Assignment 1

4 March 2020

MAM2046W 2NA

KDSMIL001

1. Newton-Raphson root-finding

   (a) We are to find $f(x)$. In the Newton-Raphson Method, the formula is

   $$x_{n+1} = x_n - \frac{f(x)}{f'(x)}$$

   So, we can deduce from the equation given to us

   $$x_{n+1} = x_n - (\cos x_n)(\sin x_n) + R\cos^2 x_n$$

   that

   $$\begin{aligned} \frac{f(x)}{f'(x)} &= \cos x_n \sin x_n + R\cos x_n \\ \implies \frac{df}{f} &= \frac{1}{\cos x_n \sin x_n + R\cos x_n} dx \end{aligned}$$

   Now, we perform a substitution in order to simplify some things. Let $x_n = \arctan u$. Then

   $$\begin{aligned} dx &= \frac{1}{1+u^2} du \\ \implies \int \frac{df}{f} &= \int \frac{1}{\cos x_n \sin x_n + R\cos x_n} dx \\ \implies \ln|f| &= \int \frac{1}{\frac{u}{\sqrt{1+u^2}}\frac{1}{\sqrt{1+u^2}} + \frac{R}{1+u^2}} \frac{1}{1+u^2} du \\ \implies \ln|f| &= \int \frac{1}{\frac{u}{1+u^2} + \frac{R}{1+u^2}} \frac{1}{1+u^2} du \\ \implies \ln|f| &= \int \frac{1}{u+R} du \\ \implies \ln|f| &= \ln(u+R) + C, \quad C \in \mathrm{Re} \\ \implies f &= (u+R)e^C \end{aligned}$$

   Let $e^C = A$, then

   $$f = A(\tan x_n + R)$$

   (b) This formula can be used to find the roots of some function f.

2. Fixed Point Iteration

   (a) Given the equation of the form $f(x) = x^2 - x - 2 = 0$, it doesn't take much manipulation to reduce it to the first two solutions:

   $$g_1(x) = x^2 - 2$$

$$g_2(x) = \sqrt{x+2}$$

For the next two solutions, a little bit more manipulation is required:

$$\begin{aligned} & x^2 - x - 2 = 0 \\ \implies & x^2 - x = 2 \\ \implies & x(x-1) = 2 \\ \implies & x = \frac{2}{x-1} \\ \implies & g_3(x) = \frac{2}{x-1} \end{aligned}$$

and

$$\begin{aligned} & x^2 - x - 2 = 0 \\ \implies & x^2 - x = 2 \\ \implies & x(x-1) = 2 \\ \implies & x - 1 = \frac{2}{x} \\ \implies & x = \frac{2}{x} + 1 \\ \implies & g_4(x) = \frac{2}{x} + 1 \end{aligned}$$

(b) To plot these functions on the same axes, I used Python with the `matplotlib` module, below is the result [Figure 1]. It's clear to see that all of the graphs intersect the $y = x$ line at two points with the exception of $g_2(x)$, which has no values < 0 as a function. The code for this can be found in [Appendix 1]

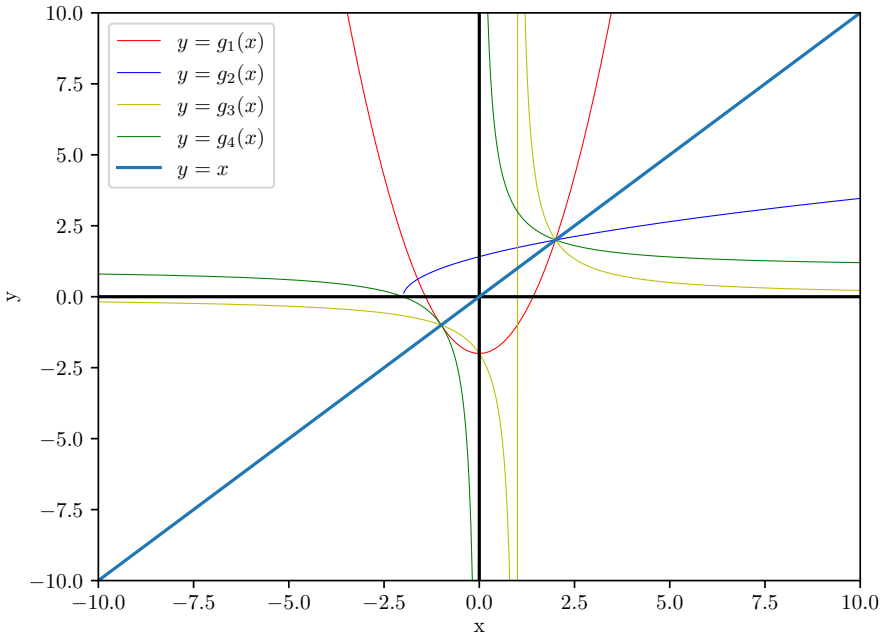

Figure 1: Plot of each $g_i(x)$

(c) As we can see from the plot, both $g_2'(x)$ and $g_4'(x)$ are less than 1, which means they will converge on the fixed point. The other two, however, will not converge on the positive root. We can't say whether they will diverge or whether they will converge on the negative root, but they will not converge on the positive root, which is what we are looking for.

3. Halley's Method

(a) The program below, [Q3a.py], results in a value of $x_n = 2.15443$ after 3 iterations, excluding the initial guess. As you can see, the stop condition for this method was that the difference between x_n and x_{n+1} has an absolute value less than 0.00001. This method is not guaranteed to work for every function, but in this case it was sufficient.

```
initialGuess = 2
def f(xn, xn2, count):
    count += 1
    xn2 = xn-(((xn**4)-(10*xn))/((2*xn**3)+(10)))
    if (abs(xn - xn2)) < 0.00001:
        return xn2, count
    else:
        return f(xn2, xn, count)

print(f(initialGuess, initialGuess, 0))
```

Q3a.py

(b) Comparing the number of iterations for Halley's Method to the number of iterations needed to reach a root when using Newton's Method, which in this case would be 4 iterations excluding the initial guess, we can see that while Halley's Method is better, it's not by much. The code performing this calculation is below [Q3b.py]. It also returns $x_n = 2.15443$.

```
initialGuess = 2
def f(xn, xn2, count):
    count += 1
    xn2 = xn-((xn**3-10)/(3*xn**2))
    if (abs(xn - xn2)) < 0.00001:
        return xn2, count
    else:
        return f(xn2, xn, count)

print(f(initialGuess, initialGuess, 0))
```

Q3b.py

(c) Firstly, to check that there is a root in the interval we can use the Intermediate Value Theorem along with the fact that $f(2) = -2$ and $f(4) = 6$. Next we can write a program, [Q3c.py] below, which does the iterations for us. We know that $f(x_1)$ is the negative value, so we don't need to code that in. Comparing the result in Problem 3a to the result obtained when using the Bisection method, calculated below in [Q3c.py], we can see that Bisection converges in 18 iterations. Compared to the 3 iterations that Halley's Method took, this is terrible.

```python
def f(x):
    return x**3-10
x1 = 2
x2 = 4
x4 = abs(x1-x2)
count = 0
found = False
while x4 > 0.00001:
    count += 1
    x3 = (x2+x1)/2
    if f(x3) < 0:
        x1 = x3
    elif f(x3) > 0:
        x2 = x3
    elif f(x3) == 0:
        print(x3, count)
        found = True
        break
    x4 = abs(x1-x2)
if found == False:
    print(x3, count)
```

Q3c.py

4. Firstly, the functions we are looking at are

$$g_1(x) = x^2 - 2$$
$$g_2(x) = \sqrt{x+2}$$
$$g_3(x) = \frac{2}{x-1}$$
$$g_4(x) = \frac{2}{x} + 1$$

Using these functions, the code below uses Fixed Point Iteration to find one of the roots of the original function $f(x) = x^2 - x - 2$. We hoped to find the positive root that was visible in Figure 1, but after running the program we found these results:

Function	Fixed Point	# of Iterations
$g_1(x)$	Could not converge	100
$g_2(x)$	2	7
$g_3(x)$	-1	21
$g_4(x)$	2	16

These values are all rounded as the program ran to an accuracy of 0.00001, which returned values such as 1.9999998198672908, which are unnecessarily verbose.

On the topic of convergence to a root, all but one of the functions converged to a root, but $g_3(x)$ seemed to converge on the negative root, namely -1. $g_1(x)$ did not converge, which was expected as $|g_1'(x)| > 1$ at the root. The same was true for $g_3(x)$, but its derivative was negative at the root, which led to it converging on the negative root. This is consistent with our prediction in Question2a but it also shows us which function converges on the negative root and which diverges.

```
from math import sqrt
def g1(x):
    return x**2-2
def g2(x):
    return sqrt(x+2)
def g3(x):
    return 2/(x-1)
def g4(x):
    return (2/x)+1
def FPI(g, startX):
    diff = abs(g(startX)-startX)
    xNew = g(startX)
    count = 0
    while diff > 0.00001 and count < 100:
        xNew = g(xNew)
        diff = abs(g(xNew)-xNew)
        count += 1
    if count == 100:
        return "Could not converge"
    else:
        return xNew, count
print(FPI(g1, 1.5))
print(FPI(g2, 1.5))
print(FPI(g3, 1.5))
print(FPI(g4, 1.5))
```

Q4.py

Appendix

Appendix 1: Q2.py

```python
from matplotlib import pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
import matplotlib
matplotlib.use('pgf')
matplotlib.rcParams.update({
    'pgf.texsystem': 'pdflatex',
    'font.family': 'serif',
    'text.usetex': True,
    'pgf.rcfonts': False,
})

x = np.linspace(-10, 10, 500)
y = np.linspace(-10, 10, 500)

def y1(x):
    return (x**2)-2
def y2(x):
    return np.sqrt(x+2)
def y3(x):
    return 2/(x-1)
def y4(x):
    return (2/x)+1

y32 = y3(x)[~np.isnan(y3(x))]

plt.plot(x, y1(x), color='r', lw=0.5, label='$y = g_1(x)')
plt.plot(x, y2(x), color='b', lw=0.5, label='$y = g_2(x)$')
plt.plot(x, y32, color='y', lw=0.5, label='$y = g_3(x)$')
plt.plot(x, y4(x), color='g', lw=0.5, label='$y = g_4(x)$')
plt.plot(x, np.zeros(500), color='black')
plt.plot(np.zeros(500), y, color='black')
plt.plot(x, y, label='$y = x$')

plt.ylim(-10, 10)
plt.xlim(-10, 10)
plt.xlabel('x')
plt.ylabel('y', rotation=0)
plt.legend()
plt.savefig('2NA Assignments\Assignment 1\Q2 Plot.pgf')
# plt.show()
```