[[1]](#footnote-0)

Newton-Raphson: adapting to non-optimal situations (March 2018)

Saúl Enrique Labra Cruz A01020725, Alberto Pascal Garza A01023607, Rodrigo García Mayo A01024595, Manuel Guadarrama Villarroel A01020829

***Abstract*—Newton-Raphson method allows us to make an approximation of the different roots a given function has. Unfortunately, this method starts failing whenever our initial conditions are not optimal or require to calculate more than one root per guess point. The objective of this paper is to show, by solving the three presented problems, that the failing points of the Newton-Raphson method can be surpassed. The first problem to be addressed consists on the steps calculated on the Newton-Raphson’s iterations growing exponentially, resulting in inaccurate approximations. Secondly, we will overcome the obstacle of finding both a left and a right root with a single initial guess. Lastly, we will find all of the roots of a function that lie within a given interval, without knowing how many to find nor the initial guesses to use. The obtained results are observed to be extremely accurate, showing that it is indeed possible to make adaptations to the Newton-Raphson in order to cope with the presented obstacles.**

# INTRODUCTION

This paper reviews some of the problems that can be solved evaluating the roots of a function. The roots of a function are an important mathematical concept because they specify what happens when a particular state becomes zero. There are many ways to approximate the values for the roots in a function. In this paper we will only evaluate the Newton-Raphson root finding method.

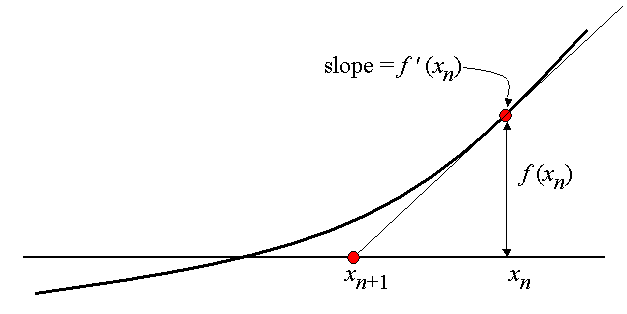
Newton-Raphson is a numerical method used to progressively approximate the roots of a given function (f(x)=0). Before you can start with the method you need to have an initial value (Xi) in which it will start the approximations. There is no formal rule as to how to obtain this value, the only requirement is that in order to find a certain root, the initial guess must be closer to that root than any of the other roots. Once you get the Xi, the next step is to apply the Newton-Raphson method, which states that the next best approximation is equal to the current approximation minus the function you want to evaluate over its derivative.



*Figure 1: the Newton Raphson Methods*

In order to obtain a more accurate solution this process must be iterated as many times as required to reduce the approximation error. The approximation error may vary for each specific problem.

What this method does is draw a tangent line in f(Xi) with a slope of f’(Xi) as seen in the graph below. Then we solve the tangent to y=0 where it intersects with the X-Axis and solve Xi = Xi+1 to get a better approximation.



*Figure 2: Representation of the approximation when generating the tangent line*

Even though this method is efficient, there can be some situations in which the method may not work. This situations are generated mainly because the initial value is near a critical point in the function such as a maximum, minimum or inflection point. When you create the tangent line close to the critical point the tangent is sent far away from the area of interest, which leads the method to not converge or converge after a big amount of iterations. Another problem occurs with Newton-Raphson when there are multiple roots. Newton-Raphson uses the derivative of the function as a denominator in its formula so, for values that converge close to the root, the derivative gets close to zero or zero and it can lead to a division over zero, which is indeterminate, or get a huge value which can make the method to converge in any root or not converge at all.

The accuracy of this method relies in the number of iterations, the distance from the initial guess to the root and the operation made between the function and its derivative made in the Newton-Raphson formula. The accuracy of the current result when using an iterative method is measured using the Relative Approximation Error. This error calculates the percentage of error between your current data and past data with the following formula:

This paper aims to give a solution to some of the problems previously mentioned. The first problem in this paper is to solve the Beattie-Bridgeman equation, an equation of state that is used in thermodynamics, to describe the state of the object being studied by relating pressure, temperature and volume. Once the equation is solved the compressibility factor for that gas is calculated. The data used in the problem will have as variables the temperature and the pressure. For this problem, our hypothesis is that our initial guess plays an important role in the process of finding the root because we might be crossing through critical points. Therefore, to prevent our Newton-Raphson from failing, we need to take into account that the volume has a dependency with some of the other variables. That way we can adjust the initial value to make a better approximation.

For the second problem, given the function we need to propose a method that finds the closest left and right root for any initial value X in the function. Our hypothesis is that the Newton Raphson method will need a modification in order to approximate both left and right roots from a single Xi value. Because Newton Raphson as it is, only approximates the closest root to the Xi given. So, in order to calculate the second root we propose to change the sign of the created tangent so the approximation will head to the root on the other side. We also expect that given the position of the values of the points we need to solve, which are close to critical points, the Newton Raphson method will not converge without making modifications. Our proposal is to adapt the method by multiplying it with a constant that will make the interval for the the approximations go bigger or lower. If the intervals become lower, the number of iterations we will need to make are going to increase but will eventually converge in the root.

The third problem requires us to find all the roots of the following function: , in the interval: (-3,4). We should also consider these questions:

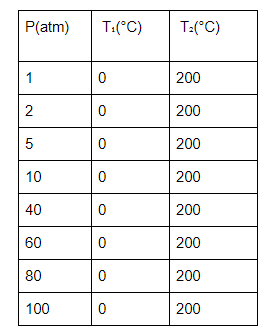
* If the roots are searched by the left of the point and the method finds it by the right, what is the implication?
* Which is the problem when some zero or close-zero point is evaluated at the derivative?

Our hypothesis is that, in order to calculate all the roots of the function in that interval, we will have to use different values for our initial guesses. Unfortunately, the number of roots that lie within that interval is unknown to us. Therefore, comparing the values of the first derivative every once in a while within the given interval will allow us to know if our function has already crossed the X axis or not. Once we have an estimate of where our roots might be located we can then proceed to select our initial guesses and perform Newton-Raphson for each one of them.

# Methodology

## Problem 1

The first problem requested us to calculate an approximation of the volumes for 1 mol of methane making use of the Beattie-Bridgeman formula. The requested volumes had the following pressure-(P) and temperatures-(T) values (Table 1):



*Table 1: Requested volumes and temperatures*

However, it is important to note that the Beattie-Bridgeman formula is composed of several other variables, each one with its own formula:

**Brittie-Bridgeman:**

**Variables for the Beattie-Bridgeman equation:**

*Where A0=2.2789, B0=0.05587, a=0.01855, b=-0.01587, c = 128,000 and R=0.08205*

The first problem we encountered was deciding which initial guesses to use. Trying to decide by ourselves what our initial guesses would only result in our approximation growing exponentially or giving NaN values. Fortunately, the problem also recommended us to use the molar volumes as initial guesses. According to the ideal gas theory, a molar volume for a gas can be calculated by solving for V in following formula:

Resulting in:

Once knowing the formula, we proceeded to calculate our initial guesses for the Newton-Raphson iterations by substituting the corresponding temperature and pressure values. It is important to note that all of the previous formulas required the temperature to be in Kelvin and not Centigrees. Therefore, we needed to convert our temperatures to Kelvin with the following formula:

Since our initial volumes are the theoretical volumes, our approximations will be really close to our initial values. Therefore, the number of iterations needed was always extremely low. However, as the problem suggested, we kept our maximum number of iterations as twenty iterations. In order to break our Newton-Raphson method, we needed a criteria to make it stop. The criteria used was the relative approximation error. According to the instructions, our program should stop whenever our error reached a value equal or lower than 0.000001. We calculated our current error after each iteration with the following formula:

Now that we know the approximate volumes and every given temperature and pressure, as well as the fact that that those values are extremely close to the theoretical values, we can proceed to calculate the compression factor. The compression factor was calculated with the following formula:

At last, we proceeded to graph our results by making use of Matlab’s function plot() . In the first part of the graph we set in the pressure values for our Y axis and the Compression factor for our X axis. For the second part of the graph, our pressure values remained as our Y axis while our X axis turned out to be our approximated volumes.

The obtained values from the Beattie-Bridgeman formula tells us the volume 1 mol of methane reaches with certain conditions of pressure and temperature. This values have many applications. Among others it can predict how methane will react when is subjected to certain conditions. You can calculate how much the methane will expand when changing temperature. Or you can calculate how much methane is inside a volume with specific temperature and pressure. In the Results section we will analyze the change in volume with same pressure but different temperatures.

## Problem 2

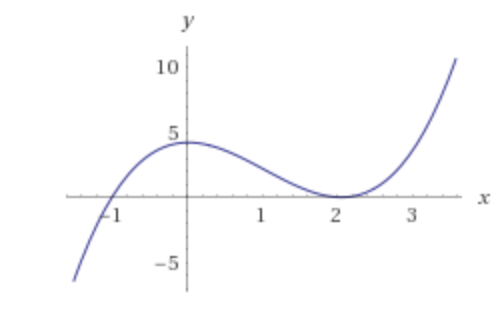
For the second problem we had to calculate the closest left and right roots of some initial guesses of “x” given by the problem. The initial guesses were the following:

x₁ = 0.0161

x₂ = 2.051

x₃ = 0.5

The first step was to solve the equation to zero. Newton-Raphson’s method requires the evaluated to be solved to zero and will only throw one value per guess. The direction in which our guess will be guided is determined by the signs of our evaluated function over the evaluated derivative. Therefore, we started by graphing the function to be able to calculate the regular results. The graph is the following:



*Figure 3: Graph of*

It is important to note that our initial guesses were all extremely close to the function’s critical points. Therefore, we found out that the Newton-Raphson’s method would start failing because the values of the derivatives would easily reach zero, resulting in NaN or infinite values for our steps. In order to solve this, we were forced to make a adaptation to the Newton Raphson formula, modifying the steps Newton- Raphson takes. To accomplish this, we multiplied the factor (f(x)f'(x)) by a constant to effectively slow its growth. The formula for the Newton-Raphson used is:

Where “Xnew” is our new approximation, “Xi” our current “X” value and “a” our constant to slow the process.

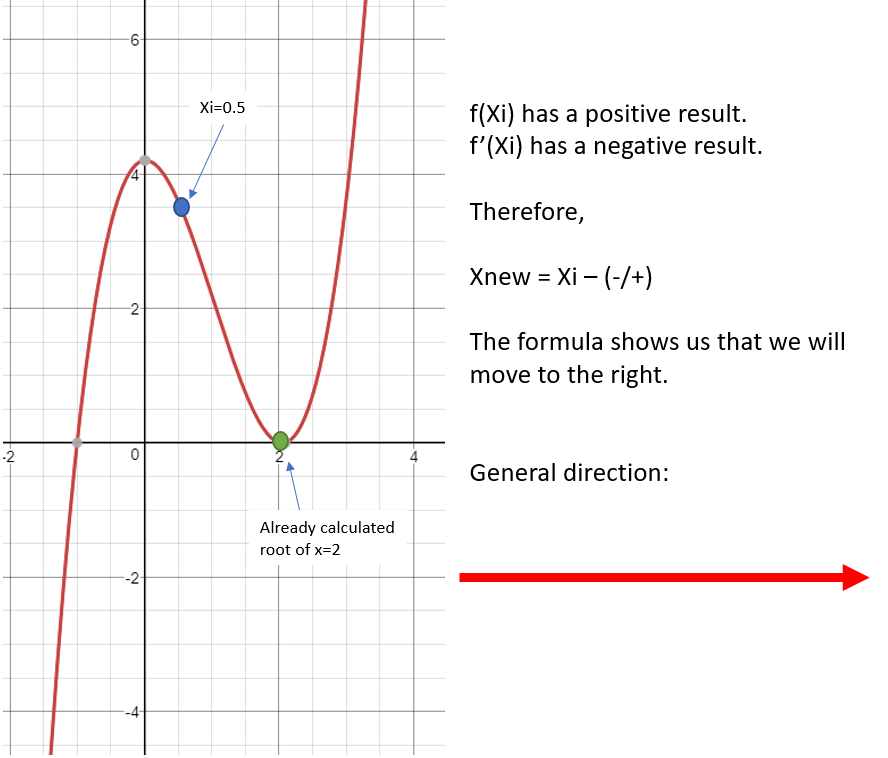
For the first part of the problem, which consists of finding the regular roots, we can use any constant that is lower than one. This is because we are following the natural flow of the method and we just need to make a couple more steps than the ones we would regularly do. The constant is only needed to avoid taking big jumps for the initial value of 0.5, which happens to be exactly the same distance from the root of x=2 and x=-1. The other initial guesses could work without the need of a constant. This initial value, without a constant to make its steps just a tad bit smaller, will start moving towards the root of 2 but will eventually jump between the said root and x=-1, ending up in a third root of x=2.1. While the number of iterations needed should not be that big, our initial guess of x=0.0161 requires more iterations because it keeps jumping from one side of the function to the other. After trial and error, we noticed that this point managed to approximate its root quite preciscely by reaching 50 iterations. However, we decided to set the maximum number of iterations to 100 to maximize the accuracy.

Once we calculated our natural roots, we needed to force our Newton-Raphson to move to the opposite direction in order to find the root on the other side of Xi. We accomplished this by initially having our constant change signs. This means that our constant “a” would actually start as “-a”. In other words, our Newton-Raphson started iterating with the next formula:

It is important to note, however, that we did not maintain our constant as “-a” through the whole process. Having a negative constant was only for the method to walk to the opposite direction than the one it took to find the first root. However, our constant should return to its original value “+a” soon after. The moment we reverted “a” was whenever we detected that our derivative had changed its sign.

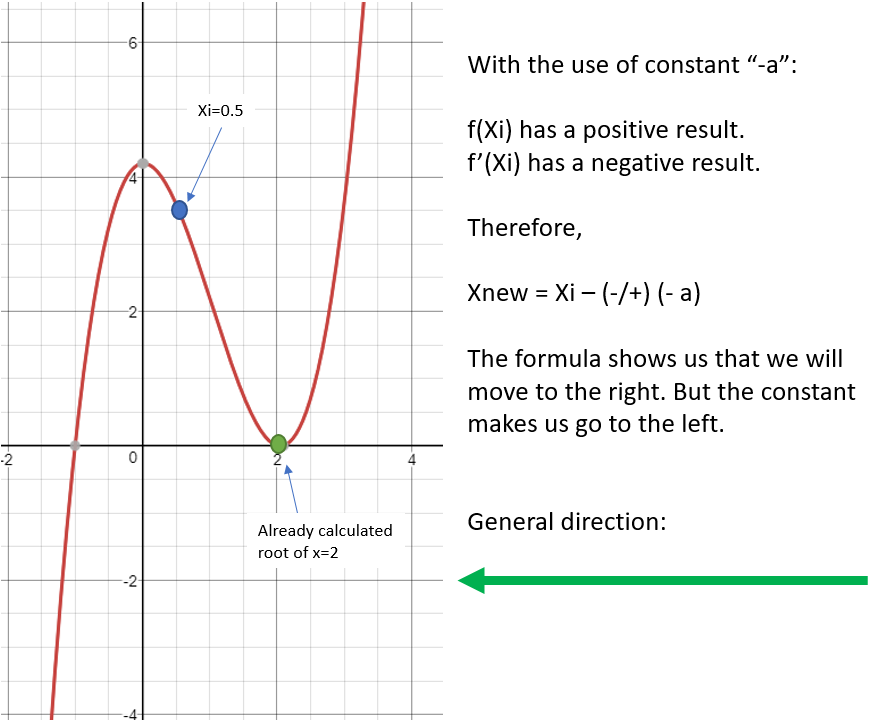
In other words, when our derivative started being a negative value or a positive value, we returned to use the original constant “+a” whenever we saw the value became either positive (if it started as a negative value) or negative (if it started as a positive value). It is important to note that, once that we reverted our constant to its original value, we never again made use of the initial “-a”. As a result, the Newton-Raphson’s method returned to how it regularly works. Therefore, this part of our method works as follows:

Taking our first point Xi=0.5 as an example:



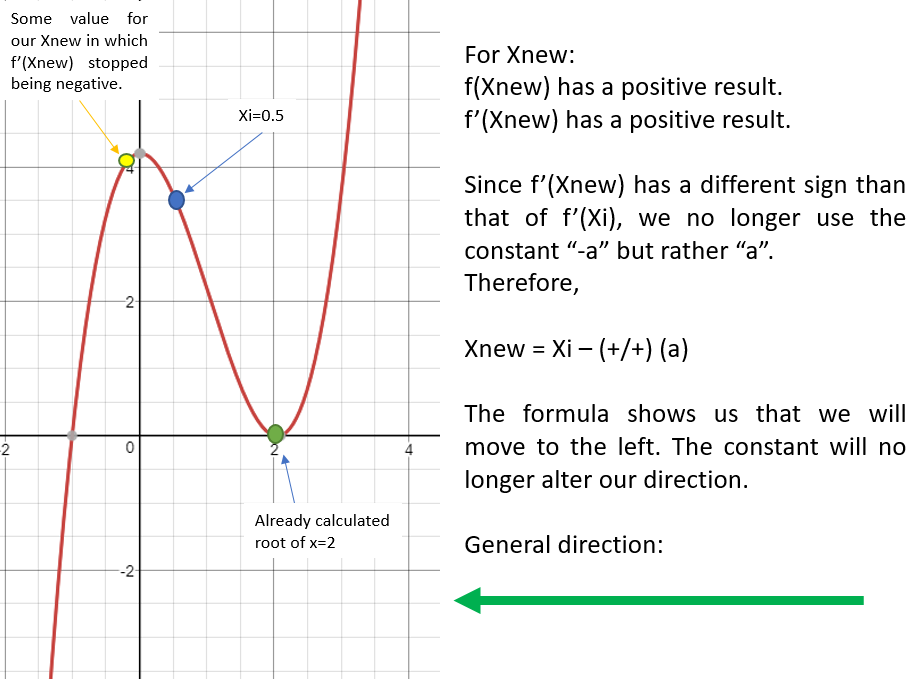
*Figure 4: Graph of the function with the initial guess (purple) and root (green) plots*

However, our constant starts being negative. Therefore:



*Figure 5: Graph of the function and the initial guess (purple) and root (green) plots*

The first couple of iterations will work with our Newton-Raphson moving towards the opposite direction. However, whenever the method detects that our derivative no longer has the same sign which started when we began iterating, our constant “a” will return to its original value. Graphically, this happens like:



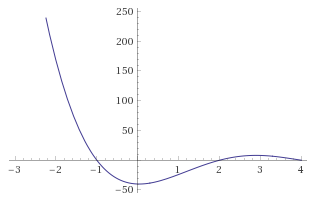
*Figure 6: Graph of the function with the initial guess modification*

Lastly, our Newton-Raphson will finish iterating to get the value of the root, which in this case happens to be “-1”. Nevertheless, this change in the constant will make the possible error more notorious. That is, we run the risk of having our values skyrocket towards nonsense, especially when our x is 0.0161 because like we mentioned before it needed more iterations than the others to get a more precise value of the root. Fortunately, this can be solved for most of our points by keeping our constant value of 0.5, which makes our steps significantly smaller. However, for x=0.0161 we noticed that this constant will not work. By trial and error, we noticed that the constant hast to be of the order 1 x 10-4 . Therefore, we decided to calculate our new constant as the difference between the Xcritical closest to that point and our X of 0.0161 to keep a certain relation with our problem. Notice that any constant from that order would be sufficient, however. Xcritical-Xi in this situation results in a value of 0.000135, perfectly matching our previously discovered condition.

Finding the closest left and right root for a certain x in a function can help us.

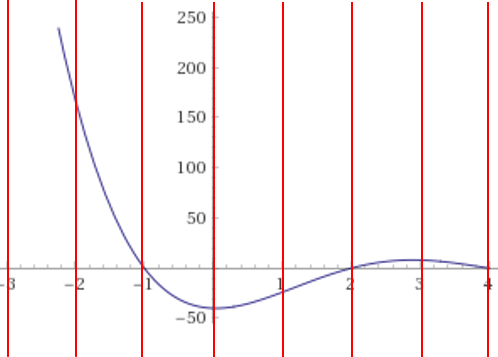
## Problem 3

This problem required to find all the roots of a given polynomial inside a certain interval of evaluation, all within a single run of a MATLAB program. In order to know what the real values of the roots were we obtained the graph of the function within the interval. This graph is shown in Figure 3. If we analyze the graph we can clearly see that there are three different roots in said interval. However, one of those roots is x=4, which is the upper limit of our evaluation range and is not included. Therefore, we should only find two different roots.



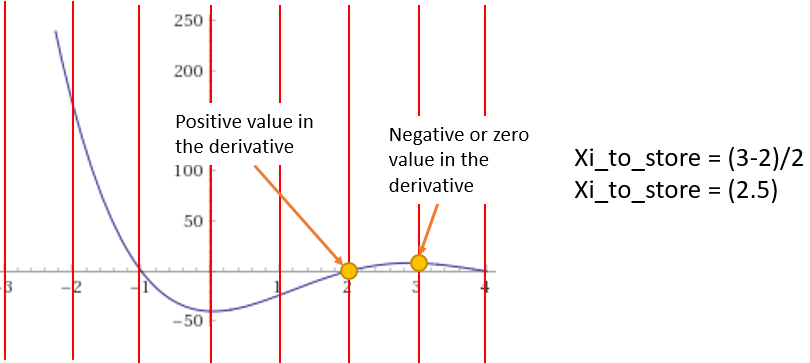
*Figure 7: Graph of , in the interval (-3,4)*

Unfortunately, we cannot depend on how the graph looks like when making our program, which leaves us with two different problems: not knowing the number of roots to find and not knowing what initial guesses to use. However, an easy method of finding out how many possible roots we have is with the inflexion points. While we did not properly calculate the inflexion points for this function because it would be extremely complicated and would require us to deal with the same problems we already have, we can make an estimation of their location. In order to make this estimation, we decided to divide our range of evaluation. In other words, we decided to check what our derivatives’ values are throughout the whole range, moving along the function in intervals of 1. Graphically, our divisions would look as follows:



*Figure 8: Subdivisions for the graph of , in the interval (-3,4)*

Before calculating our derivatives, however, we made some assumptions in order to make less calculations. To find the first root we can use the lower end of our range. Therefore, the first value we used as initial guess was that of X= -3. Once we calculated the values of our derivatives, we proceeded to compare them. In order for us to know between which of the subdivisions could an inflexion point exist, we need to check whether the value of the derivative in the left subdivision and the value of the derivative in the right subdivision contain the same sign or not. In other words, we have to evaluate if both values are either positive or negative, but not one of each. Whenever one of those values turns out to be positive while the other one is negative we have found the interval between which an inflexion point exists. However, since we already have a first initial guess, the first time we detect that variation in the derivatives will not be taken into account. This is because our function has not changed the signs of its derivatives a single time from -3 to said point and xi=-3 will already help us calculate the corresponding root. For the next times we find a change in signs, we will store as our initial guess as the average of the lower end and the higher end of this new interval. For example:



*Figure 9: Example of the change in signs in the derivatives for the graph of , in the interval (-2,3)*

With this, we were able to locate all the possible “good” points for our initial guesses. Lastly, we determined that our error tolerance would be of 0.000001 to be able to approximate quite precisely the roots of our function and proceeded to apply Newton-Raphson. This problem can be implemented anytime you need to find the roots of a function with a certain interval.

# Results

## Problem 1

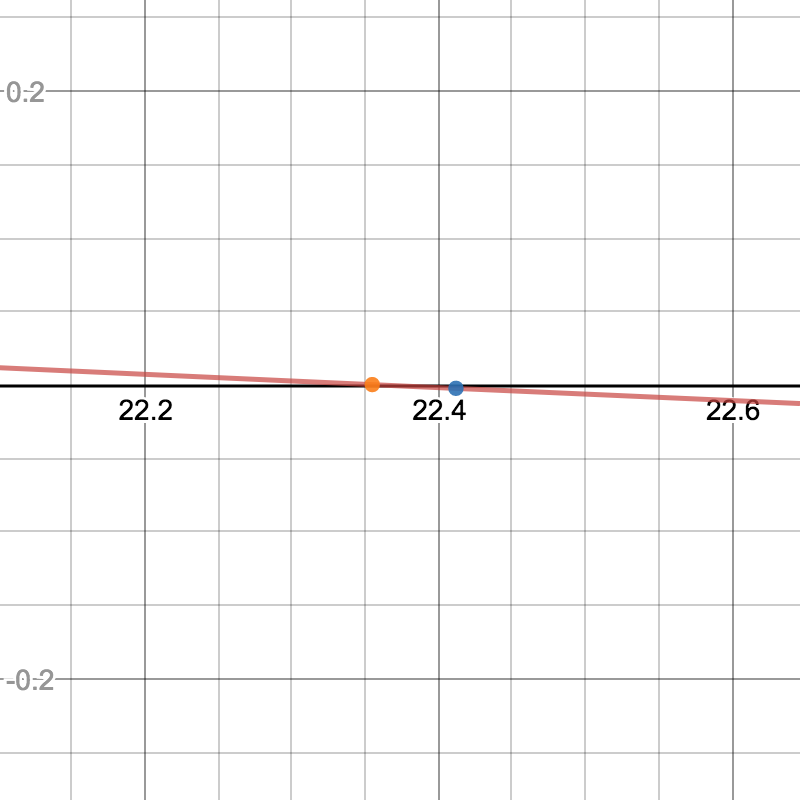
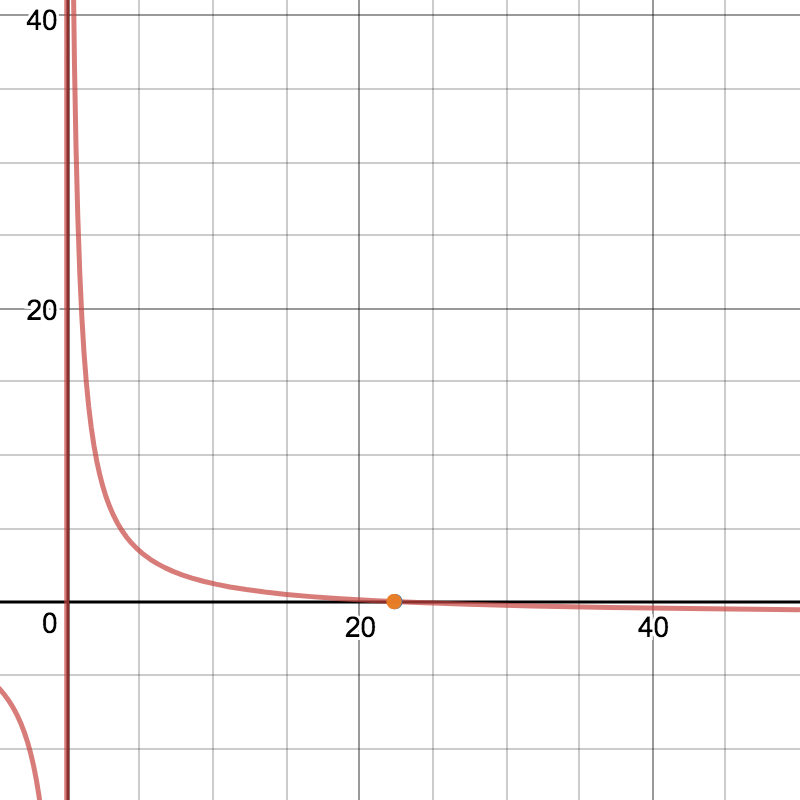
The first step of calculating the initial guesses for the Newton Raphson method throwed the following data (table 2 row 1 to row 3), indicating where to start calculating the roots for every case (specific temperature and pressure) and with this information the real root was calculated (table 2 row 4) with several iterations of the method fulfilling the predefined error and iterations tolerance (0.000001 and 20 respectively), a true relative error was obtained by comparing the obtained volume with the method and the initial guess.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P(atm) | Temperature (K) | Initial Volume Guess (L) | Root Volume (L) | True Relative Error (Initial guess and root) |
| 1 | 273.15 K | 22.4119 | 22.3599 | 0.2328% |
| 2 | 273.15 K | 11.2059 | 11.1539 | 0.4669% |
| 5 | 273.15 K | 4.4823 | 4.4302 | 1.1781% |
| 10 | 273.15 K | 2.2411 | 2.1890 | 2.3845% |
| 40 | 273.15 K | 0.5602 | 0.5080 | 10.2755% |
| 60 | 273.15 K | 0.3735 | 0.3217 | 16.1013% |
| 80 | 273.15 K | 0.2801 | 0.2292 | 22.1866% |
| 100 | 273.15 K | 0.2241 | 0.1748 | 28.1635% |
| 1 | 473.15 K | 38.8219 | 38.8180 | 0.0102% |
| 2 | 473.15 K | 19.4109 | 19.4070 | 0.0205% |
| 5 | 473.15 K | 7.7643 | 7.7606 | 0.0489% |
| 10 | 473.15 K | 3.8821 | 3.8786 | 0.0927% |
| 40 | 473.15 K | 0.9705 | 0.9684 | 0.2147% |
| 60 | 473.15 K | 0.6470 | 0.6459 | 0.1692% |
| 80 | 473.15 K | 0.48527446875 | 0.48516 | 0.0236% |
| 100 | 473.15 K | 0.388219575 | 0.38907 | 0.2186% |

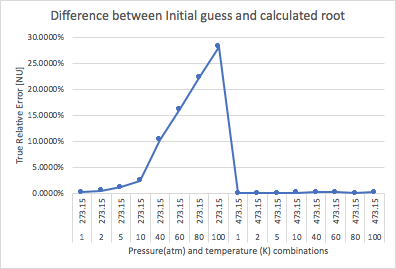
*Table 2: Pressure (atm) and temperature (K) combinations, the initial guesses, the roots and the true relative error between the initial guess and the root obtained after the method*

In the figures below (Figures 10 and 11) a graph of the function is shown, the first initial guess and root are plotted one beside the other to show their proximity and demonstrate why using the compressibility of an ideal gas (Z=1) for solving for the volume in the formula was an ideal initial guess, it was also observed that the more the temperature rises the less difference between the initial guess and the calculated root after Newton-Raphson thus a smaller error and the more pressure the more error (Figure 12).

The ideal case when making the Newton-Raphson method is to execute it with the closest initial guess to the root because in this way it is almost certain that the method will calculate the closest root to the evaluation point because its nature indicates that the root that will be calculated is the one that is closer to the intersection between the tangent line departing from the initial guess point.



*Figures 10 and 11: Function graph and first combination initial guess and root*



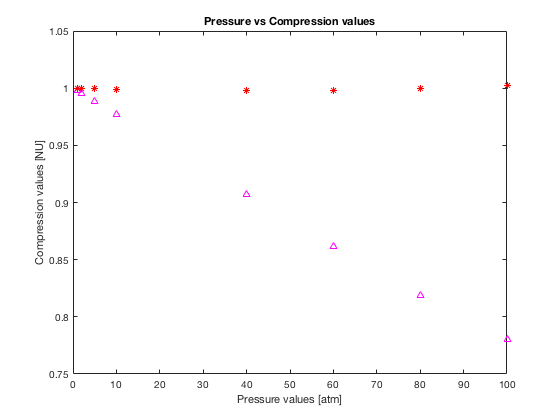
*Figure 12: True Relative Error initial guesses and roots*

After getting our initial guesses for solving Newton- Raphson, the method was executed giving the next output indicating satisfactory results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pressure (atm) | Temperature (K) | Obtained Volume (L) | Approximation Relative Error | Necessary Iterations |
| 1 | 273.15 | 22.3599 | 2.9284e-09 | 3 |
| 2 | 273.15 | 11.1539 | 4.7009e-08 | 3 |
| 5 | 273.15 | 4.4302 | 4.0096e-14 | 4 |
| 10 | 273.15 | 2.1890 | 8.8453e-12 | 4 |
| 40 | 273.15 | 0.5080 | 6.3795e-07 | 4 |
| 60 | 273.15 | 0.3217 | 2.6744e-12 | 5 |
| 80 | 273.15 | 0.2292 | 3.4934e-10 | 5 |
| 100 | 273.15 | 0.1748 | 2.1786e-08 | 5 |
| 1 | 473.15 | 38.8180 | 1.8304e-14 | 3 |
| 2 | 473.15 | 19.4070 | 1.4645e-13 | 3 |
| 5 | 473.15 | 7.7606 | 5.6995e-12 | 3 |
| 10 | 473.15 | 3.8786 | 7.0129e-11 | 3 |
| 40 | 473.15 | 0.9684 | 2.12e-09 | 3 |
| 60 | 473.15 | 0.6459 | 8.2968e-10 | 3 |
| 80 | 473.15 | 0.4851 | 2.6316e-13 | 3 |
| 100 | 473.15 | 0.3890 | 2.5299e-09 | 3 |

*Table 3: Cases of pressures and temperatures, root and required iterations to solve with Newton Raphson*

Once the real volumes were obtained the correct compressibility factors were calculated with the formula and plotted in a graphic to show the behaviour when the temperature and pressure vary (figure 13).



*Figure 13: Pressure vs compression values*

## Problem 2

The following table contains the results we got from the methodology we previously discussed using the adapted Newton-Raphson method (Table 4).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Initial X value | Root 1 | Alfa (a) for first root | Number of Iterations | Approximate Relative Error for Root 1 |
| 0.0161 | -1 | 0.5 | 59 | 6.3226e-07 |
| 2.051 | 2.1 | 0.5 | 25 | 8.9721e-07 |
| 0.5 | 2 | 0.5 | 23 | 6.688e-07 |
| -2 | -1 | 0.5 | 21 | 6.4704e-07 |
| 3 | 2.1 | 0.5 | 24 | 6.6234e-07 |

*Table 4: Table containing the roots for the initial values of x and its corresponding approximate relative error*

|  |  |  |  |
| --- | --- | --- | --- |
| Root 2 | Alfa (a) for second root \*1 | Number of Iterations | Approximate Relative Error for Root 2 |
| 1 | 0.000135 | 41863 | 9.9997e-07 |
| 2 | 0.5 | 24 | 5.8562e-07 |
| -1 | 0.5 | 19 | 7.7876e-07 |
| No other root | 0.5 | 2 \*2 | 0.1554 \*2 |
| No other root | 0.5 | 2 \*2 | 2.1742e-05 \*2 |

*Table 4 (continuation)*

\*1 Note:This values start being the negative of the number displayed. However, during the process they return to the displayed value.

\*2This values are calculated. However, they are nonsense. This were the last values before noticing that there are no other existing roots.

## Problem 3

The following table (Table 5) contains the results of the initial guesses to be used where our inflexion points might be located.

|  |  |
| --- | --- |
| Initial Guess | Value |
| Root 1 | -3 |
| Root 2 | 2.5 |

*Table 5*

The following table (Table 6) contains the results of the roots after each iteration of the Newton-Raphson and the errors.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration | Current value of root 1 | Approximate Relative Error | Current value of root 2 | Approximate Relative Error |
| 0 | -1.9668 | \* - | 1.6797 | \* - |
| 1 | -1.3408 | 0.4668 | 1.9670 | 0.1461 |
| 2 | -1.0614 | 0.2633 | 1.9995 | 0.0163 |
| 3 | -1.0025 | 0.0587 | 2 | 2.6012X10-4 |
| 4 | -1 | 0.0025 | - | - |

*Table 6*

\* This Error cannot be calculated due to the lack of a previous value.

# Discussions and Conclusions

In conclusion, Newton-Raphson is a powerful method to approximate almost any given function. Unfortunately, there are certain situations in which the method fails to fulfill our necessities. However, it is possible to implement partial or total adaptations for each specific problem in order to guide us to the desired solution. To be able to find the work-arounds we often find ourselves in the need of using outside equations to calculate our initial guesses, implementing a mathematical break to slow our iterations or even force our method to work in a different manner.

Whenever we require Newton-Raphson to calculate an approximation for a function in which our variable has a dependency with other variables as seen in the first problem, we can just find said relation and use it to calculate several initial guesses. We will require a different initial guess for every time any of the variables our guess depends on changes its value. As a result, our initial guesses will have the theoretical value of said approximation, and will also allow us to need less iterations and get more precise results.

By looking at the results we got from the first problem, we can see that at higher pressure the obtained volume decreases. At the same time, if we increase the temperature of a gas its volume will increase. And that can be seen by analyzing the Beattie Bridgeman formula. That happens because the pressure is inversely proportional to the volume and the temperature is directly proportional to the volume. The solutions given to this problem shows us the properties of methane under certain conditions. The hypothesis we posed was fulfilled and the proposed initial values we calculated led, by using the Newton-Raphson method, to a close approximation of the real values of the volume.

Our conclusion for the second problem is that the necessary adaptations we hypothesized were indeed necessary because the initial values we needed to use in the Newton-Raphson method were problematic because they were close to critical points. We saw that using a value of 0.5 would diminish the function enough for it to approximate the value of the roots correctly. This specific problem made us realize the way the method iterates should be adapted for the each problem because that way we can augment or reduce the iterations it needs to do to get close to the result and the makes the method more efficient.

For the third problem, we faced the obstacle of not knowing the number of roots we were looking for nor the initial guesses we could use to obtain them. We managed to overcome this by dividing the search interval and scanning for the potential location of the function’s inflexion points. By doing this, we were able to determine the number of roots to find, and the inflexion points allowed us to continue the normal process of Newton-Raphson to approximate said roots.

Overall, this problems made us realize the importance of adapting the mathematical methods to the specific problems at hand. The task of an engineer is to think of a way of implementing the knowledge to solve the problems. Matlab is an impressive tool that helps in making models that give solutions to the problems but its the engineer the one that has to make everything work. The numerical methods show us different ways of solving a problem and they depend on the available data.

# references

## [1] Chapra, S., Canale, R. (7th Edition). (2015). Numerical Methods for Engineers. New York, NY: McGraw-Hill.

## [2] Duesberg, G. (24/08/2016). Compressibility Factor. Retrieved from https://chemistry.tcd.ie/staff/people/duesberg/ASIN/lecture%20notes/Chapter%203\_%20Real%20Gases\_annotated.pdf

## [3] Gershon, A., Yung, E., Khim, J. (n.d.). Newton Raphson Method. Brilliant. Retrieved from https://brilliant.org/wiki/newton-raphson-method/

## [4] Gramoll, K., & Huang, M. (n.d.). The Ideal-gas Equation of State. Retrieved from http://www.ecourses.ou.edu/cgi-bin/ebook.cgi?topic=th&chap\_sec=02.4&page=theory

## [5]Hansen, E. Patrick, M. Numer. Math. (10/02/1976). A family of root finding methods. Retrieved from https://doi.org/10.1007/BF01396176

## [6] N.A. (23/10/2006). The Newton Raphson Method. Retrieved from https://www.math.ubc.ca/~anstee/math104/104newtonmethod.pdf

## [7] Kreinovich, V. (13/10/2012). How to define relative approximation error of an interval estimate. Retrieved from http://www.m-hikari.com/ams/ams-2013/ams-5-8-2013/kreinovichAMS5-8-2013-1.pdf

## [8] Smith, M. (10/1/1998). Newton Raphson Technique. Retrieved from http://web.mit.edu/10.001/Web/Course\_Notes/NLAE/node6.html

## VI. Attachments

**Problem 1 (Matlab code)**

% Alberto Pascal A01023607

% Saúl E. Labra A01020725

% Rodrigo García A01024595

% Manuel Guadarrama A01020829

clc;

clear all;

close all;

%Storing the constants of the problem

R=0.08205;

T=273.15;

pressures=[1,2,5,10,40,60,80,100];

V=(R\*T)./pressures;

P=0;

Bo=0.05587;

A0=2.2789;

c=128000;

a=0.01855;

b=-0.01587;

% Other variables;

B= (R.\*T.\*Bo)-A0-((R.\*c)./(T.^2));

G= ((-1.\*(R.\*T.\*Bo.\*b)) + (A0.\*a) -((R.\*c.\*Bo)./(T.^2)));

D= (R.\*Bo.\*b.\*c)./(T.^2);

num\_iter=1;

Initial\_Vol=zeros(1,2\*length(pressures));

%Arrays to store results

Vol\_end=zeros(1,2\*length(pressures));

Err\_End=zeros(1,2\*length(pressures));

%Defining the function and its derivative

f= @(V, x)((R\*T)./V) + (B./(V.^2)) + (G./(V.^3))+(D./(V.^4)) -pressures(x);

%Array to store results of the function

Arr\_fun=zeros(16,20);

df=@(V) -1.\*((2.\*B.\*(V.^2) + (4.\*D) + (3.\*G.\*V) + (R.\*T.\*(V.^3)))./(V.^5));

%Defining tolerance and intial error

Err\_tol=.000001;

error = 1000000;

%Array to store Z (compresibility factor)

z\_zero=zeros(1,length(pressures));

z\_two=zeros(1,length(pressures));

%Iterates over each scenario (pressure)

for x=1:1:length(pressures)

P=pressures(x);

Volum= R\*T/P;

Initial\_Vol(1,x)=Volum;

%Newton Raphson is executed while the iterations doesn't surpass 20 and

%the error tolerance is fulfilled

while (error>Err\_tol && num\_iter<20)

%The function is evaluated and saved with the current pressure iteration

num1=f(Volum, x);

Arr\_fun(x, num\_iter)=num1;

%The derivative of the function is evaluated with the current pressure iteration

num2=df(Volum);

%Newton Raphson formula is applied

Vnew = Volum - (num1/num2);

%After the first iteration error can be calculated

if num\_iter>1

error =(100\*(Vnew-Volum)/Vnew);

end

Volum=Vnew;

num\_iter=num\_iter+1;

end

%Volume and error after NR is saved

Vol\_end(x)=Volum;

Err\_End(x)=error;

%Compresibility factor is calculated

z\_zero(x) = P\*Volum/(R\*T);

disp(['The volume ' num2str(x) ' is ' num2str(Vol\_end(x)) ' with an error of ' num2str(Err\_End(x)) ' after iteration ' num2str(num\_iter -1)]);

%Resetting initial error

error=1000000;

num\_iter=1;

end

%The same code is executed for the second temperature

T2=473.15;

B= (R.\*T2.\*Bo)-A0-((R.\*c)./(T2.^2));

G= ((-1.\*(R.\*T2.\*Bo.\*b)) + (A0.\*a) -((R.\*c.\*Bo)./(T2.^2)));

D= (R.\*Bo.\*b.\*c)./(T2.^2);

f2= @(V, x)((R\*T2)./V) + (B./(V.^2)) + (G./(V.^3))+(D./(V.^4)) -pressures(x);

df2=@(V) -1.\*((2.\*B.\*(V.^2) + (4.\*D) + (3.\*G.\*V) + (R.\*T2.\*(V.^3)))./(V.^5));

for x=1:1:length(pressures)

P=pressures(x);

Volum= R\*T2/P;

Initial\_Vol(1,x+8)=Volum;

%Newton Raphson

while (error>Err\_tol && num\_iter<20)

num1=f2(Volum, x);

Arr\_fun(x, num\_iter)=num1;

num2=df2(Volum);

Vnew = Volum - (num1/num2);

if num\_iter>1

error =(100\*(Vnew-Volum)/Vnew);

end

Volum=Vnew;

num\_iter=num\_iter+1;

end

Vol\_end(x+8)=Volum;

z\_two(x) = P\*Volum/(R\*T2);

Err\_End(x+8)=error;

error=1000000;

disp(['The volume ' num2str(x+8) ' is ' num2str(Vol\_end(x+8)) ' with an error of ' num2str(Err\_End(x+8)) ' after iteration ' num2str(num\_iter -1)]);

num\_iter=1;

end

%A plot of the compressibility factors vs the pressure is shown

plot(pressures,z\_zero,'m^', pressures,z\_two, 'r\*');

title('Pressure vs Compression values');

ylabel('Compression values [NU]') % x-axis label

xlabel('Pressure values [atm]')

**Problem 2 (Matlab code)**

% Alberto Pascal A01023607

% Saúl E. Labra A01020725

% Rodrigo García A01024595

% Manuel Guadarrama A01020829

clc;

clear all;

close all;

Xi=1;

f= @(x) x^3 - (31/10)\*x^2 + (1/10)\*x + (21/5);

df = @(x) 3\*x^2 - (31/5)\*x + (1/10);

Xi\_values=zeros(2,3);

Xi\_values(1,1)=0.5;

Xi\_values(1,2)=0.0161;

Xi\_values(1,3)=2.051;

Xi\_values(2,1)=0.5;

Xi\_values(2,2)=0.0161;

Xi\_values(2,3)=2.051;

solutions=zeros(2,3);

Err\_tol=.000001;

initial\_error=10000;

error = 1000000;

constant=0.5;

Err\_vals=zeros(2,3);

Iterations=zeros(2,3);

%we calculate the normal values the newton raphson would give

for x=1:1:3

Xi= Xi\_values(1,x);

disp(['First root for ' num2str(Xi) ' : ']);

for n=0:1:100

num1=f(Xi);

num2=df(Xi);

xnew= Xi- (num1/num2)\*constant;

if(n>=1)

error= abs(((xnew-Xi)/xnew));

end

if (error<=Err\_tol)

Xi=xnew;

break;

end

Xi=xnew;

% disp ([' Mi Xi ahora va a valer ' num2str(Xi)]);

end

solutions(1,x)=Xi;

Iterations(1,x)=n;

Err\_vals(1,x)=error;

disp(['My approximation for X= ' num2str(Xi) ' with error of ' num2str(error) '% after iteration : ' num2str(n)]);

error=10000;

end

for x=1:1:3

Xi= Xi\_values(1,x);

disp(['Second root for ' num2str(Xi) ' : ']);

for n=0:1:50000 % this number of cycles are because of constant2. We divided by 10,000 our original value. Therefore, 10,000 times more cycles.

num1=f(Xi);

num2=df(Xi);

if n==0

if num2>=0

sign=-1;

else

sign=1;

end

end

if(num2>0&& sign==-1) || (num2<0 && sign==1)

constant=-0.5;

constant2=-0.000154; % this value is obtained by doing Xmax-X where xmax= 0.016254 and x=0.0161

else

constant=0.5;

constant2=0.000154;

end

if x==2

xnew= Xi-(num1/num2)\*constant2;

else

xnew= Xi - (num1/num2)\*constant;

end

if(n>=1)

error= abs(((xnew-Xi)/xnew));

if n==1

initial\_error=error;

end

if error>initial\_error

disp('No other roots for this point');

break;

end

end

if (error<=Err\_tol)

Xi=xnew;

break;

end

Xi=xnew;

% disp ([' Mi Xi ahora va a valer ' num2str(Xi)]);

end

solutions(2,x)=Xi;

Iterations(2,x)=n;

Err\_vals(2,x)=error;

disp(['My approximation of X= ' num2str(Xi) ' with error of ' num2str(error) '% after iteration : ' num2str(n)]);

error=10000;

end

% second part x=-2 and x=3.

Xi\_2\_values=zeros(2,2);

Xi\_2\_values(1,1)=-2;

Xi\_2\_values(2,1)=-2;

Xi\_2\_values(1,2)=3;

Xi\_2\_values(2,2)=3;

Err\_vals\_2=zeros(2,3);

Iterations\_2=zeros(2,3);

solutions\_2=zeros(2,3);

disp('Last point');

for x=1:1:2

Xi= Xi\_2\_values(1,x);

disp(['First root for ' num2str(Xi) ' : ']);

for n=0:1:100

num1=f(Xi);

num2=df(Xi);

xnew= Xi- (num1/num2)\*constant;

if(n>=1)

error= abs(((xnew-Xi)/xnew));

end

if (error<=Err\_tol)

Xi=xnew;

break;

end

Xi=xnew;

% disp ([' Mi Xi ahora va a valer ' num2str(Xi)]);

end

solutions\_2(1,x)=Xi;

Iterations\_2(1,x)=n;

Err\_vals\_2(1,x)=error;

disp(['My approximation for X= ' num2str(Xi) ' with error of ' num2str(error) '% after iteration : ' num2str(n)]);

error=10000;

end

for x=1:1:2

Xi= Xi\_2\_values(1,x);

disp(['Second root for ' num2str(Xi) ' : ']);

dont\_print=false;

for n=0:1:50000 % this number of cycles are because of constant2. We divided by 10,000 our original value. Therefore, 10,000 times more cycles.

num1=f(Xi);

num2=df(Xi);

if n==0

if num2>=0

sign=-1;

else

sign=1;

end

end

if(num2>0&& sign==-1) || (num2<0 && sign==1)

constant=-0.5;

constant2=-0.000154; % this value is obtained by doing Xmax-X where xmax= 0.016254 and x=0.0161

else

constant=0.5;

constant2=0.000154;

end

if x==2

xnew= Xi-(num1/num2)\*constant2;

else

xnew= Xi - (num1/num2)\*constant;

end

if(n>=1)

error= abs(((xnew-Xi)/xnew));

if n==1

initial\_error=error;

end

if error>initial\_error

disp('No other roots for this point');

dont\_print=true;

break;

end

end

if (error<=Err\_tol)

Xi=xnew;

break;

end

Xi=xnew;

% disp ([' Mi Xi ahora va a valer ' num2str(Xi)]);

end

solutions\_2(2,x)=Xi;

Iterations\_2(2,x)=n;

Err\_vals\_2(2,x)=error;

if dont\_print==false

disp(['My approximation for X= ' num2str(Xi) ' with error of ' num2str(error) '% after iteration : ' num2str(n)]);

end

error=10000;

end

**Problem 3 (Matlab code):**

% Alberto Pascal A01023607

% Saúl E. Labra A01020725

% Rodrigo García A01024595

% Manuel Guadarrama A01020829

clc;

clear all;

close all;

%Original function and its first and second derivative

f = @(x) x.^4 - 10.\*x.^3 + 27.\*x.^2 - 2.\*x - 40;

df = @(x) 4.\*x.^3 - 30.\*x.^2 + 54.\*x - 2;

%Error tolerance for NR method and initial error before execution

err\_tol = 0.0001;

error = 100;

%Definition of the interval to find roots (-3,4)

lowest=-3;

highest=4;

%Stores the values where the function is going to be evaluated (intervals of 1)

values = lowest-1:1:highest;

%Stores the initial points for Newton Raphson

Xnew\_arr=zeros(1,4);

%The first X for Newton Raphson will be the lower limit

Xnew\_arr(1,1)=lowest;

%Solution stores the roots

solutions=zeros(1,4);

count\_sol=1;

count=2;

first=true;

next=true;

%Cycle moves along the function in intervals of 1 from 2 (because code

%requires a "n-1" position in the array values) till the end of the array

%values

for n=2:1:length(values)

%stores the previous and current iteration derivative of the function

store\_prev = df(values(n-1));

store\_curr=df(values(n));

%In case there isn't a change of sign

if (store\_prev<=0 && store\_curr<=0)

end

if (store\_prev >=0 && store\_curr>=0)

end

%In case there's a change of sign and it is the first value of the

%interval

if(store\_prev >=0 && store\_curr<=0) || (store\_prev <=0 && store\_curr>=0) && first==false

first=true;

next=false;

end

%In case there's a change of sign and is the first point of the

%interval

if(store\_prev >=0 && store\_curr<=0) || (store\_prev <=0 && store\_curr>=0&& next==false)

%Calulates the point to evaluate Newton Raphson with an average of

%the interval

num\_add=((values(n-1) + values(n))/2) ;

%If the new value is lower than the lower limit of the interval

if(num\_add<lowest)

num\_add=lowest;

end

%If the new value is greater than the upper limit of the interval

if(num\_add>highest)

num\_add=highest;

end

%Stores the position in X where Newton Raphson is going to be

%calculated

Xnew\_arr(count)= num\_add;

count=count+1;

end

end

%Get the first root fo the function

%Cycle with 20 as the limit of iterations

for x=1:1:count-1

Xi=Xnew\_arr(x);

for i=0:1:20

num1 = f(Xi);

num2 = df(Xi);

Xnew = Xi - (num1/num2);

%After the first iteration rel. Error can be calculated

if(i>=1)

error = abs((Xnew-Xi)/Xnew);

end

%When the error fulfills the tolerance, the cycle breaks

if(error<=err\_tol)

break;

end

%The new value of Xi is assigned

Xi = Xnew;

disp (['It.#: ' num2str(i) ' Xi = ' num2str(Xi) ' with error of ' num2str(error)]);

end

%The value of the root is dispalyed

solutions(count\_sol)=Xnew;

count\_sol=count\_sol+1;

disp (['The root ' num2str(x) ' is ' num2str(Xnew) ' with error of ' num2str(error)]);

error=10000;

end

1. Saúl Enrique Labra Cruz A01020725, Instituto Tecnológico de Estudios Superiores de Santa Fé, Av. de los Poetas 100, Santa Fe, La Loma, 01389, Ciudad de México CDMX (A01020725@itesm.mx).

   Alberto Pascal Garza A01023607, Instituto Tecnológico de Estudios Superiores de Santa Fé, Av. de los Poetas 100, Santa Fe, La Loma, 01389, Ciudad de México CDMX (A01023607@itesm.mx).

   Rodrigo García Mayo A01024595, Instituto Tecnológico de Estudios Superiores de Santa Fé, Av. de los Poetas 100, Santa Fe, La Loma, 01389, Ciudad de México CDMX (A01024595@itesm.mx).

   Manuel Guadarrama Villarroel A01020829, Instituto Tecnológico de Estudios Superiores de Santa Fé, Av. de los Poetas 100, Santa Fe, La Loma, 01389, Ciudad de México CDMX (A01020829@itesm.mx). [↑](#footnote-ref-0)