Laboratory 1: Finding minima of functions

Ruaidhrí Campion
SF Theoretical Physics
19333850

3rd March 2021

Contents

1 Introduction

The aim of this laboratory was to numerically calculate the roots of functions using two different methods: the bisection method and the Newton-Raphson method.

1.1 The Bisection Method

The first method used in this laboratory was the bisection method. This method relies on Bolzano's Theorem[1], which states that, for a continuous function f, if $f(a)$ and $f(b)$ have opposite signs, then there must lie a root in between a and b, i.e. $f(x) = 0$ for some $x \in (a, b)$. For the bisection method to work, one must choose points x_1 and x_3 such that $f(x_1)$ is negative and $f(x_3)$ is positive, and calculate the midpoint x_2 of these points using the equation

$$x_2 = \frac{x_1 + x_3}{2}. \tag{1}$$

If $f(x_2) = 0$, then x_2 is a root, and no further calculation is required. If $f(x_2) < 0$, then x_2 replaces x_1. If $f(x_2) > 0$, then x_2 replaces x_3. In the cases where $f(x_2) \neq 0$, x_2 is found for the new values of x_1 and x_3 using equation 1. This process is repeated until $f(x) \approx 0$, within a specified tolerance. This method was used in Exercise 1 to find the roots of the equation $f(x) = 2x^2 - 10x + 12$.

1.2 The Newton-Raphson Method

The second method used in this laboratory was the Newton-Raphson method. This method relies on the Taylor series expansion.[2] The series

$$f(a + x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} x^n \tag{2}$$

expresses a continuous function about a given point a as a polynomial in terms of its derivatives at that point. This infinite sum is not an approximation, however it can be truncated at the linear term in x to approximate the function

$$\begin{aligned} f(a + x) &= f(a) + x\, f'(a) + \mathcal{O}\left(x^2\right) \\ &\approx f(a) + x\, f'(a). \end{aligned} \tag{3}$$

Letting $a + x$ be a root results in $f(a + x) = 0$. Rearranging equation 3 leads to

$$x \approx -\frac{f(a)}{f'(a)}. \tag{4}$$

Since $a + x$ is a root of the function, adding a to each side of equation 4 results in an expression for the root

$$a + x \approx a - \frac{f(a)}{f'(a)}. \tag{5}$$

[1] Bolzano, 1817
[2] Taylor, 1715

For the Newton-Raphson method, an initial guess of the root a is chosen and the approximate root is calculated using equation 5. This approximate root takes the new value of a, and the process is repeated until $f(a) \approx 0$, within a specified tolerance.

This method can also be used to find maximums/minimums of a function. If $f'(a) = 0$, then a is a local maximum or minimum of the function. Equation 5 can be changed to calculate the max/min by simply finding the root of $f'(x)$

$$a + x \approx a - \frac{f'(a)}{f''(a)}. \tag{6}$$

This method was used in Exercise 2 to find the roots of the equation $f(x) = 2x^2 - 10x + 12$, and in Exercise 3 to find the bond length of the interaction potential between two ions by finding the minimum of $V(x) = A\,e^{-\frac{x}{\rho}} - \dfrac{e^2}{4\pi\,\varepsilon_0\,x}$, the potential as a function of separation between the ions.

2 Methodology

2.1 Exercise 1: Finding Roots of a Parabolic Function - Bisection Method

1. For this exercise, the parabola $f(x) = 2x^2 - 10x + 12$ was chosen.
2. Initial points x_1 and x_3 were then defined. If either $f(x_1) > 0$ or $f(x_3) < 0$, then the user was asked to change x_1 and x_3. If either x_1 or x_3 were already roots, then the programme stopped. This was achieved using the following `if/elif` statements:

```
from sys import exit
if f(x1) > 0 or f(x3) < 0:
    print("Programme not run.  Change x1 and x3 \
    so that f(x1) < 0 and f(x3) > 0.")
    exit(0)
elif f(x1) == 0 and f(x3) == 0:
    print("x1 =", x1, "and x3 =", x3, "are roots.")
    exit(0)
elif f(x1) == 0:
    print("x1 =", x1, "is a root.")
    exit(0)
elif f(x3) == 0:
    print("x3 =", x3, "is a root.")
    exit(0)
```

3. x_2 was calculated using equation 1, and replaced either x_1 or x_3 based on its sign.
4. The function was plotted, including the midpoint $(x_2, f(x_2))$ after this iteration.

5. Step 3 was repeated a number of times until $f(x_2) \approx 0$, within a specified tolerance, and the number of steps to do so was counted. This was achieved using the following `while` loop:

```
nsteps = 1
while abs(f(x2)) > tol:
    x2 = (x1 + x3) / 2
    if f(x2) < 0:
        x1 = x2
    elif f(x2) > 0:
        x3 = x2
    nsteps += 1
```

   Here, `nsteps` was set to 1, as one iteration had already been carried out.

6. Steps 2-5 were repeated to approximate the other root of the parabola, using different initial values x_1 and x_3.

7. A function was defined to calculate the number of iterations carried out to find a root of the parabola, with tolerance, x_1 and x_2 as parameters. This function was plotted against a range of tolerances, for given x_1 and x_3.

2.2 Exercise 2: Finding Roots of a Parabolic Function - Newton-Raphson Method

1. The parabola used in Exercise 1 was also chosen for this exercise, with derivative $f'(x) = 4x - 10$.

2. x_1 was initialised. If $f'(x) = 0$, then the user was asked to change x_1, as this would result in a singularity using equation 5. If x_1 was already a root, then the programme stopped. This was achieved using the following `if`/`elif` statements:

```
from sys import exit
if f_prime(x1) == 0:
    print("Programme not run.  Choose an x1 that \
    is not a minimum or maximum of the function.")
    exit(0)
elif f(x1) == 0:
    print("x1 =", x1, "is a root.")
    exit(0)
```

3. One iteration of the Newton-Raphson method was carried out, setting x_1 to the value of $x_1 - \dfrac{f(x_1)}{f'(x_1)}$.

4. The function was plotted, including the point $(x_1, f(x_1))$ after this iteration.

5. Step 3 was repeated a number of times until $f(x_1) \approx 0$, within a specified tolerance, and the number of steps to do so was counted. This was achieved using the following `while` loop:

```
nsteps = 1
while abs(f(x1)) > tol:
    x1 = x1 - f(x1) / f_prime(x1)
    nsteps += 1
```

As in Exercise 1, `nsteps` was set to 1 as one iteration had already been carried out.

6. Steps 3-5 were repeated to approximate the other root of the parabola, using a different initial value x_1.

7. A similar function to that in Exercise 1 Step 7 was defined to calculate the number of iterations required to approximate a root using the Newton-Raphson method, with tolerance and x_1 as parameters. This was plotted against the same range of tolerances, for given x_1.

2.3 Exercise 3: Finding Minima Roots of a Potential Energy Function

1. The functions $V(x) = A\,e^{-\frac{x}{p}} - \dfrac{e^2}{4\pi\,\varepsilon_0\,x}$ and $F(x) = -V'(x) = \dfrac{A}{p}\,e^{-\frac{x}{p}} - \dfrac{e^2}{4\pi\,\varepsilon_0\,x^2}$ were defined, with x in nm, V in eV and F in eV nm^{-1}. $\dfrac{e^2}{4\pi\,\varepsilon_0}$, A and p were set as 1.44 eV nm, 1090 eV and 0.033 nm, respectively, so that $V(x)$ was the interaction potential between Na^+ and Cl^- as a function of the seperation between the ions, and $F(x)$ was the force acting on the ions as a function of the separation. These functions were plotted against a range of x values.

2. The functions $V'(x) = -F(x)$ and $V''(x) = \dfrac{A}{p^2}\,e^{-\frac{x}{p}} - \dfrac{e^2}{2\pi\,\varepsilon_0\,x^3}$ were defined.

3. The Newton-Raphson method was implemented to find the bond length of the ions, i.e. the minimum of $V(x)$, using equation 6 with a specified tolerance, using a similar `while` loop as in Exercise 2 Step 5.

3 Results

3.1 Exercise 1: Finding Roots of a Parabolic Function - Bisection Method

For the first root, x_1 and x_3 were set as 2.5 and 1.0, respectively. After one iteration, the midpoint $x_2 = 1.75$ was set as x_3. After 14 steps, the root was approximated to be 2.000030517578125, within a tolerance of 0.0001. For the second root, x_1 and x_3 were set as 2.5 and 4.0, respectively. After one iteration, the midpoint $x_2 = 3.25$ was set as x_3. After 14 steps, the root was approximated to be 2.999969482421875, within a tolerance of 0.0001.

The following graphs of the parabola, initial points x_1 and x_3 and midpoint x_2 were obtained for each set of initial values:

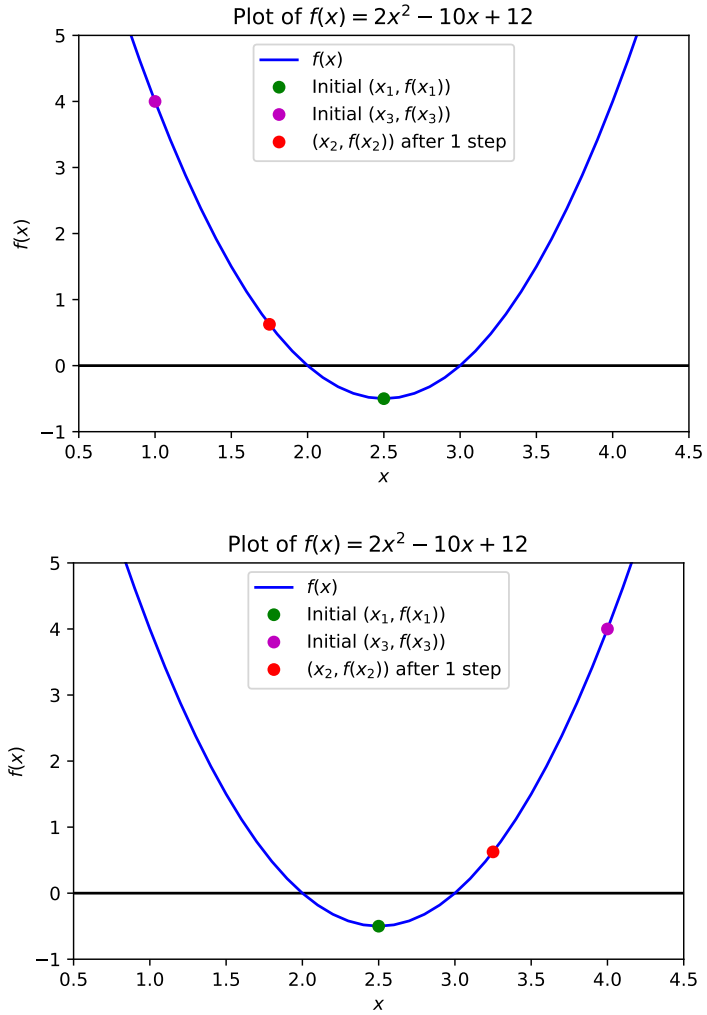
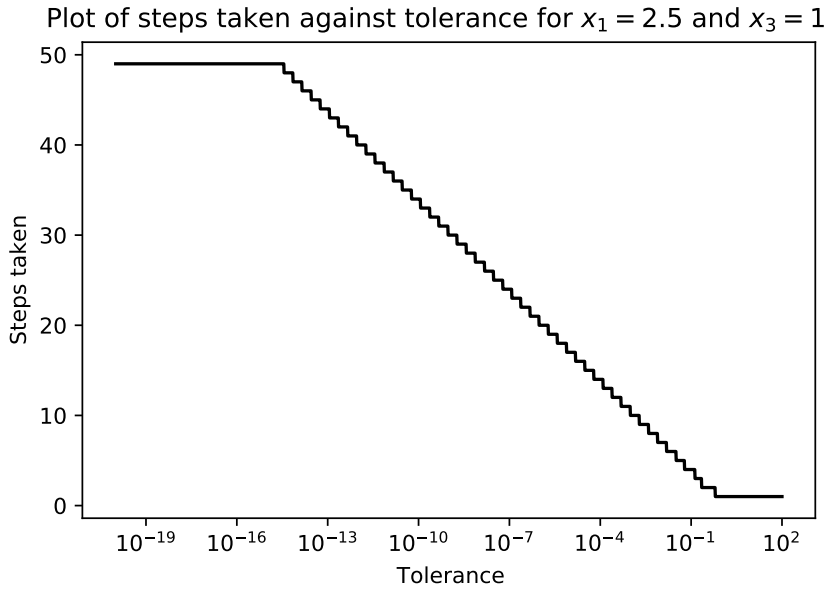


Figure 1: Graphs of $f(x)$ and x_2 after one iteration, for each set of initial x_1 and x_3.

The number of steps to find a root using the bisection method was plotted against a range of tolerances from 10^{-20} to 100, for initial $x_1 = 2.5$ and $x_3 = 1.0$. The following graph and data was obtained:



Tolerance	Steps taken
10^{2}	0
10^{1}	0
10^{0}	0
10^{-1}	4
10^{-2}	7
10^{-3}	10
10^{-4}	14
⋮	⋮
10^{-13}	44
10^{-14}	47
10^{-15}	49
10^{-16}	49
⋮	⋮
10^{-20}	49

```
x = []
for i in np.linspace(-20.0, \
2.0, 2300, endpoint = True):
    x.append(10.0 ** i)
y = []
for j in x:
    y.append(steps(j, 2.5, 1.0))
plt.plot(x, y)
plt.xscale("log")
```

Figure 2: Graph of steps taken to reach the approximate root against the allowed tolerance for the approximate root, a table of the points on the graph, and a summarised version of the code used to plot the graph, for $x_1 = 2.5$ and $x_3 = 1.0$. `steps` was defined to be a function that found the number of steps taken to reach an approximate root of $f(x)$, with tolerance, x_1 and x_3 as parameters.

3.2 Exercise 2: Finding Roots of a Parabolic Function - Newton-Raphson Method

For the first root, x_1 was set as 1.0. After one iteration, x_1 was set as 1.666666666666665. After 4 steps, the root was approximated to be 1.9999847409781035, within a tolerance of 0.0001. For the second root, x_1 was set as 4.0. After one iteration, x_1 was set as 3.3333333333333335. After 4 steps, the root was approximated to be 3.000015259021897, within a tolerance of 0.0001.

The following graphs of the parabola, initial point x_1 and x_1 after one iteration were obtained for each initial point:

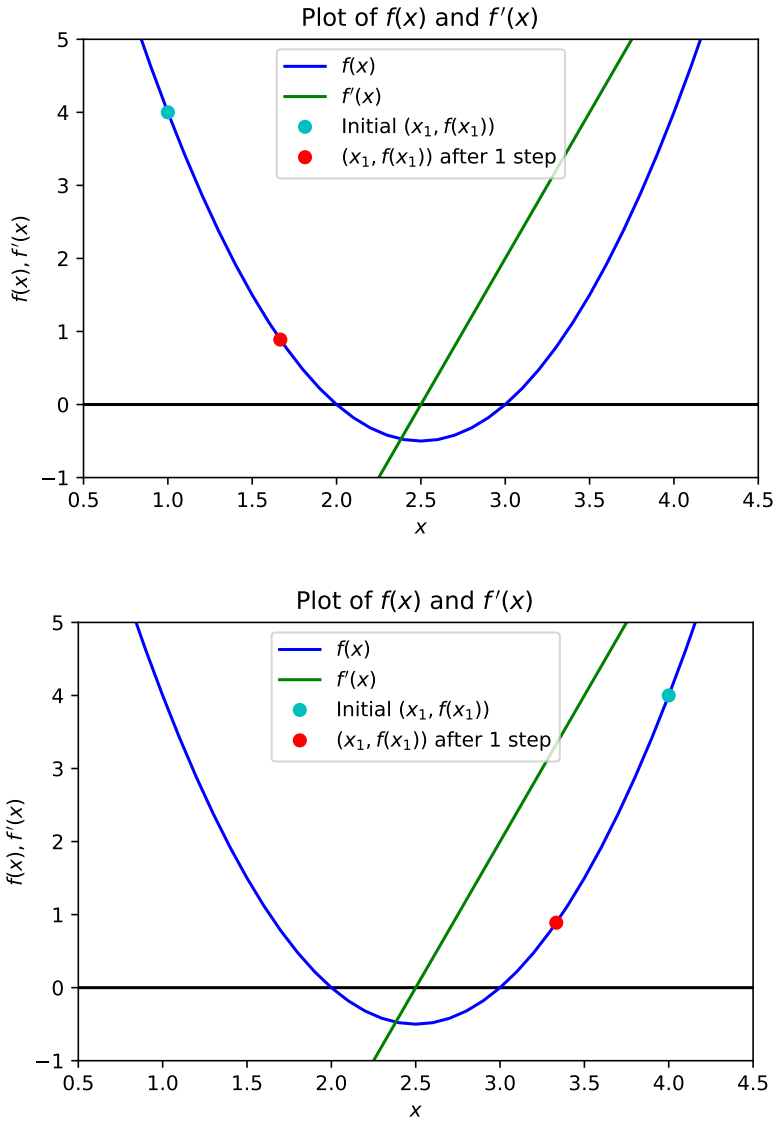
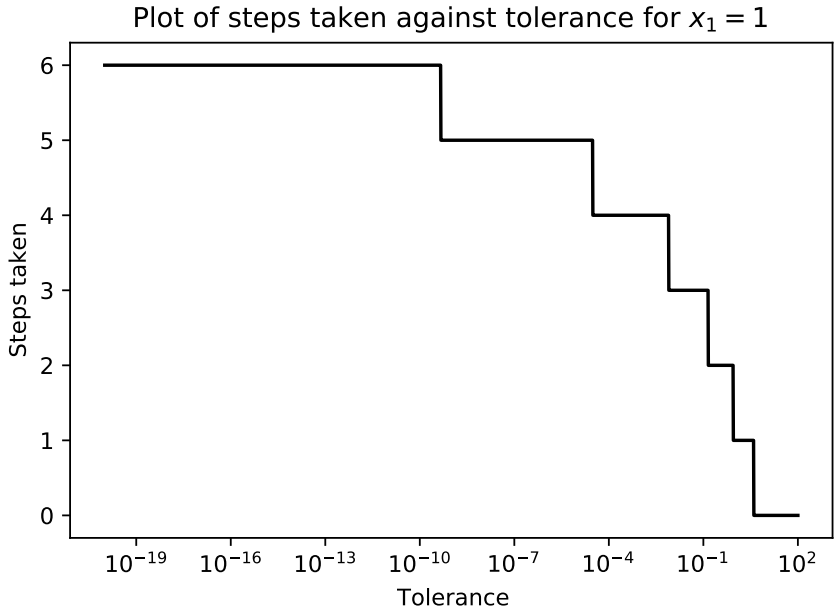


Figure 3: Graphs of $f(x)$, $f'(x)$ and x_1 after one iteration, for each initial x_1.

The number of steps to find a root was plotted against a range of tolerances from 10^{-20} to 100, for initial $x_1 = 1.0$. The following graph and data was obtained:



Tolerance	Steps taken
10^{2}	0
10^{1}	0
10^{0}	1
10^{-1}	3
10^{-2}	3
10^{-3}	4
10^{-4}	4
10^{-5}	5
⋮	⋮
10^{-9}	5
10^{-10}	6
⋮	⋮
10^{-20}	6
⋮	⋮
10^{-100}	6

```
x = []
for i in np.linspace(-20.0, \
2.0, 2300, endpoint = True):
    x.append(10.0 ** i)
y = []
for j in x:
    y.append(steps(j, 1.0))
plt.plot(x, y)
plt.xscale("log")
```

Figure 4: Graph of steps taken to reach the approximate root against the allowed tolerance for the approximate root, a table of the points on the graph, and a summarised version of the code used to plot the graph, for $x_1 = 1.0$. `steps` was defined to be a function that found the number of steps taken to reach an approximate root of $f(x)$, with tolerance and x_1 as parameters.

3.3 Exercise 3: Finding Minima Roots of a Potential Energy Function

x_1 was set to be 0.24 nm as the initial guess of the corresponding minimum of $V(x)$. After 4 steps, the bond length was approximated to be 0.23605384841577942 nm, within a tolerance of 10^{-14} eV.

The following graphs of $V(x)$ and $F(x)$, including $(0.24, (V(0.24))$ and $(0.24, F(0.24))$ respectively, were obtained:

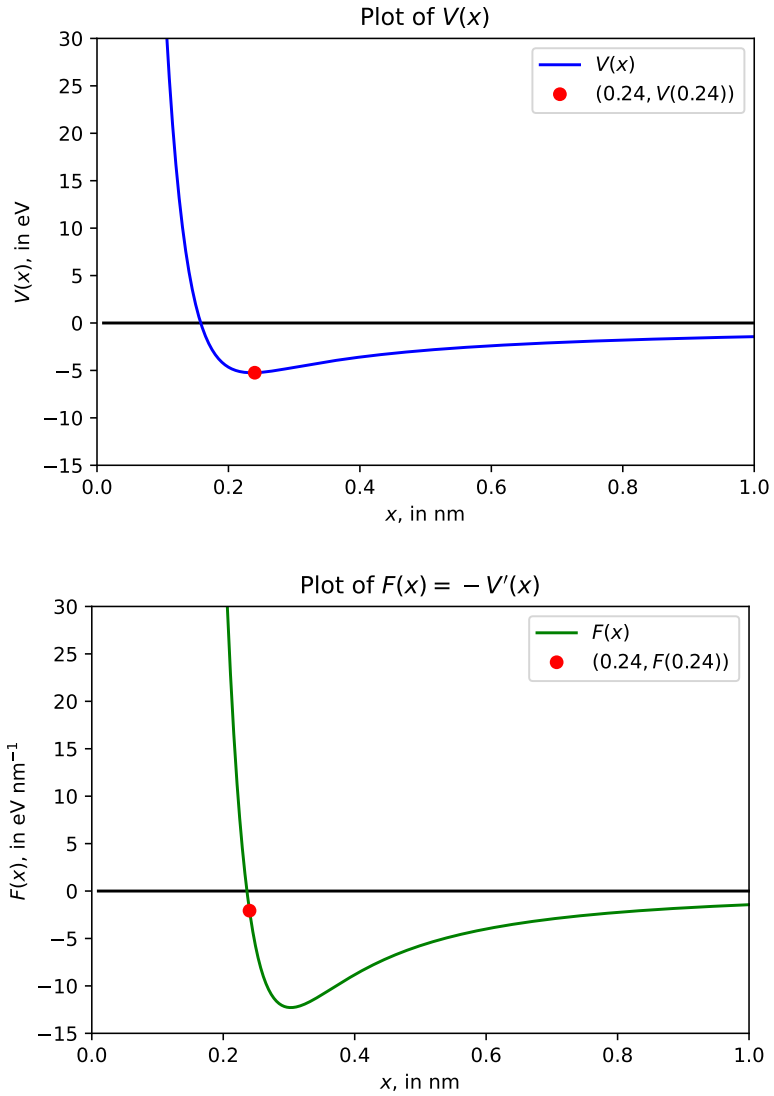


Figure 5: Graphs of $V(x)$ and $F(x)$, including the initial guess of the minimum $x_1 = 0.24$ nm. Since the derivative of a function evaluated at a minimum or maximum of $V(x)$ is 0 eV, and $F(x) = -V'(x)$, then $F(x)$ at the minimum of $V(x)$ will also be 0 eV. It is thus sufficient to consider $F(x)$ at the minimum of $V(x)$.

4 Discussion

4.1 Precision of Approximate Roots

From Figures 2 and 4, the number of steps required to approximate the analytic root $x = 2$ for $f(x) = 2x^2 - 12x + 10$ using both the bisection method and the Newton-Raphson method seemingly reached a maximum. These maximum values were attained at a tolerance of approximately 10^{-15}. For Exercise 3, if the tolerance was taken to be 10^{-15} eV, the programme was stuck in an infinite loop, and never attained an approximate root within this tolerance. This is due to the fact that Python has a limit on the number of digits a float can have. If this limit was increased, then both methods would take more steps for tolerances smaller than 10^{-15} to approximate the same root for Exercises 1 and 2, and a more precise approximate root would be calculated for Exercise 3.

4.2 Bisection Method vs Newton-Raphson Method Efficiency

The following figure is a graph of the efficiency of both methods approximating the same root:

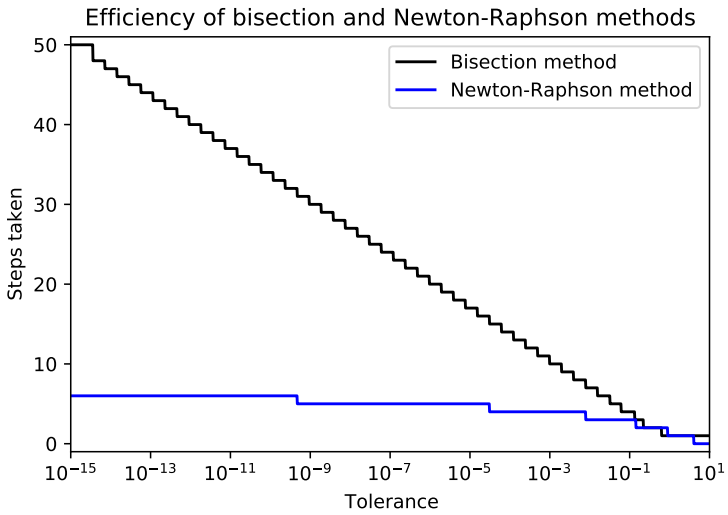


Figure 6: Graph of number of steps required to approximate the analytic root $x = 2$ for $f(x) = 2x^2 - 10x + 12$, for both the bisection method and Newton-Raphson method.

For tolerances greater than 0.1, the two methods require a similar number of steps to approximate a root. However, the Newton-Raphson method needs far fewer steps when considering a tolerance less than 0.1. Clearly, for precise approximations, the Newton-Raphson method is much more efficient at approximating roots than the bisection method.

4.3 Conditions on Initial Points

For the Newton-Raphson method, from equation 5, the derivative of the function evaluated at the initial guess cannot be 0, i.e. the initial guess cannot be a maximum or minimum, or else the approximate root after one iteration would be a singularity.

The Newton-Rapshon method also requires the initial guess to be on the "same side" as the root with respect to a maximum or minimum; that is, for example, if the actual root is less than a certain minimum or maximum, then to approximate this root, the initial guess must also be less than this minimum or maximum. For Exercise 3, if the initial guess was taken to be 0.4 nm, which is clearly greater than the maximum of $V'(x)$ (as it is clearly greater than the minimum of $F(x) = -V'(x)$), then for a tolerance of 10^{-14} eV, the approximate root of $V'(x)$ is 15,502,127.088146694 nm $\approx$ 1.55 cm. Even though $V'(x)$ evaluated at this point is indeed approximately 0 eV within the specified tolerance, this point is not a root, as $V'(x)$ asymptotes to 0 eV as x tends to infinity, but never reaches 0 eV. Thus for any non-zero tolerance, the Newton-Raphson method can result in a value that is not actually a root, if certain initial guesses are chosen.

The only condition for the initial points of the bisection method is that they have opposite sign. Although the Newton-Raphson method is more efficient than the bisection method, it is comparably easier to choose compatible initial points for the bisection method than for the Newton-Raphson method.

5 Conclusions

Both the bisection method and Newton-Raphson method were successful in approximating roots of a parabola. For a tolerance of 0.0001, the approximations ranged from 1.9999847409781035 to 2.00003051757812 for an analytic root of 2, and from 2.999969482421875 to 3.000015259021897 for an analytic root of 3. For these roots, the bisection method required 14 steps, and the Newton-Raphson method required 4 steps. For tolerances as low as to 10^{-15}, the bisection method required as many as 49 steps, and the Newton-Raphson method required up to 6 steps. The Newton-Raphson method was also successful in approximating the bond length between two ions. The bond length between Na^+ and Cl^- was found to be 0.23605384841577942 nm, within a tolerance of 10^{-14} eV.

Approximations for roots were found to be largely dependent on the precision in which figures are considered in calculations.

The Newton-Raphson method was found to be far quicker than the bisection method in finding a root of a function. Despite this, it is much simpler to meet the requirements of the initial points needed for the bisection method than that of the guess needed for the Newton-Raphson method.

6 References & Appendix

1. B. Bolzano, *Rein Analytischer Beweis...*, Gottlieb Haase, Prague, 1817.
2. B. Taylor, *Methodus Incrementorum Directa & Inversa*, London, 1715.
All code used in this laboratory can be found here:
`https://github.com/campioru/SF_Lab_1`