COMP 361: Elementary Numerical Methods
Assignment no.3

Duc Nguyen

Gina Cody School of Computer Science and Software Engineering
Concordia University, Montreal, QC, Canada

Winter 2020

Contents

NOTE:
The folder <program> attached in the same directory contains every program for the assignment. Consult the file README.md inside the folder for build guide and list of dependencies.
The programs were written in Python with Jupyter Notebook.

1 Newton's method to compute the cube root of 6

Problem: Show how to use Newtons method to compute the cube root of 5. Numerically carry out the first 10 iterations of Newtons method, using $x_0 = 1$. Analytically determine the fixed points of the Newton iteration and determine whether they are attracting or repelling. If a fixed point is attracting then determine analytically if the convergence is linear or quadratic. Draw the x_{k+1} versus x_k diagram, again taking $x_0 = 1$, and draw enough iterations in the diagram, so that the long time behavior is clearly visible. For which values of x_0 will Newtons method converge?
Solution:

1.1 Iterations using Newton's method

Cube root of 5 is a zero of the function g (x) such that:
$g(x) = x^3 - 5$

The first 10 iterations using Newton's method were carried out by using Python with Jupyter Notebook. Check out the full source code and presentation in directory: *program/Problem1.ipynb* Below is the main algorithm:

```
# The main algorithm
def newton_raphson(f, diff, init_x, max_iter=1000):
    x = init_x
    estimates = []
    listX = [x]
    for i in range(max_iter):
        deltaX = -f(x)/diff(x)
        x = x + deltaX
        listX.append(x)
        estimates.append(x)
    return x, listX, estimates
```

The results after computing are as following:

Iteration no. 1 Xi = 1 X(i+1) = 2.333333333333333
Iteration no. 2 Xi = 2.333333333333333 X(i+1) = 1.8616780045351473
Iteration no. 3 Xi = 1.8616780045351473 X(i+1) = 1.722001880058607
Iteration no. 4 Xi = 1.722001880058607 X(i+1) = 1.7100597366002945
Iteration no. 5 Xi = 1.7100597366002945 X(i+1) = 1.709975950782189
Iteration no. 6 Xi = 1.709975950782189 X(i+1) = 1.709975946676697
Iteration no. 7 Xi = 1.709975946676697 X(i+1) = 1.709975946676697
Iteration no. 8 Xi = 1.709975946676697 X(i+1) = 1.709975946676697
Iteration no. 9 Xi = 1.709975946676697 X(i+1) = 1.709975946676697
Iteration no. 10 Xi = 1.709975946676697 X(i+1) = 1.709975946676697

1.2 Determine fixed points

Newton's method: $x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}$

Applying Newton's method

$$\implies f(x) = x - \frac{x^3 - 5}{3x^2} = \frac{3x^3 - x^3 + 5}{3x^2} = \frac{2x^3 + 5}{3x^2}$$

To find the fixed points, let x = f (x)

$$\begin{aligned} x &= \frac{2x^3 + 5}{3x^2} \\ 3x^3 &= 2x^3 + 5 \\ x^3 &= 5 \\ \implies x &= \sqrt[3]{5} \end{aligned}$$

The fixed point of the Newton iteration is $x = \sqrt[3]{5}$

1.3 Fixed point analytically

$$f'(x) = (\frac{2x^3 + 5}{3x^2})' = (\frac{2}{3}x + \frac{5}{3x^2})' = \frac{2}{3} - \frac{10}{3x^3}$$

$$\implies |f'(x^*)| = |f'(\sqrt[3]{5})| = |\frac{2}{3} - \frac{10}{3(\sqrt[3]{5})^3}| = |\frac{2}{3} - \frac{10}{15}| = 0$$

$|f'(x^*)| = 0 \implies$ the fixed point is **attracting** and it is converging **quadratically**.

1.4 Fixed point iteration diagram

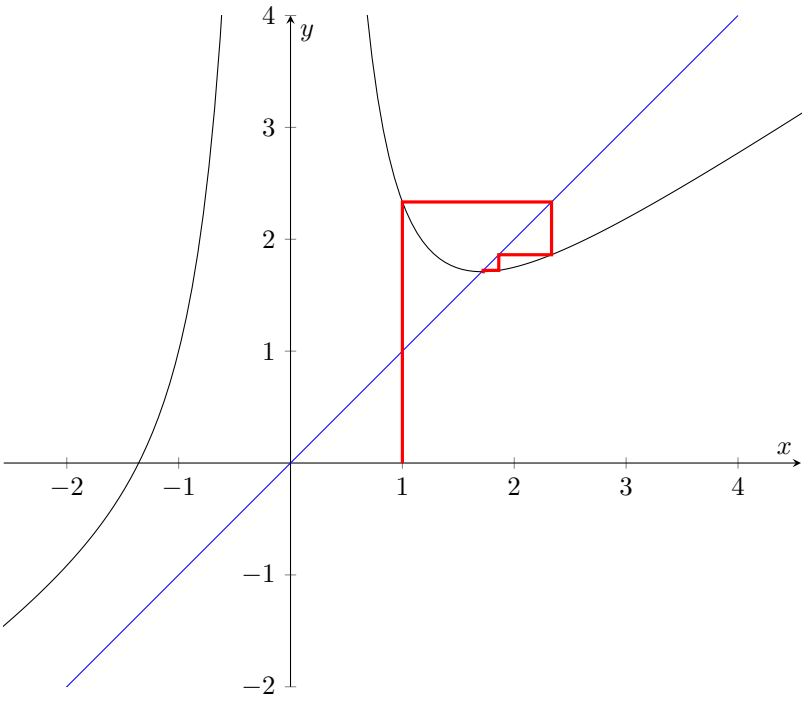

Figure 1: Fixed point iteration diagram

1.5 Newton's method converges

The fixed point is always attracting regardless of the value of x^0; thus the Newton's method always converges at the fixed point.

2 Chord method to compute the cube root of 5

Problem: Also use the Chord method to compute the cube root of 5. Numerically carry out the first 10 iterations of the Chord method, using $x^0 = 1$. Analytically determine the fixed points of the Chord iteration and determine whether they are attracting or repelling. If a fixed point is attracting then determine analytically if the convergence is linear or quadratic. If the convergence is linear then determine analytically the rate of convergence. Draw the x^{k+1} versus x^k diagram, as in the Lecture Notes, again taking $x^0 = 1$, and draw enough iterations in the diagram, so that the long time behavior is clearly visible. (If done by hand then make sure that your diagram is sufficiently accurate, for otherwise the graphical results may be misleading.)
Do the same computations and analysis for the Chord Method when $x^0 = 0.1$. More generally, analytically determine all values of x^0 for which the Chord method will converge to the cube root of 5.
Solution:

2.1 With $x_{(0)} = 1$

2.1.1 Numerically carry out first 10 iterations

Check out the full source code and presentation in directory: *program/Problem2.ipynb* Below is the main algorithm:

```
# Main algorithm
def chord(f, diff, init_x, max_iter=1000):
    x = init_x
    estimates = []
    listX = [x]
    for i in range(max_iter):
        deltaX = -f(x)/diff(init_x)
        x = x + deltaX
        listX.append(x)
        estimates.append(x)
    return x, listX, estimates
```

The results are as following:

Xi = 1 X(i+1) = 2.333333333333333

Xi = 2.333333333333333 X(i+1) = -0.23456790123456672

Xi = -0.23456790123456672 X(i+1) = 1.4364009049609154

Xi = 1.4364009049609154 X(i+1) = 2.115184017622358

Xi = 2.115184017622358 X(i+1) = 0.6274038354390186

Xi = 0.6274038354390186 X(i+1) = 2.2117476794083464

Xi = 2.2117476794083464 X(i+1) = 0.271918086443556

Xi = 0.271918086443556 X(i+1) = 1.9318829289112252

Xi = 1.9318829289112252 X(i+1) = 1.1951766954547978

Xi = 1.1951766954547978 X(i+1) = 2.2927610409500208

2.1.2 Determine the fixed points

Cube root of 5 is a zero of function g (x) Thus, let $g(x) = x^3 - 5$

Definition 1. ***Chord method*** $x^{(k+1)} = x^k - \frac{g(x^k)}{g'(x^{(0)})}$

Apply Chord method with g (x) and $x_0 = 1$:

$$f(x) = x - \frac{x^3 - 5}{3x_0^2} = x - \frac{x^3 - 5}{3} = \frac{3x - x^3 + 5}{3}$$

To find the fixed point of f (x), let x = f (x)

$$\begin{aligned} x &= f(x) \\ \implies x &= \frac{3x - x^3 + 5}{3} \\ \implies 3x &= 3x - x^3 + 5 \\ \implies 5 - x^3 &= 0 \\ \implies x &= \sqrt[3]{5} \end{aligned}$$

The fixed point of f (x) is $x = \sqrt[3]{5}$

2.1.3 Analyze fixed points of Chord iteration:

$$\begin{aligned} f'(x) &= (x - \frac{x^3 - 5}{3})' \\ &= 1 - x^2 \\ \implies |f'(x_*)| &= |1 - x_*^2| = |1 - (\sqrt[3]{5})^2| = 1.924 > 1 \end{aligned}$$

$|f'(x_*)| > 1$; thus the fixed point is **diverge**

2.1.4 Fixed point iteration diagram

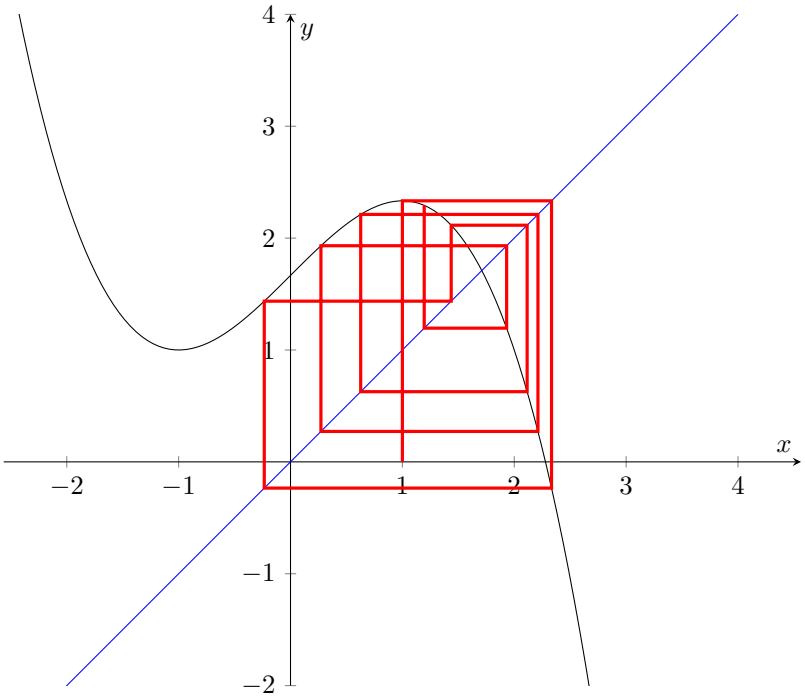

Figure 2: Fixed point iteration no.1 to no. 10 diagram

2.2 With $x^0 = 0.1$

2.2.1 Numerically carry out first 10 iterations

Check out the full source code and presentation in directory: *program/Problem2.ipynb*

The results are as following:

Iteration no. 1 Xi = 0.1 X(i+1) = 166.7333333333333

Iteration no. 2 Xi = 166.7333333333333 X(i+1) = -154505913.52345666

Iteration no. 3 Xi = -154505913.52345666 X(i+1) = 1.229459037686188e+26

Iteration no. 4 Xi = 1.229459037686188e+26 X(i+1) = -6.194709380101413e+79

Iteration no. 5 Xi = -6.194709380101413e+79 X(i+1) = 7.923946873048754e+240

Note: More than iteration 5, the result is too big to be computed and displayed

2.2.2 Determine the fixed points

Let $g(x) = x^3 - 5$

Apply Chord method with g (x) and $x_0 = 0.1$:

$$f(x) = x - \frac{x^3 - 5}{3x_0^2} = x - \frac{x^3 - 5}{0.03} = \frac{0.03x - x^3 + 5}{0.03}$$

To find the fixed point of f (x), let x = f (x)

$$\begin{aligned} x &= f(x) \\ \implies x &= \frac{0.03x - x^3 + 5}{0.03} \\ \implies 0.03x &= 0.03x - x^3 + 5 \\ \implies x^3 - 5 &= 0 \\ \implies x &= \sqrt[3]{5} \end{aligned}$$

The fixed point of f (x) is $x = \sqrt[3]{5}$

2.2.3 Analyze fixed points of Chord iteration:

$$f'(x) = (x - \frac{x^3 - 5}{0.03})'$$

$$||f'(x_*)| = |1 - 100x_*^2| = |1 - 100(\sqrt[3]{5})^2| = 281.401 > 1$$

$|f'(x_*)| > 1$; thus the fixed point is **diverge**

2.2.4 Fixed point iteration diagram

The result is too big that it is insufficient to draw such iteration graph

2.3 Analyze the condition of $x^{(0)}$ to make the Chord method converge

To find the fixed point, the equation is:

$$\begin{aligned} x &= x - \frac{x^3 - 5}{3x_0^2} \\ \implies x^3 - 5 = 0 &\implies x = \sqrt[3]{5} \end{aligned}$$

Conclude: the value of the fixed point does not depend on the value of x_0 in Chord method. It is always $\sqrt[3]{5}$

The Chord method converges when

$$\implies |f'(x_{(*)})| < 1 \implies |(x - \frac{x^3 - 5}{3x_0^2})'| < 1$$

$$\implies |1 - \frac{1}{3x_0^2}(3x^2)| < 1 \implies |1 - \frac{x^2}{x_0^2}| < 1 \textbf{(1)}$$

$$\frac{x^2}{x_0^2} \geq 0 \forall x_0 \neq 0 \textbf{(2)}$$

$$\text{From (1) and (2)} \implies 0 \leq \frac{x^2}{x_0^2} < 1$$

$$\implies 0 \leq \frac{(\sqrt[3]{5})^2}{x_0^2} < 1 \implies \frac{\sqrt[3]{5}^2}{x_0^2} - 1 < 0$$

$$\implies \frac{\sqrt[3]{5}^2 - x_0^2}{x_0^2} < 0 \text{ However, } x_0^2 > 0 \forall x_0 \neq 0$$

$$\implies \sqrt[3]{5}^2 - x_0^2 < 0 \implies x_0^2 > \sqrt[3]{5}^2 \implies x_0 > \sqrt[3]{5}$$

Conclusion: The Chord method converges only when $x_0 > \sqrt[3]{5}$

3 Iteration of discrete logistic equation

Problem:
Consider the *discrete logistic equation*
$x^{k+1} = cx^k(1 - x^k)$, k = 0, 1, 2, 4,...
For each of the following values of c, determine analytically the fixed points and whether they are attracting or repelling: c = 0.70, c = 1.00, c = 1.80, c = 2.00, c = 3.30, c = 3.50, c = 3.97. (You need only consider physically meaningful fixed points, namely those that lie in the interval [0, 1].) If a fixed point is attracting then determine analytically if the convergence is linear or quadratic. If the convergence is linear then analytically determine the rate of convergence. For each case include a statement that describes the behavior of the iterations
Solution:

3.1 General analysis

$$\begin{aligned} f'(x) &= (cx_* - cx_*^2)' \\ &= c - 2cx_* \\ &= c(1 - 2x_*) \end{aligned}$$

To determine the fixed point, let $f(x) = x^* = cx^*(1 - x^*)$

$$\begin{aligned} &\implies cx_*^2 + x_* - cx_* = 0 \\ &\implies x_* = 0 \text{ and } x_* = 1 - \frac{1}{c} \end{aligned}$$

- With the fixed point x = 0, $|f'(x)| = |c(1 - 2 * 0)| = c$. Thus, the fixed point is attracting when $0 \leq c < 1$, and repelling when $1 < c$
- With the fixed point $x = 1 - \frac{1}{c}$, $|f'(1 - \frac{1}{c})| = c[1 - 2(1 - \frac{1}{c})] = 2 - c$. Thus, the function is attracting when $1 < c < 3$, repelling otherwise.

Check out the full source code and presentation in directory: *program/Problem3.ipynb*
Below is the main algorithm:

```
def f(c, x):
    return c*x*(1 - x)

def logistic_equation(f, c, init_x, max_iter=1000):
    x = init_x
    estimates = []
    listX = [x]
    for i in range(max_iter):
        x = f(c, x)
        listX.append(x)
        estimates.append(x)
    return x, listX, estimates
```

3.1.1 c = 0.70

c = 0.70, meaning fixed points of the function are 0 and $1 - \frac{1}{c} = \frac{-3}{7}$

- The fixed point x = 0 is attracting. $|f'(0)| = 0.7 \implies$ the rate of convergence is 0.7
- The fixed point x = 0.7 is repelling (0.7 < 1)

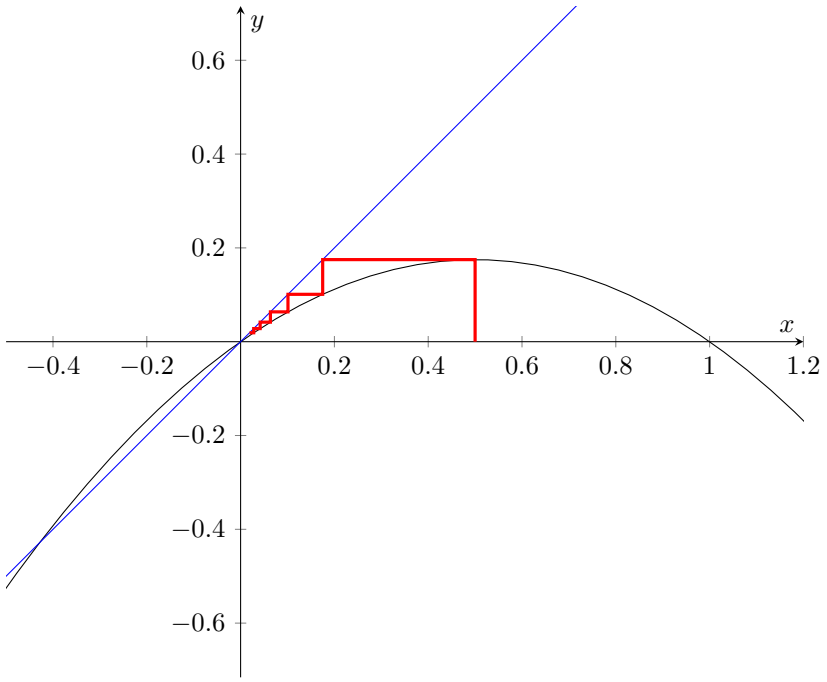

Figure 3: c = 0.7

3.1.2 c = 1.00

- The fixed point x = 0 is attracting. $|f'(0)| = 1 \implies$ the rate of convergence is 1

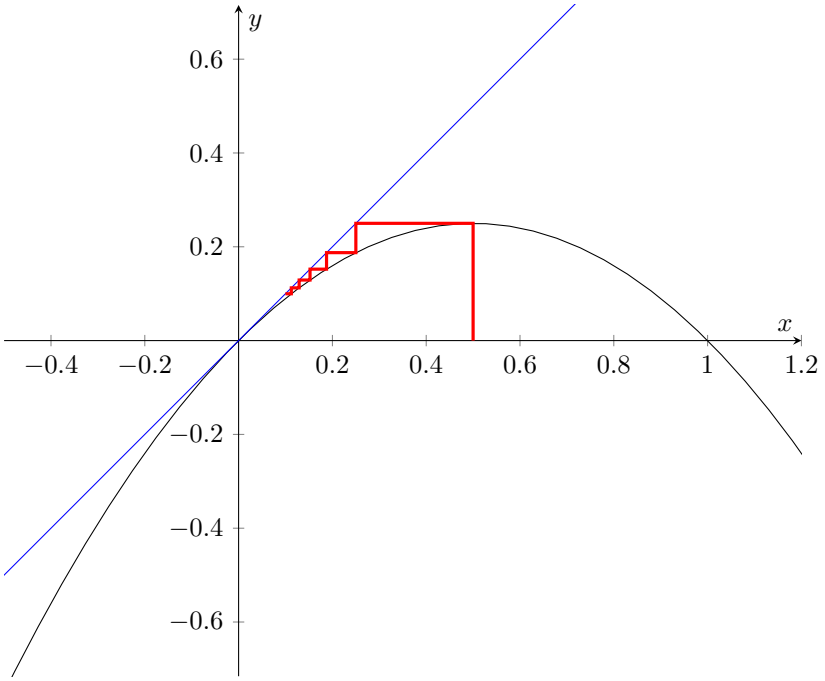

Figure 4: c = 1.00

3.1.3 c = 1.8

- The fixed point x = 0 is repelling. ($|f'(0)| = 1.8$)
- The fixed point x = 0.44 is repelling ($0.44 < 1$)

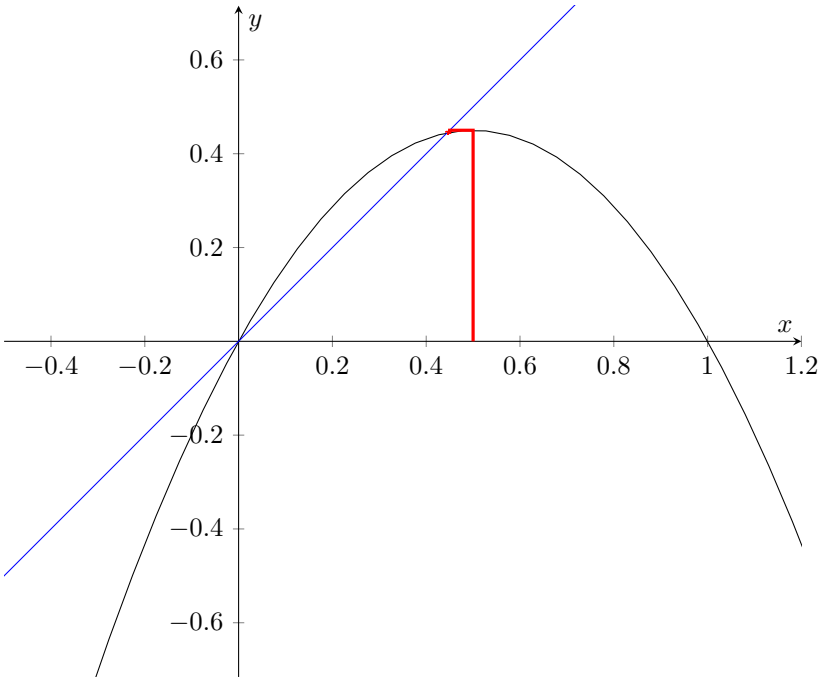

Figure 5: c = 1.80

3.1.4 c = 2.00

- The fixed point x = 0 is repelling. ($|f'(0)| = 2$)
- The fixed point x = 0.5 is repelling ($0.5 < 1$)

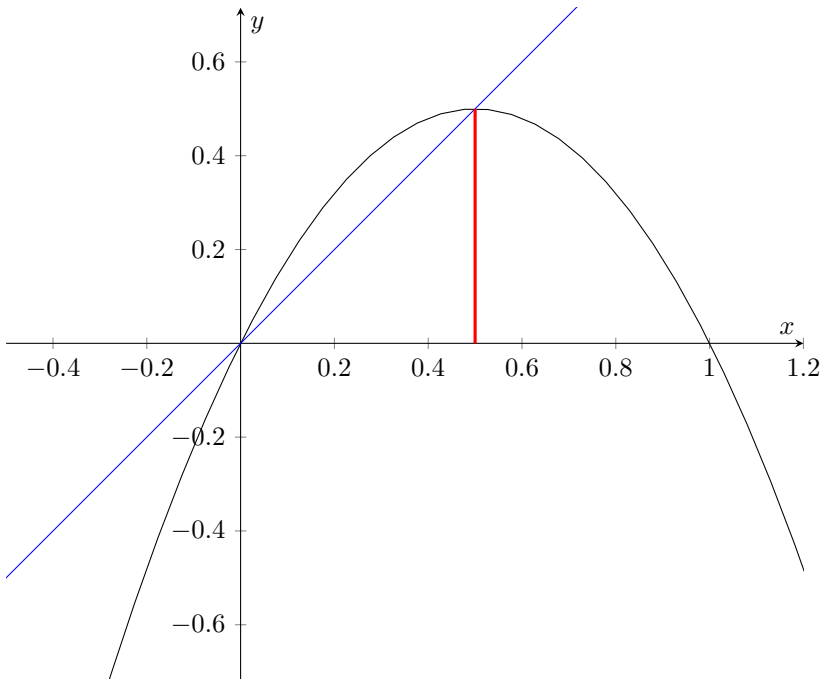

Figure 6: c = 2

3.1.5 c = 3.30

- The fixed point x = 0 is repelling. ($|f'(0)| = 3.3$)
- The fixed point x = 0.69 is repelling ($0.69 < 1$)

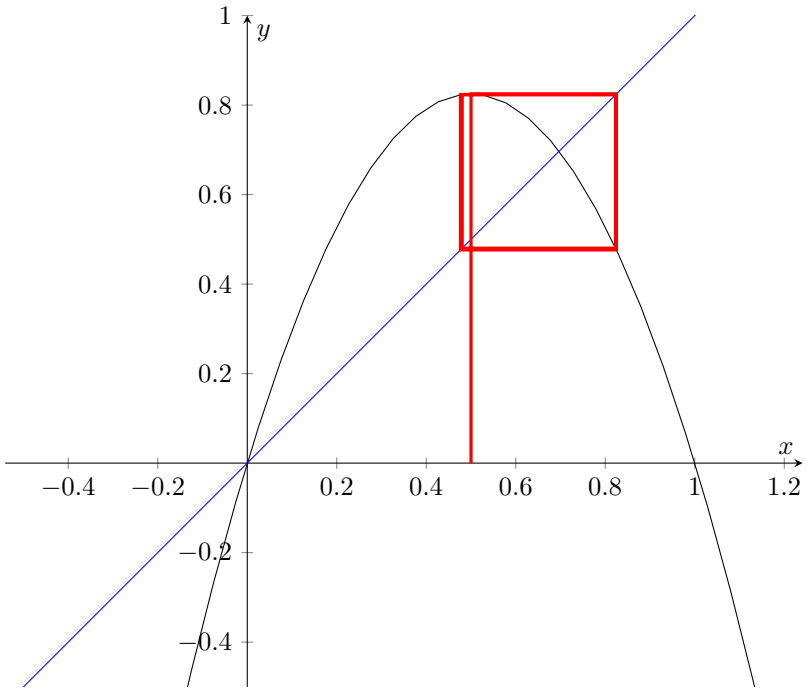

Figure 7: c = 3.3

3.1.6 c = 3.50

- The fixed point x = 0 is repelling. ($|f'(0)| = 3.5$)
- The fixed point x = 0.71 is repelling ($0.71 < 1$)

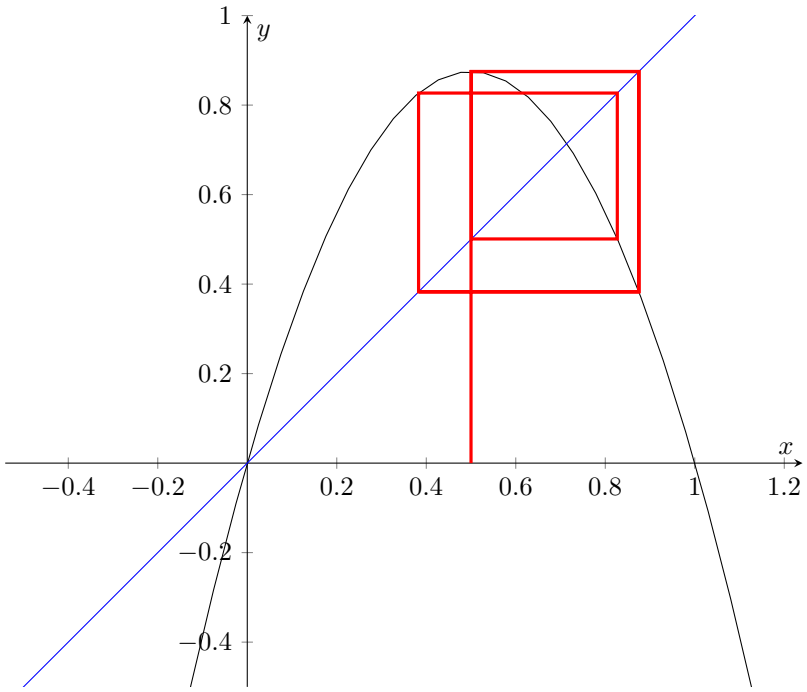

Figure 8: c = 3.5

3.1.7 c = 3.97

- The fixed point x = 0 is repelling. ($|f'(0)| = 3.97$)
- The fixed point x = 0.74 is repelling ($0.74 < 1$)

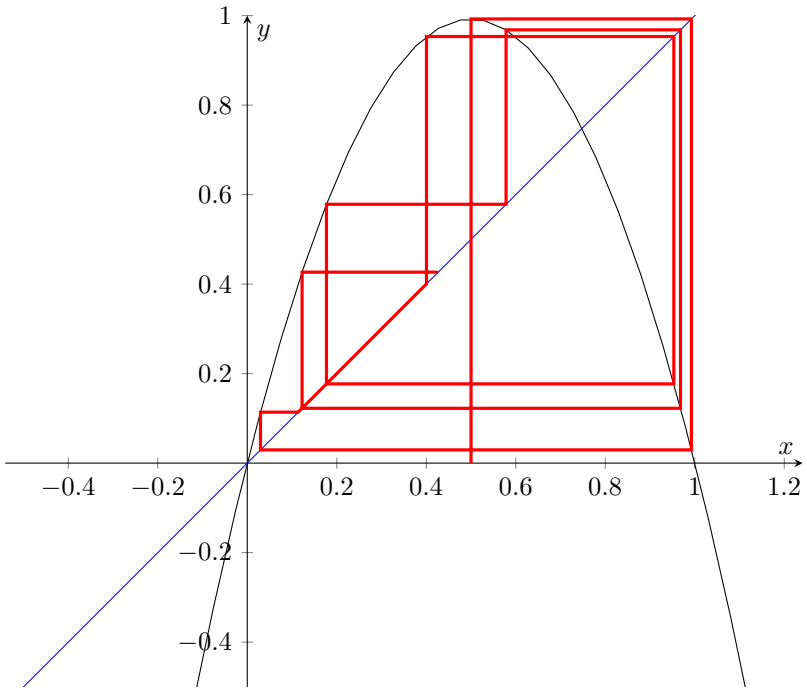

Figure 9: c = 3.97

4 Solving a system of nonlinear equations by Newton's method

Problem
Consider the example of solving a system of nonlinear equations by Newtons method, as given on Pages 95 to 97 of the Lecture notes. Write a program to carry out this iteration, using Gauss elimination to solve the 2 by 2 linear systems that arise. Use each of the following 16 initial data sets for the Newton iteration:

$(x_1^0, x_2^0) = (\text{i, j}),$
$i = 0, 1, 2, 3,$
$j = 0, 1, 2, 3.$

Present and discuss your numerical results in a concise manner.
Solution:

The code and full presentation is available in the directory: *program/Problem4.ipynb*

Below is the supporting functions for the main algorithm:

```
# The original system of linear equation
def G(x):
    g1 = pow(x[0], 2)*x[1] - 1
    g2 = x[1] - pow(x[0], 4)
    return np.array([g1, g2], dtype = np.float)

def convertG(x):
    for number in x:
        if(number != 0):
            number = -1 * number
    return x

# Jacobian matrix
def dG(x):
    J = np.zeros([len(x), len(x)], dtype = np.float)
    eps = 1e-10
    for i in range(len(x)):
        x1 = x.copy()
        x2 = x.copy()

        x1[i] += eps
        x2[i] -= eps

        g1 = G(x1)
        g2 = G(x2)

        J[ : ,i] = (g1 - g2) / (2 * eps)

    return J

def calculateDeltaX(dG, G):
```

```
    return np.linalg.solve(dG, G)

def estimateIteration(guess, deltaX):
    return np.add(guess, deltaX)
```

The main functions are as following:

```
def solveIteration(guessX1, guessX2):
    initGuess = np.array([guessX1, guessX2], dtype = np.float)
    funcG = G(initGuess)
    funcG = convertG(funcG)
    funcDG = dG(initGuess)
    deltaX = calculateDeltaX(funcDG, funcG)
    results = estimateIteration(initGuess, deltaX)
    return results

def SolveDataSet(rangeSet):
    count = 0
    for i in guessSet:
        for j in guessSet:
            count = count + 1
            print("\nIteration: no.", count)
            print("i = ", i, " j = ", j)
            estimation = solveIteration(i, j)
            print("x1 = ", estimation[0], " x2 = ", estimation[1])

guessSet = range(1, 4)
SolveDataSet(guessSet)
```

The output is as following:

Iteration: no. 1 i = 1 j = 1 x1 = 1.0 x2 = 1.0

Iteration: no. 2 i = 1 j = 2 x1 = 1.0 x2 = 2.999999917259636

Iteration: no. 3 i = 1 j = 3 x1 = 1.0 x2 = 4.999999834519272

Iteration: no. 4 i = 1 j = 4 x1 = 1.0 x2 = 6.999999751778907

Iteration: no. 5 i = 2 j = 1 x1 = 2.477272687783008 x2 = 1.2727272501617188

Iteration: no. 6 i = 2 j = 2 x1 = 2.46323525578939 x2 = 2.8235293436255824

Iteration: no. 7 i = 2 j = 3 x1 = 2.449999962766836 x2 = 4.39999988416349

Iteration: no. 8 i = 2 j = 4 x1 = 2.437499963801091 x2 = 5.999999834519272

Iteration: no. 9 i = 3 j = 1 x1 = 3.7443757939848985 x2 = 1.3926382860186328

Iteration: no. 10 i = 3 j = 2 x1 = 3.7398369866787884 x2 = 2.9024376358618897

Iteraton: no. 11 i = 3 j = 3 x1 = 3.7353532030813827 x2 = 4.418179444595575

Iteration: no. 12 i = 3 j = 4 x1 = 3.730923441770851 x2 = 5.939755560512124

Iteration: no. 13 i = 4 j = 1 x1 = 4.997806722363986 x2 = 1.4385965612489156

Iteration: no. 14 i = 4 j = 2 x1 = 4.995865220763078 x2 = 2.9416346189274662

Iteration: no. 15 i = 4 j = 3 x1 = 4.993931258042135 x2 = 4.446602869886977

Iteration: no. 16 i = 4 j = 4 x1 = 4.992004790375868 x2 = 5.95349009345808

5 Newton's method and Bisection method to approximate $\sqrt{(R)}$

Problem: Newton's method to find $\sqrt{(R)}$ is:

$x_{n+1} = \frac{1}{2}(x_n + \frac{R}{x_n})$

- Perform three iterations of scheme (1) for computing $\sqrt{(2)}$, starting with $x_0 = 1$.
- Perform three iterations of the bisection method for computing $\sqrt{(2)}$, starting with interval [1,2].
- Find theoretically the minimum number of iterations in both schemes to achieve 10^{-6} accuracy.
- Find numerically the minimum number of iterations in both schemes to achieve 10^{-6} accuracy and compare your results with the theoretical estimates.

Solution:

5.0.1 Iterations using Newton scheme

The iterations were conducted using Jupyter Notebook with Python. The main algorithm is as followed. The full source code can be found at *program/Problem5.ipynb*

```
def newton_f(x, R):
    return (1/2)*(x + (R/x))

def newton_method(f, x, R, NumOfIteration = 1000):
    init_x = x
    estimates = []
    listX = [x]
    for i in range(numOfIteration):
        x = newton_f(x, R)
        listX.append(x)
        estimates.append(x)
    return listX, estimates
```

The results are as following:

Iteration no. 1 X = 1 Xi = 1.5

Iteration no. 2 X = 1.5 Xi = 1.4166666666666665

Iteration no. 3 X = 1.4166666666666665 Xi = 1.4142156862745097

Final estimation: 1.4142156862745097

5.0.2 Iterations using Bisection method

The iterations were conducted using Jupyter Notebook with Python. The main algorithm is as followed. The full source code can be found at *program/Problem5.ipynb*

```
def f(x):
    return x**2 -2

def bisection_method(f, a, b, NumOfIteration = 1000):
    """
    a, b: the interval
    f: the function to be approximated
    """
    if f(a) * f(b) >= 0:
        print('Bisection method fails')
        return None, None

    a_n = a
    b_n = b
    midpointRecord = []
    for n in range(0, NumOfIteration):
        midpoint = (a_n + b_n)/2
        midpointRecord.append(midpoint)
        fMidpoint = f(midpoint)
        if f(a_n) * fMidpoint < 0:
            b_n = midpoint
        elif f(b_n) * fMidpoint <0:
            a_n = midpoint
        elif fMidpoint == 0:
            print('Found exact solution')
            return midpoint
    return (a_n + b_n)/2, midpointRecord
```

The results are as following:

Iteration no. 1 Midpoint: 1.5

Iteration no. 2 Midpoint: 1.25

Iteration no. 3 Midpoint: 1.375

Final estimation: 1.4375

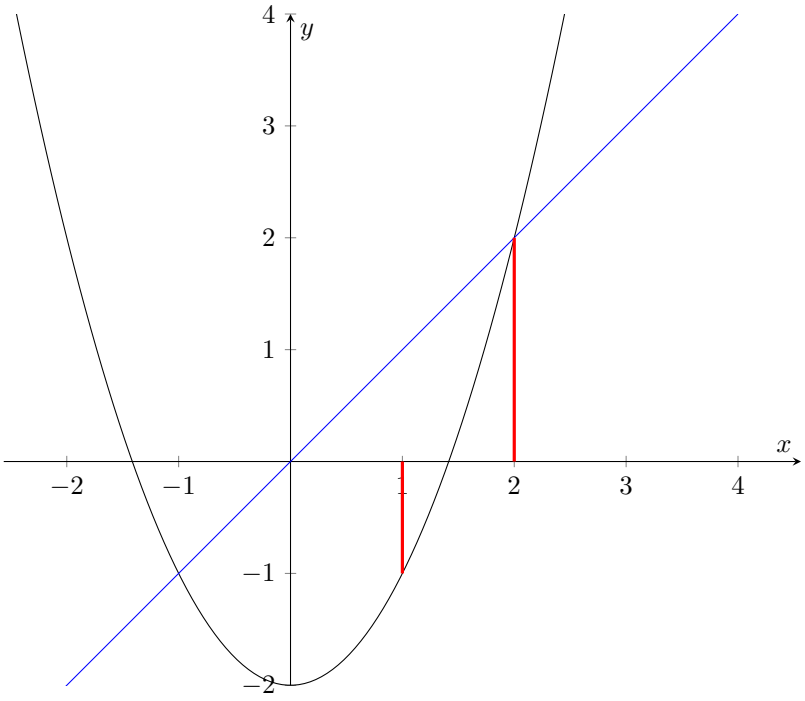

Figure 10: Bisection method demonstration diagram

5.0.3 Minimum number of iterations for 10^{-6} accuracy

Scheme (1)
Bisection method
Error of the approximation:

$$e_n = |r - c_n| \leq \frac{b_0 - a_0}{2} \frac{1}{2^n}.$$

With

- n: the n^{th} iteration
- e_n Denotes the error
- r: the real value
- c_n: The estimation at iteration n
- a_0, b_0: The 2 starting points of the starting interval

$$\implies e_n \leq \frac{2-1}{2} \frac{1}{2^n} \leq 10^{-6}$$
$$\implies 2^{n+1} \geq 10^6 \implies n+1 \geq 20 \implies n \geq 19$$

Conclusion: With the minimum of number of iteration is 19, the function will have 10^{-6} accuracy.

5.0.4 Numerically find the minimum number of iterations

In order to find the minimum number of iterations, Jupyter Notebook with Python was utilized. The full source code is presented with example in the directory: *program/Problem5.ipynb*
Newton's Scheme
The main algorithm is as following:

```
# Define global criteria
tol = pow(10, -6)
r = math.sqrt(2) # real value
x = 1
R = 2

def newton_MinOfIteration(f, x, R, tol, r, numOfIteration = 1000):
    init_x = x
    n = 0
    for i in range(numOfIteration):
        n += 1
        x = newton_f(x, R)
        if(abs(r - x)) <= tol:
            return n
    print('The min of iteration satisfies tolerance is beyond the
        allowed number of iteration')
    return None
```

```
n = newton_MinOfIteration(newton_f, x, R, tol, r)
print('\nTo achive the accuracy of ', tol, ' with Newton\'s method, the
    minimum number of iteration is ', n)
```

The result:

> To achive the accuracy of 1e-06 with Newton's method, the minimum number of iteration is 4

textbfBisection method

Similarly, the main algorithm to find the minimum number of iteration to get 10^{-6} accuracy is as following:

```
# Define global criteria
tol = pow(10, -6)
r = math.sqrt(2) # real value
x = 1
R = 2

def bisection_MinOfIteration(f, a, b, r, tol, NumOfIteration = 1000):
    if f(a) * f(b) >= 0:
        print('Bisection method fails')
        return None, None
    a_n = a
    b_n = b
    n = 0
    for n in range(0, NumOfIteration):
        n += 1
        midpoint = (a_n + b_n)/2
        if abs(midpoint - r) <= tol:
            return n

        fMidpoint = f(midpoint)
        if f(a_n) * fMidpoint < 0:
            b_n = midpoint
        elif f(b_n) * fMidpoint <0:
            a_n = midpoint
        elif fMidpoint == 0:
            print('Found exact solution')
            return n
    print('The min of iteration satisfies tolerance is beyond the
        allowed number of iteration')
    return None

# Numerically calculate the minimum number of n for accuracy of 10^(-6)
n = bisection_MinOfIteration(f, a, b, r, tol)
print('\nTo achive the accuracy of ', tol, ' with Bisection method, the
    minimum number of iteration is ', n)
```

The result:

> To achive the accuracy of 1e-06 with Bisection method, the minimum number of iteration is 19

Conclusion: The result is as calculated in the previous question, stating the correctness of theoretically calculating.

6 Bonus: A modified Newton's method:

Problem:
The **multiplicity** of the zero x^* is the least integer m such that $f^(k) * (x^*) = 0$ for $0 \leq k < m$, but $f^(m) * (x^*) \neq 0$. Show analytically that in the case of a zero of multiplicity m, the **modified Newton's method:**
$x_{n+1} = x_n - m * (\frac{f(x_n)}{f'(x_n)})$
is quadratically convergent.

Answer: Let $g(x) = (x - m)^k h(x)$, with $h(m) \neq 0$. Substitue g'(x) and g"(x) into the Newton's method |f'(x)|.
Substitue m into the function to get |f'(m)| = 0 $\implies$ The function converges quadratically