)
3  public class VDW {
4      static final double EPSILON = 0.001;
5      // Number of Moles Function: n*n*n-187.309938*n*n+9019.899913*n-8833
           .654173
6      static double func(double n)
7      {
8          return n*n*n-187.309938*n*n+9019.899913*n-8833.654173;
9      }
10     // Number of Moles Derivative Function: 3*n*n-2*187.309938*n+9019.899913
11     static double derivFunc(double n)
12     {
13         return 3*n*n-2*187.309938*n+9019.899913;
14     }
15     // Function to find the root
16     static void newtonRaphson(double n)
17     {
18         double h = func(n) / derivFunc(n);
19         while (Math.abs(h) >= EPSILON)
20         {
21             h = func(n) / derivFunc(n);
22     // Newtons Method Equivalent: n(i+1) = n(i) - f(n) / f'(n)
23             n = n - h;
```

```
24          }
25          System.out.print("Value of the"
26                  + " root is n = "
27                  + Math.round(n * 100.0) / 100.0);
28      }
29      // Driver Code
30      public static void main (String[] args)
31      {
32          // Assumed Initial Values
33          double n0 = 0.9845;
34          newtonRaphson(n0);
35      }
36  }
37  // Code Amended by Engr. Abdulhalim Musa Abubakar
```

Figure 22. Number of Moles Implementation Using Java Source Code

As shown in Figure 23 and unlike all the other programs, the Java output window gives an approximate value of the roots.

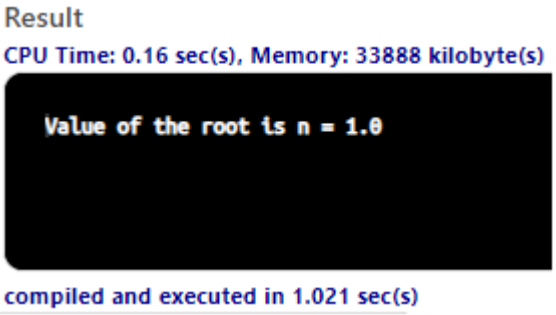


Figure 23. Number of Moles Result After Running the Java Program

The flowchart produced from Java source code in Figure 22 is as shown in Figure 24.

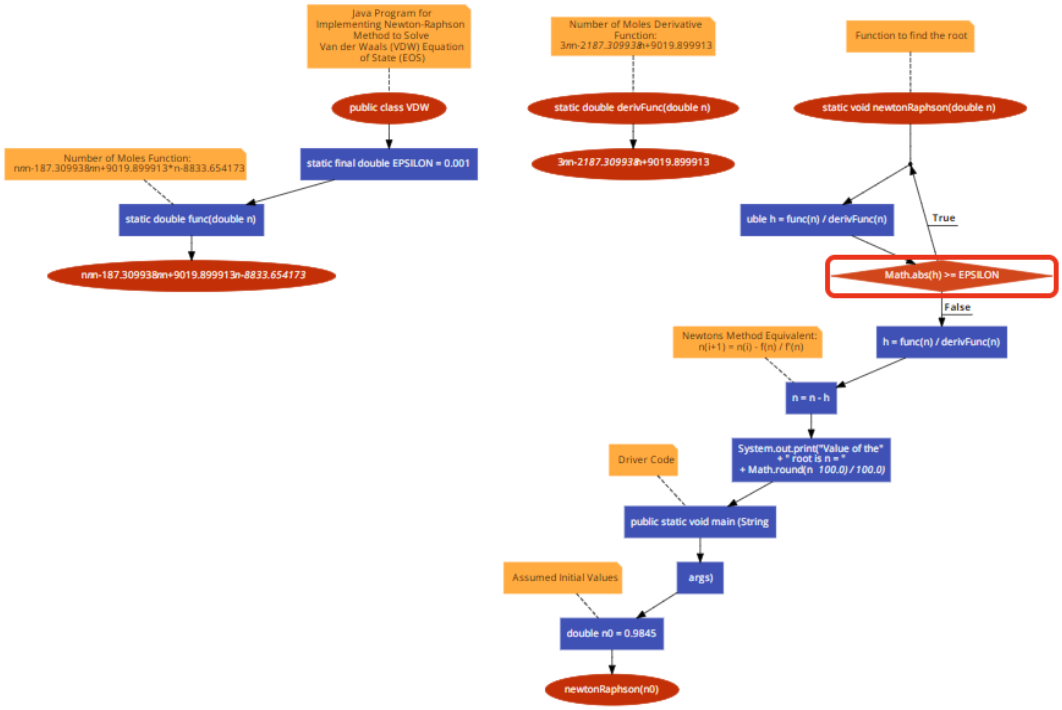


Figure 24. Java Flowchart from Newton-Raphson Method

Hence, flowcharts for all the written Java source codes in this work can represented same way as in Figure 24.

MATLAB Program Solution for Newton's Method

In a single window, MATLAB produces the results of the iteration involves in computing volume and number of moles of NH_3 in short and simple lines of codes shown in Figure 25 and 26.

vdw.m

```
clc
clear
fun = @(v) v*v*v-7.147585603*v*v+1.039002295*v-0.038828778;
roots = vdwnewton(fun, 6, 8)

function output = vdwnewton(fun,a,p)
syms v;
z = fun(v);
derZ = diff(z);
out = zeros(1,p+1);
out(1) = a;

for idx = 1 : p
    numeratorZ = subs(z,v,out(idx));
    denominatorZ = subs(derZ,v,out(idx));
    out(idx+1) = out(idx) - double(numeratorZ)/double(denominatorZ);
end
output = out;
end
```

Command Window

New to MATLAB? See resources for Getting Started.

```
    6.0000    7.5093    7.0613    7.0010    6.9999    6.9999    6.9999    6.9999
```

Figure 25. Volume Calculation in MATLAB in 8 Iterations

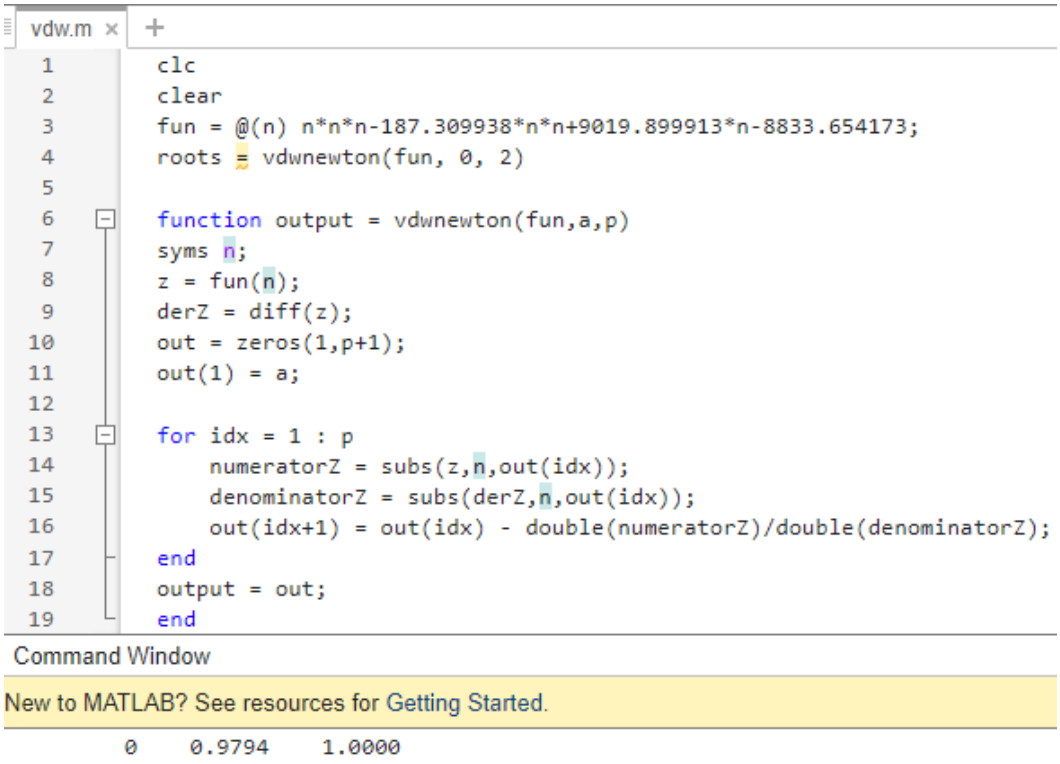


Figure 26. Number of Moles Calculation in MATLAB in 2 Iterations

In line 4 of the two MATLAB source codes, the initial guess is first specified, separated by comma and then the number of iterations. So, '6, 8' in Figure 25 means V_0 = 6, stopping after 8 iterations; while '0, 2' in Figure 26 means n_0 = 0, stopping after 2 iterations.

R Programming Solution to VDW EOS

With the right programming, R codes for a particular problem solution implementation wouldn't be lengthy, according to Mailud (2022). Capabilities of the R language also bears resemblance with the functions of SPSS, Stata and SAS softwares for big data analytics (Ozgur et al., 2017).

Figure 27 and 28 presents the R source codes for solving volume and number of moles in Problem 1.

```
> vdwnewton <- function(f, df, v0, tol=1e-10, maxiter=100)
+ {
+   # f is the function of volume from Van der Waals EOS
+   # df is the derivative of f
+   # v0 is the initial guess for the root (from ideal gas law)
+   # tol is the tolerance for the error
+   # maxiter is the maximum number of iterations
+
+   for (i in 1:maxiter)
+   {
+     v1 <- v0 - f(v0)/df(v0)             # Update v using Newton's Method
+     if (abs(v1 - v0) < tol) return(v1)  # Check for Convergence
+     v0 <- v1                            # Update v0 for the next iteration
+   }
+   return(v1)                            # Return the result if convergence was not achieved
+ }
> f <- function(v) v^3-7.147585603*(v^2)+1.039002295*v-0.038828778
> df <- function(v) 3*v^2-2*7.147585603*v+1.039002295
> v0 <- 7.1092
> result <- vdwnewton(f, df, v0)
> print(result)
[1] 6.999948
```

Figure 27. R Source Code to Implement Volume Calculation from VDW EOS

```
> vdwnewton <- function(f, df, n0, tol=1e-10, maxiter=100)
+ {
+   # f is the function of Number of moles (n) from Van der Waals EOS
+   # df is the derivative of f
+   # n0 is the initial guess for the root (from ideal gas law)
+   # tol is the tolerance for the error
+   # maxiter is the maximum number of iterations
+
+   for (i in 1:maxiter)
+   {
+     n1 <- n0 - f(n0)/df(n0)             # Update n using Newton's Method
+     if (abs(n1 - n0) < tol) return(n1)  # Check for Convergence
+     n0 <- n1                            # Update n0 for the next iteration
+   }
+   return(n1)                            # Return the result if convergence was not achieved
+ }
> f <- function(n) n^3-187.309938*n^2+9019.899913*n-8833.654173
> df <- function(n) 3*n^2-2*187.309938*n+9019.899913
> n0 <- 0.9845
> result <- vdwnewton(f, df, n0)
> print(result)
[1] 1.000007
```

Figure 28. R Source Code to Implement Number of Moles Computation from VDW EOS

Both MATLAB and R source codes are not flexible because they cannot be re-run to solve other problems without changing the codes. All inputs are specified within the lines of codes to display only the result of the iteration shown in 'blue' color.

CONCLUSION

This work strives to revive interest in the use of programming languages in the related field of study by showcasing the capability of various programming languages in solving numerical analysis problems related to chemical engineering. Apart from the one used, several others, namely, JavaScript, Ruby, Swift, PHP, SQL, Dart, Objective-C, Julia, Visual Basic .NET, C#, and Go can be tested. The C++ source code written here is designed to solve, not only Problem 1, but any other problems encountered in the field, unlike the 5 other programs written, without changing the codes. Users can go further to write a simple general program using Java, Python, FORTRAN, R and MATLAB programming softwares capable of solving all relevant parameters in VDW EOS. This would be

very tasking, as simpler programming corresponds to fewer mistakes and better performance – but would solve limitations not seen in the written C++ source code for solving related problems. Further studies should look at how to convert the written codes to software applications for installation and use in personal computers.

ACKNOWLEDGEMENT

We thank all anonymous reviewers for their efforts and relevant comments. All authors equally contributed to this work.

REFERENCES

Abubakar, A. M., Elshahhat, A., Ndaba, S., Mobolaji, A. T., Thiagarajan, B., & Mansour, E. M. (2023). Chemical engineering numerical analysis with R: Peng–Robinson equation of state. Journal of Data Science and Intelligent Systems (JDSIS), 00(00), 1–20. https://doi.org/10.47852/bonviewJDSIS3202665

Abubakar, A. M., Francis, O. C., Sarkinbaka, Z. M., & Yahaya, M. S. (2021). Simplex C++ syntax for solving chemical engineering cost optimization problems. Research Inventy: International Journal of Engineering And Science, 11(7), 39–47. https://doi.org/10.5281/zenodo.5146856

Abubakar, A. M., & Mustapha, A. A. (2021). Newton's method cubic equation of state C++ source code for iterative volume computation. SSRG International Journal of Recent Engineering Science, 8(3), 12–22. https://doi.org/10.14445/23497157/IJRES-V8I3P103

Adams, M. D. (2021). Lecture slides for programming in C++ [The C++ language, libraries, tools, and other topics] (p. 2887). Department of Electrical and Computer Engineering: University of Victoria. http://www.ece.uvic.ca/~mdadams/cppbook

Ahuja, P. (2019). Introduction to numerical methods in chemical engineering (2nd ed.). PHI Learning Private Limited (Delhi-110092).

Al-Malah, K. I. M. (2014). MATLAB-Numerical methods with chemical engineering applications. McGraw-Hill Education: Cenveo Publisher Services.

Albrecht, J. (2021). Step into the digital age with Python (p. 47). American Institute of Chemical Engineers (AIChE). aiche.org/cep

Allen, M., & Vahid, F. (2020). Teaching Coral before C++ in a CS1 course. ASEE'S Virtual Conference (SEEVC)-Paper ID 29886, 1–15.

Andika, R., & Putra, Z. A. (2022). Teaching programming to chemical engineering students. ASEAN Journal of Science and Engineering Education, 2(1), 51–60.

Asadi, F. (2022). MATLAB programming. In Applied Numerical Analysis with MATLAB/Simulink (pp. 257–293). SLEST Book Series: Springer Nature Switzerland AG. https://doi.org/10.1007/978-3-031-19366-8_10

Ateeq, M., Habib, H., Umer, A., & Rehman, M. U. I. (2014). C++ or Python? Which one to begin with: A learner's perspective. 2014 International Conference on Teaching and Learning in Computing and Engineering, 64–69. https://doi.org/10.1109/LaTiCE.2014.20

Bakari, H. R., Adegoke, T. M., & Yahya, A. M. (2016). Application of Newton Raphson method to non-linear models. International Journal of Mathematics and Statistics Studies, 4(4), 21–31. www.eajournals.org

Balaji, G. V, & Seader, J. D. (1995). Application of interval Newton's method to chemical engineering problems. Reliable Computing, 1, 215–225. https://doi.org/10.1007/BF02385253

Bamdad, F. (2004). Solution of cubic equations by iteration methods on a pocket calculator. Journal of Chemical Education, 84(5), 758–761. https://doi.org/10.1021/ed081p758

Bandele, S. O., & Adekunle, A. S. (2015). Development of C++ application program for solving quadratic equation in elementary school in Nigeria. Journal of Education and Practice, 6(28), 70–77. www.iiste.org

Baptista, L. (2021). Using Python and Google Colab to teach physical chemistry during pandemic (pp. 1–11). Universidade do Estado do Rio de Janeiro.

Bect, D. A. C., Carothers, J. M., Subramanian, V. R., & Pfaendtner, J. (2016). Data science: Accelerating innovation and discovery in chemical engineering. AlChE Journal, 62(5), 1402–1416. https://doi.org/10.1002/aic.15192

Bell, I. H., Deiters, U. K., & Leal, A. M. M. (2022). Implementing an equation of state without derivatives: teqp (T. Loew, P. Walker, J. Young, A. Jaeger, & D. Zhu (eds.); pp. 1–31).

Bergmann, S. D. (2021). Computer science principles with C++. Open Educational Resources, 26, 253. https://doi.org/10.31986/issn.2689-0690_rdw.oer.1025

Bishop, J., & Bishop, N. (2000). Object-orientation in Java for scientific programmers. ACM SIGCSE Bulletin, 1–7. https://doi.org/10.1145/330908.331885

Blagoev, I. (2018). Using R programming language for processing of large data sets. In R. D. Andreev (Ed.), International Conference on Big Data, Knowledge and Control Systems Engineering (BdKCSE'2018) (pp. 91–97). Institute of Information and Communication Technologies (IICT) of the Bulgarian Academy of Sciences.

Blount, B., & Chatterjee, S. (1999). An evaluation of Java for numerical computing. Scientific Programming, 7, 97–110.

Boisvert, R. F., Moreira, J., Philippsen, M., & Pozo, R. (2001). Java and numerical computing (pp. 1–11). www.mcours.com

Bose, S. K. (2019). Numerical methods of mathematics implemented in Fortran (P. V Subrahmanyam, Y. P. Chaubey, J. Cuellar, J. Matkowski, T. Parthasarathy, M. D. Sikiric, & B. D. Sharma (eds.)). Springer Nature Singapore Pte Ltd. https://doi.org/10.1007/978-981-13-7114-1

Boynton, W. P., & Bramley, A. (1922). A modification of van der Waals' equation (Vol. 20, Issue 1).

Braun, W. J. (2021). A first course in statistical programming with R (2nd ed.). Cambridge University Press.

Chan, B. K. C. (2018). Data analysis using R programming. In Biostatistics for Human Genetic Epidemiology (No. 1082; Advances in Experimental Medicine and Biology (AEMB)). Springer, Cham. https://doi.org/10.1007/978-3-319-93791-5_2

David, C. W. (2015). The van der Waals equation as a cubic. Chemistry Education Materials, 88, 1–7. https://opencommons.uconn.edu/chem_educ/88

de Sousa, V. M., de Moura, F. P., de Sousa, M. P., Lacerda, R. F., & Passos, F. G. O. (2021). Hidrouff density calculator: An open-source software for predicting density of complex mixtures using Peng-Robinson equation of state (PREOS). https://doi.org/10.5281/zenodo.5719756

Diaz-Bejarano, E., Francia, V., & Colletti, F. (2019). Data analysis for chemical engineers: Introduction to R. In Introduction to Software for Chemical Engineers (2nd ed., pp. 221–250). Institute of Mechanical, Process & Energy Engineering, School of Engineering & Physical Sciences. https://doi.org/10.1201/9780429451010-7

Djamila, H. (2017). Excel spreadsheet in teaching numerical methods. IOP Conference Series: Journal of Physics (ICoAIMS), 890(012093), 1–7. https://doi.org/10.1088/1742-6596/890/1/012093

Evans, R. (2014). R programming (pp. 1–82). Michaelmas2014. http://www.stats.ox.ac.uk/~evans/teaching.htm

Farrell, J. (2022). Java programming (10th ed.). Cengage.

Finkel, D., Hooker, C., Salvidio, S., & Sullivan, M. (1994). Teaching C++ to high school students (pp. 286–289). Association of Computing Machinery (ACM).

Frank, W. F. X., & GroBmann, D. (2006). Remarks on van der Waals' equation, the cohesion pressure of a gas, and the role of intermolecular forces (pp. 1–22).

Garces, J. (2015). Comments on the repulsive term of van der Waals equation of state (pp. 1–9).

Garcia-Rodriguez, M., Anel, J. A., Foujols, M.-A., & Rodeiro, J. (2016). FortranAnalyser: A software tool to assess Fortran code quality. IEEE Access, 4, 1–4. https://doi.org/10.1109/ACCESS.2017.DOI

Gor, G. (2021). CHE 490-HM2: Special topic-Python programming for chemical engineers. Chemical and Materials Engineering Syllabi, 161, 1–7. https://digitalcommons.njit.edu/cme-syllabi/161

Hart, W., & Laird, C. D. (2014). Rethinking the C++/Python boundary in modeling and optimization tools. Sandia National Laboratories.

Hawick, K. A. (2011). Engineering domain-specific languages and automatic code generation for computational simulations of complex systems. https://doi.org/10.2316/p.2011.758-046

Hernandez, E. M., & Martin, M. (2019). Python for chemical engineering. In Introduction to Software for Chemical Engineers (2nd ed., pp. 185–220). CRC Press.

Hinsen, K. (2013). A glimpse of the future of scientific programming. In K. Hinsen & K. Laufer (Eds.), Computing in Science and Engineering (Vol. 15, Issue 1, pp. 84–88). IEEE CS and AIP. https://doi.org/10.1109/MCSE.2013.1

Hossain, E. (2022). Numerical methods in MATLAB. In MATLAB and Simulink Crash Course for Engineers (Vol. 1965, pp. 139–149). Springer. https://doi.org/10.1007/978-3-030-89762-8_7

Jarvinen, H.-M., & Ala-Mutka, K. (2004). Supporting students in C++ programming courses with automatic program style assessment. Journal of Information Technology Education, 3, 245–263.

Johnson, S. R., Prokopenko, A., & Evans, K. J. (2019). Automated Fortran-C++ bindings for large-scale scientific applications.

Johnston, D. C. (2014). Advances in thermodynamics of the van der Waals fluid. In IOP Concise Physics (p. 67). Morgan & Claypool Publishers. https://doi.org/10.88/978-1-627-05532-1ch1

Kadam, G. (2013). C++ programs for chemical engineering (p. 42). TKIET, Warananagar.

Kamarudin, S., Punzalan, C., Derahman, M. N., Mohd, S. M., Jan, N. M., & Wahab, A. N. A. (2022). The student's self perception on learning C++ programming via the cryptography project: A case study. International Journal of Academic Research in Progressive Education and Development, 11(2), 1534–1544. https://doi.org/10.6007/IJARPED/v11-i2/14031

Kapuno, R. R. A. (2008). Programming for chemical engineers using C, C++, and MATLAB (1st ed.). Jones & Bartlett Learning, LLC.

Kedward, L., Aradi, B., Certik, O., Curcic, M., Ehlert, S., Engel, P., Goswami, R., Hirsch, M., Lozada-Blanco, A., Magnin, V., Markus, A., Pagome, E., Pribee, I., Richardson, B., Snyder, H., Urban, J., & Vandenplas, J. (2022). The state of Fortran. In K. Hinsen (Ed.), Computing in Science and Engineering (pp. 1–11). IEEE Computer Society. https://doi.org/10.1109/MCSE.2022.3159862

Kiusalaas, J. (2013). Numerical methods in engineering with Python3. Cambridge University Press: The Pennsylvania State University. www.cambridge.org/9781107033856

Kontogeorgis, G., Privat, R., & Jaubert, J.-N. (2019). Taking another look at the van der Waals equation of state-Almost 150 years later. Journal of Chemical and Engineering Data, 64, 4619–4637. https://doi.org/10.1021/acs.jced.9b00264

Kutarov, V., & Schieferstein, E. (2020). Van der Waals equation for the description of monolayer formation on arbitrary surfaces. Colloids and Interfaces, 4(1), 1–10. https://doi.org/10.3390/colloids4010001

Lee, P. A., & Phillips, C. (1998). An assessment of C++ as an introductory teaching language (pp. 1–14). School of Computing Science: University of Newcastle upon Tyne.

Li, H. (2022). Introduction to numerical methods in Java. In Numerical Methods Using Java (Vol. 1291, pp. 1–69). Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6797-4_1

Mailud, T. (2022). Introduction to R programming. In Beginning Data Science in R 4 (pp. 1–50). Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-8155-0_1

Mak, L., & Taheri, P. (2022). An automated tool for upgrading Fortran codes. Software, 1(3), 299–315. https://doi.org/10.3390/software1030014

Marie, van S. (2022). Integrating Python into a physical chemistry lab. Journal of Chemical Education, 99(7), 2604–2609. https://doi.org/10.1021/acs.jchemed.2c00193

Mehtre, V. V, & Pal, A. (2019). MATLAB basics: Commands, functions and operations. International Journal of Research in Engineering, Science and Management (IJRESM), 2(11), 708–710. www.ijresm.com

Mejia, A., Müller, E. A., & Maldonado, G. C. (2021). SGTPy: A Python code for calculating the interfacial properties of fluids based on the Square Gradient Theory using the SAFT-VR Mie equation of state. Universidad de Concepción. https://github.com/gustavochm/SGTPy

Meltzer, R. (2021). A helpful guide for beginners: Python vs C++. Course Report. coursereport.com

Moreira, J. E., Midkiff, S. P., Gupta, M., Artigas, P. V, Snir, M., & Lawrence, R. D. (2000). Java programming for high-performance numerical computing. IBM Systems Journal, 39(1), 21–56.

Nasri, Z., & Binous, H. (2009). Applications of the Peng-Robinson equation of state using MATLAB. Chemical Engineering Education, 43(2), 115–124.

Nguyen, H. (2019). Exploring Python as a replacement for C++ in imperative programming for computing science at Radboud University (P. Achten & E. Barendsen (eds.)). Radboud University.

Nyhoff, L. (2012). Programming in C++ for engineering science (1st ed.). Routledge, CRC Press: Taylor and Francis Group. http://cs.calvin.edu/books/c++/engr-sci

Ott, J., Pritchard, M., Best, N., Linstead, E., Curcic, M., & Baldi, P. (2020). A Fortran-Keras deep learning bridge for scientific computing. Scientific Programming, 8888811, 1–13. https://doi.org/10.1155/2020/8888811

Ozgur, C., Colliau, T., & Rogers, G. (2017). Matlab vs. Python vs. R. Journal of Data Science, 15, 355–372.

Pandey, Y. N., Rastogi, A., Kainkaryam, S., Bhattacharya, S., & Saputelli, L. (2020). Machine learning in the oil and gas industry-Including geosciences, reservoir engineering, and production engineering with Python (W. Spahr, C. S. John, J. Markham, & A. Mirashi (eds.)). https://doi.org/10.1007/978-1-4842-6094-4

Pao, Y. C. (2018). Engineering analysis: Interactive methods and programs with FORTRAN, QuickBASIC, MATLAB, and Mathematica. CRC Press: Taylor and Francis Group.

Papari, M. M., Moghadashi, J., Hosseini, S. M., & Akbari, F. (2011). Modification of van der Waals family equations of state. Journal of Molecular Liquids, 1258, 57–60. https://doi.org/10.1016/j.molliq.2010.10.009

Passos, W. Dos. (2009). Numerical methods, algorithms and tools in C# (1st ed.). CRC Press. https://doi.org/10.1201/9781420007602

Peng, R. D. (2018). R programming for data science. Lean Publishing. http://leanpub.com/rprogramming

Pine, D. J. (2019). Introduction to Python for science and engineering. In Series in Computational Physics (pp. 1–23). CRC Press: Taylor and Francis Group. https://lccn.loc.gov/2018027880

Prodanov, E. M. (2022). Mathematical analysis of the van der Waals equation. Physica B: Physics of Condensed Matter, 640(414077), 1–6. https://doi.org/10.1016/j.physb.2022.414077

Rassokhin, D. (2020). The C++ programming language in cheminformatics and computational chemistry. Journal of Cheminformatics, 12(10), 1–16. https://doi.org/10.1186/s13321-020-0415-y

Rault, J. (2019). The modified van der Waals equation of state-Part IV: Crystalline materials. The European Physical Journal B, 92(22), 1–32. https://doi.org/10.1140/epjb/e2018-90452-6

Rose, G. K. (2017). Computational methods for nonlinear systems analysis with applications in mathematics and engineering (B. A. Newman, D. T. Nguyen, J. Z. Hao, & G. J. Hou (eds.)) [Old Dominion University]. https://doi.org/10.25777/m09c-zj95

Saggion, A., Faraldo, R., & Pierno, M. (2019). Thermodynamics: Fundamental principles and applications (M. Cini, A. Ferrari, S. Forte, G. Montagna, O. Nicrosini, L. Peliti, A. Rotondi, P. Biscari, N. Manini, & M. Hjorth-Jensen (eds.)). Springer Nature Switzerland AG. https://doi.org/10.1007/978-3-030-26976-0

Salamanca, A. V. (2020). Use of mathematical methods in the resolution of chemical engineering problems (J. J. P. Gonzalez & F. R. Baxarias (eds.)). Universitat Politechnica de Catalunya (UPC).

Salcedo-Diaz, R., Ruiz-Femenia, R., & Gomez-Rico, M. F. (2006). An interactive tool for chemical engineers students using easy Java simulations. University of Alicante, 1–12.

Schauber, S. K. (2015). Introduction to the R programming language for statistical computing and graphics. UiO: Centre for Educational Measurement, Faculty of Educational Sciences. https://doi.org/10.13140/RG.2.1.2405.5768

Scheiber, E. (2007). Java in scientific computation: An educational approach. The 4th International Conference on Virtual Learning (ICVL 2009), 181–188.

Scheinerman, E. (2006). C++ for mathematicians-An introduction for students and professionals (1st ed.). CRC Press.

Schildt, H. (2022). Java-A beginner's guide (L. McClain, P. Mon, E. Walters, D. Coward, & L. McCoy (eds.); 9th ed.). 978-1-260-46355-2.

Shafeek, N., & Karunarathne, D. D. (2018). _toFlowchart: A prototype compiler to convert source-code to flowchart. 1st International Conference on Advacnces in ICT for Emerging Regions (ICTer), 157–167. https://doi.org/10.1109/ICTER.2018.8615581

Shrab, S. (2004). Modified van der Waals equation of state. WSEAS Transactions on Biology and Biomedicine, 1(4), 422–424.

Shukla, S., & Singh, S. (2022). Equations of state in chemical engineering using Python. International Journal of Engineering Applied Sciences and Technology (IJEAST), 7(1), 278–286. http://www.ijeast.com

Siddiqui, S. A., & Ahmad, A. (2020). Implementation of Newton's algorithm using FORTRAN. SN Computer Science, 1(348), 1–8. https://doi.org/10.1007/s42979-020-00360-3

Solofsson, Hebing, L., Niedenfuhr, S., Deisenroth, M. P., & Misener, R. (2019). GPdoemd: A Python package for design of experiments for model discrimination. Computer and Chemical Engineering, 125, 54–70. https://doi.org/10.1016/j.compchemeng.2019.03.010

Sudhaka, K. (2018). Python vs. R programming. International Journal of Management, IT and Engineering, 8(8), 70–79.

Sun, L. Y., Zhai, C., & Zhang, H. (2011). Applications of the Soave-Redlich-Kwong equations of state using MATLAB. Advanced Materials Research, 225(226), 492–495. https://doi.org/10.4028/www.scientific.net/AMR.225-226.492

Tang, T., & Wang, J. (2019). Study of computer software applied in teaching of design of chemical engineering principles. Open Access Library Journal, 6(e5292), 1–4. https://doi.org/10.4236/oalib.1105295

Teles, M. dos S., Vianna Jr, A. S., & Le Roux, G. A. C. (2018). Programming skills in the industry 4.0: Are chemical engineering students able to face new problems? Education for Chemical Engineers, 22, 69–76. https://doi.org/10.1016/j.ece.2018.01.002

Terrel, A. R. (2011). From equations to code: Automated scientific computing. In Computer in Science and Engineering (Vol. 13, Issue 2, pp. 78–82). https://doi.org/10.1109/MCSE.2011.31

Tholl, S. (2010). Fortran programs for chemical process design analysis and simulation (A. K. Coker (ed.)).

Tian, J., & Gui, Y. (2003). Modification of the van der Waals equation of state. Journal of Phase Equilibria, 24(6), 533–541.

Tian, J. X., & Gui, Y. (2018). An extension of the van der Waals equation of state (pp. 1–11).

Vasudevan, S. K., Abhishek, S. N., Vignesh, K., Aswin, T. S., & Nair, P. R. (2019). An innovative application for code generation of mathematical equations and problem solving. Journal of Intelligent & Fuzzy Systems, 36(3), 2107–2116. https://doi.org/10.3233/JIFS-169922

Wang, J., & Dowling, A. W. (2022). Pyomo. DOE: An open-source package for model-based design of experiments in Python. American Institute of Chemical Engineers Journal, 17813, 1–24. https://doi.org/10.1002/aic.17813

Winkel, van J. C., & Bella, C. Di. (2018). Proposal for study group: C++ education (pp. 1–8).

Xiduo, W., Jialin, L., & Zhizhen, Z. (2020). Sustainable C++ education in general high school: From teaching programming skills to developing computational thinking. 2020 15th International Conference on Computer Science & Education (ICCSE), 35–38. https://doi.org/10.1109/ICCSE49874.2020.9201830

Yeo, Y. K. (2017). Chemical engineering computation with MATLAB (1st ed.). CRC Press: Informa UK Limited. https://doi.org/10.1201/9781315114880

Zehra, F., Darakhshan, Javed, M., & Pasha, M. (2020). Comparative analysis of C++ and Python in terms of memory and time (pp. 1–11). https://doi.org/10.20944/preprints202012.0516.v1

Zheng, L., Dong, Y., & Yang, F. (2019). Elementary C++ programming. In C++ Programming (pp. 25–78). Walter de Gruyter GmbH. https://doi.org/10.1515/9783110471977-002