HOMEWORK 4

LIAM DUNGAN

Problem 1.

(a)

We can show that, if this method converges, it will converge to a solution of $f(x) = 0$ by showing $x = g(x)$ is mathematically equivalent to $f(x) = 0$.

$f(x) = 0 \Leftrightarrow -\frac{f(x)}{f'(x)} = 0 \Leftrightarrow x - \frac{f(x)}{f'(x)} = x \Leftrightarrow x + x - \frac{f(x)}{f'(x)} = x + x \Leftrightarrow x + x - \frac{f(x)}{f'(x)} = 2x \Leftrightarrow \frac{x+x-\frac{f(x)}{f'(x)}}{2} = x \Leftrightarrow g(x) = x \quad where \quad g(x) \equiv \frac{x+(x-\frac{f(x)}{f'(x)})}{2} = \frac{x+(\hat{x})}{2}$

(b) This method only converges under the condition g(x) is a contraction, just like newtons method. In example, for the interval [a, b], ther must exist L, such that $0 \leq L < 1$ and $|g(x) - g(y)| \leq L\,|x - y|$

(c) This method converges linearly, the order of converegence is 1.

Problem 2.

(a)

$|g_1'(x)| = \left|\frac{2x}{3}\right| \Longrightarrow \left|\frac{2(2)}{3}\right| = 1.33$

>1, so does not converge

$|g_2'(x)| = \left|\frac{3}{2\sqrt{3x-2}}\right| \Longrightarrow \left|\frac{3}{2\sqrt{3(2)-2}}\right| = 0.75$

<1, so converges at a rate of 0.75

$|g_3'(x)| = \left|\frac{2}{x^2}\right| \Longrightarrow \left|\frac{2}{(2)^2}\right| = 0.5$

<1, so converges at a rate of 0.5

$|g_4'(x)| = \left|\frac{-2(x^2-2)}{(2x-3)^2} + \frac{2x}{2x-3}\right| \Longrightarrow \left|\frac{-2((2)^2-2)}{(2(2)-3)^2} + \frac{2(2)}{2(2)-3}\right| = 0$

=0, so converges quadratically (at least)

(b)

For the following cases, we use an intial guess of 5 and $\epsilon = 10^{-5}$.

As you can see from the output below, g1(x) diverges, resulting in an overflow error.

Date: Nov. 14, 2017.

```
Initial guess:  5
1       x:  9.0          e:  4.0
2       x:  27.6666666667       e:  18.6666666667
3       x:  255.814814815       e:  228.148148148
4       x:  21814.4064929       e:  21558.5916781
5       x:  158622777.546       e:  158600963.14
6       x:  8.38706185214e+15   e:  8.38706169351e+15
7       x:  2.34476021705e+31   e:  2.34476021705e+31
8       x:  1.83263349182e+62   e:  1.83263349182e+62
9       x:  1.11951517178e+124  e:  1.11951517178e+124
10      x:  4.17771406619e+247  e:  4.17771406619e+247
Traceback (most recent call last):
  File "2.py", line 47, in <module>
    main()
  File "2.py", line 42, in main
    root = fpi(g1, 5, 10e-5, 50)
  File "2.py", line 35, in fpi
    e = abs(g(x) - x)
  File "2.py", line 9, in g1
    return ((math.pow(x, 2) + 2) / 3)
OverflowError: math range error
nbp-254-201:desktop liamdungan$
```

For $g_2(x), g_3(x), and g_4(x)$ the error converges as we predictied in our intial calculations. The output for all three can be seen below.

$g_2(x)$:

```
Initial guess:  5
1       x:  3.60555127546       e:  1.39444872454
2       x:  2.96928506991       e:  0.636266205554
3       x:  2.62827989562       e:  0.341005174288
4       x:  2.42586885195       e:  0.202411043671
5       x:  2.29730419315       e:  0.128564658804
6       x:  2.21176684563       e:  0.0855373475142
7       x:  2.15297481102       e:  0.0587920346102
8       x:  2.11161654499       e:  0.0413582660346
9       x:  2.08203017148       e:  0.0295863735055
10      x:  2.06060440513       e:  0.0214257663511
11      x:  2.04494821827       e:  0.0156561868643
12      x:  2.03343174334       e:  0.0115164749325
13      x:  2.02491857367       e:  0.00851316966224
14      x:  2.01860241777       e:  0.00631615590491
15      x:  2.01390348659       e:  0.00469893117683
16      x:  2.01040057197       e:  0.00350291462209
17      x:  2.00778527634       e:  0.00261529562419
18      x:  2.0058304587        e:  0.00195481764835
19      x:  2.004368074         e:  0.00146238469174
20      x:  2.00327337675       e:  0.00109469725002
21      x:  2.00245352762       e:  0.000819849138114
22      x:  2.00183929996       e:  0.000614227660226
23      x:  2.00137899956       e:  0.000460300399032
24      x:  2.00103398239       e:  0.000345017169224
25      x:  2.0007753365        e:  0.000258645883697
26      x:  2.00058141787       e:  0.000193918637783
27      x:  2.00043601587       e:  0.000145401994113
28      x:  2.00032698517       e:  0.000109030697951
29      x:  2.00024522385       e:  8.1761327321e-05
The root is:  2.00024522385
nbp-254-201:desktop liamdungan$
```

$g_3(x)$:

```
Initial guess:  5
1       x:  2.6          e:  2.4
2       x:  2.23076923077        e:  0.369230769231
3       x:  2.10344827586        e:  0.127320954907
4       x:  2.04918032787        e:  0.0542679479932
5       x:  2.024        e:  0.0251803278689
6       x:  2.01185770751        e:  0.0121422924901
7       x:  2.00589390963        e:  0.00596379788316
8       x:  2.00293829579        e:  0.00295561383828
9       x:  2.00146699267        e:  0.00147130312341
10      x:  2.00073295871        e:  0.000734033955044
11      x:  2.0003663451         e:  0.000366613612911
12      x:  2.000183139          e:  0.000183206094579
13      x:  2.00009156112        e:  9.15778854575e-05
The root is:  2.00009156112
nbp-254-201:desktop liamdungan$
```

$g_4(x)$:

```
Initial guess:  5
1       x:  3.28571428571        e:  1.71428571429
2       x:  2.46285714286        e:  0.822857142857
3       x:  2.11125052989        e:  0.351606612972
4       x:  2.01012406517        e:  0.101126464715
5       x:  2.00010046252        e:  0.0100236026528
6       x:  2.00000001009        e:  0.000100452426733
7       x:  2.0          e:  1.00906905054e-08
The root is:  2.0
nbp-254-201:desktop liamdungan$
```

$g_4(x)$ converges the fasted, then $g_3(x)$, then $g_2(x)$.

Code:

```
# Liam Dungan - CS323 Numerical Analysis - Nov. 14, 2017
import math

# For f(x) = x^2 - 3x + 2 = 0
# Each of the following represents an equivalent fixed point problem


# diverges - cannot use
def g1(x):
    return ((math.pow(x, 2.) + 2.) / 3.)


# converges
def g2(x):
    return math.sqrt((3. * x) - 2.)


# converges
def g3(x):
    return (3. - (2. / x))


# converges
def g4(x):
    return ((math.pow(x, 2.) - 2.) / ((2. * x) - 3.))
```

```python
# fpi() impliments fixed point iteration to return the root of a given problem
# @param g : the g(x) to be used in the iteration (choose one defined above)
# @param x : int, the intial guess
# @param maxep : float, the max epsilon allowed before iteration stops
# @param maxit : int, the max iterations allowed before iteration stops
def fpi(g, x, maxep, maxit):
    print "\nInitial guess: ", x
    i = 0
    e = abs(g(x) - x)
    while(e > maxep and i < maxit):
        i += 1
        e = abs(g(x) - x)
        x = g(x)
        print i, "\tx: ", x, "\te: ", e
    print "The root is: ", x
    return x


def main():
    # fpi(g1, 5, 10e-5, 50)
    fpi(g2, 5, 10e-5, 50)
    # fpi(g3, 5, 10e-5, 50)
    # fpi(g4, 5, 10e-5, 50)


if __name__ == "__main__":
    main()
```

Problem 3.

The termination cirteria used is $f(x_k) < \delta$ and $|x_k - x_{k-1}| < \varepsilon$, in this case I used $\varepsilon = 10^{-5}$.

The roots obtained for each method can be seen in the output below.

```
Newton's Method:
2.0945514817
0.56714329038
1.11415714087
1.00013365854

Bisection Method:
2.09453582764
0.56713104248
1.11415863037
1.00002288818

Secant Method:
2.09455147943
0.567143284589
1.11415713957
0.966948951573
nbp-254-201:desktop liamdungan$
```

Code:

```python
# Liam Dungan - CS323 Numerical Analysis - Nov. 14, 2017
# HW4 problem 3
import math


# Example equations
def a(x):
    return (x**3) - (2 * x) - 5


def b(x):
    return (math.e**(-x)) - x


def c(x):
    return x * math.sin(x) - 1


def d(x):
    return (x**3) - (3 * (x**2)) + (3 * x) - 1


# Derivatives of each Equation
def da(x):
    return (3 * (x**2)) - 2


def db(x):
    return -(math.e**(-x)) - 1


def dc(x):
    return x * math.cos(x) + math.sin(x)
```

```python
def dd(x):
    return (3 * (x**2)) - (6 * x) + 3


# An implimentation of the Newton's method for finding roots of a function
# param f : the function
# param df : the derivative of the fucntion
# param x : int, the intial guess
# param maxep : float, the max epsilon allowed before iteration stops i.e 10e-5
def newtons(f, df, x, maxep):
    temp = x
    x = x - (f(x) / df(x))
    e = abs(x - temp)
    while e > maxep:
        temp = x
        x = x - (f(x) / df(x))
        e = abs(x - temp)
    print x
    return x


# An implimentation of the bisection method for finding roots of a function
# param f : the function
# param a : int, start of the interval containing root
# param b : int, end of the interval containing root
# param maxep : float, the max epsilon allowed before iteration stops i.e 10e-5
def bisection(f, a, b, maxep):
    mid = (a + b) / 2.
    while (b - a) > maxep:
        if f(mid) == 0:
            return mid
        elif f(a) * f(mid) < 0:
            b = mid
        else:
            a = mid
```

```
        mid = (a + b) / 2.
    print mid
    return mid


# An implimentation of the Secant method for finding roots of a function
# param f : the function
# param x : int, the first initial guess
# param x2 : int, the second intial guess
# param maxep : float, the max epsilon allowed before iteration stops i.e 10e-5
def secant(f, x, x2, maxep):
    while abs(f(x2) - f(x)) > maxep:
        temp = x2 - (f(x2) * (x2 - x) * 1.0) / (f(x2) - f(x))
        x = x2
        x2 = temp
    print x2
    return x2


def main():
    print "\nNewton's Method: "
    newtons(a, da, 2., 10e-5)
    newtons(b, db, 2., 10e-5)
    newtons(c, dc, 1., 10e-5)
    newtons(d, dd, 2., 10e-5)
    print "\nBisection Method: "
    bisection(a, 0, 5, 10e-5)
    bisection(b, 0, 5, 10e-5)
    bisection(c, 0, 5, 10e-5)
    bisection(d, 0, 5, 10e-5)
    print "\nSecant Method: "
    secant(a, 0, 3, 10e-5)
    secant(b, 0, 3, 10e-5)
    secant(c, 0, 2, 10e-5)
    secant(d, 0, 3, 10e-5)
```

Problem 4.