University of Colorado
Boulder

HW3

Jesse Hettleman
APPM 4600

College of Engineering and Applied Science
Department of Applied Math

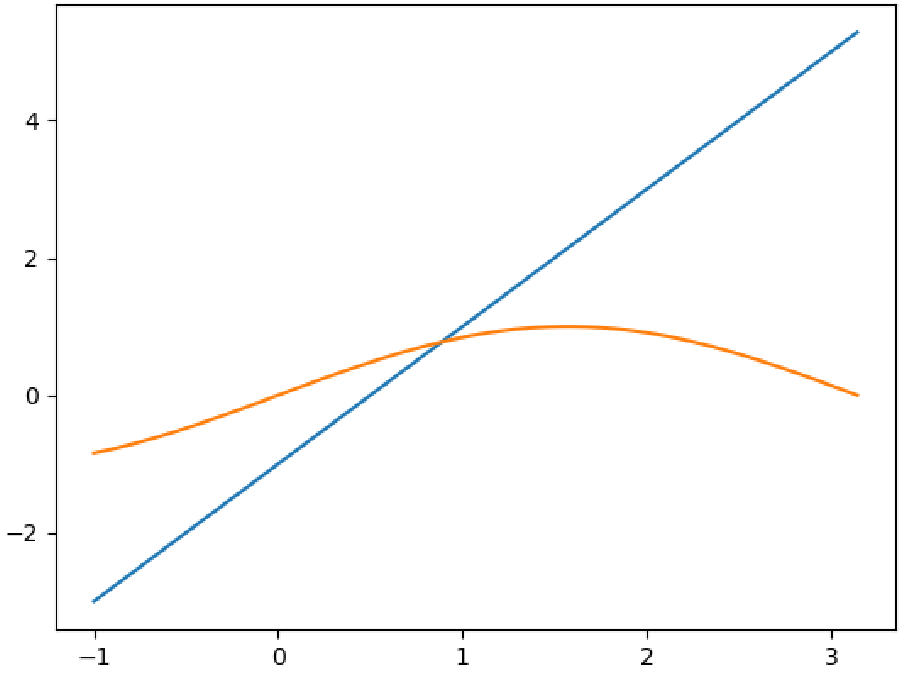
September 20, 2024

1 Problem 1

1.a

The closed interval $[0, \frac{\pi}{2}]$ contains a root r to the equation $f(x) = 2x - 1 - sin(x) = 0$. I determined this interval by plotting $y_1 = 2x - 1$ and $y_2 = sin(x)$ and visualizing where they intersect. This is displayed in the following chart:

Figure 1: Intersection of $2x - 1 = sin(x)$



If we let $f(x) = 2x - 1 - sin(x)$, then $f(0) = -1$ and $f(\frac{\pi}{2}) = \pi - 2 > 0$.

By the Intermediate Value Theorem, there exists some r within $[0, \frac{\pi}{2}]$ such that $f(r) = 0$ because $f(0) < 0$ and $f(\frac{\pi}{2}) > 0$.

1.b

We know the first derivative $f'(x) = 2 - cos(x) > 0$, which means the slope of f is always positive. This means f is monotonic and increasing, so once it intersects the x-axis at $x = r$ it will continue to increase and never intersect the x-axis again. Therefore, r somewhere within $[0, \frac{\pi}{2}]$ is the only root of the equation.

1.c

Using the bisection method with an error tolerance of $1 * 10^{-8}$, we approximate $r = 0.88786221$. This required 27 iterations to achieve the desired level of accuracy. This number makes sense because we expect 3 or 4 iterations for each decimal digit of accuracy. See 6 for code resulting in these calculations. This code was modified from the class example on canvas.

2 Problem 2

2.a

Using the bisection method with $f(x) = (x-5)^9$, $a = 4.82$, $b = 5.2$, and an error tolerance of $tol = 1*10^{-4}$ yields a root approximation of $r = 5.000073242187501$. It took 11 iterations to achieve this estimation, which makes sense because there are four digits of precision.

2.b

Using the bisection method with the expanded version of $f(x)$ and all the same inputs yields a root approximation of $r = 4.82$. The bisection method encounters an error in this case and returns the input value for a as an estimation of r.

2.c

The reason this is occurring is because the expanded version of the polynomial calculates $f(a) < 0$ and $f(b) < 0$, so when it tests to see if the bisection method will work it gets an error because $f(a)*f(b) > 0$. The actual value for $f(b)$ is greater than 0, and the non-expanded version of the function correctly calculates this, so it is able to proceed with the root finding method. The expanded polynomial is most likely getting this incorrect due to loss of precision during repeated subtraction operations of close numbers. I determined this was the source of error by printing $f(a)$ and $f(b)$ for each version. See 6 for code.

3 Problem 3

3.a

From class we know an upper bound on the number of iterations, n, is given by:

$$n > \lceil log_2(\frac{b-a}{tol}) - 1 \rceil$$

Calculating the upper bound for this formula using the inputs $a = 1$, $b = 4$, and $tol = 1*10^{-3}$ for the function $f(x) = x^3 + x - 4 = 0$ yields $n = 11$.

3.b

Using the bisection code from class, the upper bound $n = 11$ appears to be correct. The bisection method estimated $r = 1.378662109375$ within 11 iterations. See 6 for code.

4 Problem 4

4.a

Given the sequence $x_{n+1} = -16 + 6x_n + \frac{12}{x_n}$ and letting $\lim_{n\to\infty} x_n = x$, we know:

$$\lim_{n\to\infty} x_{n+1} = \lim_{n\to\infty} -16+6x_n+\frac{12}{x_n} \tag{1}$$
$$x = -16+6x+\frac{12}{x} \tag{2}$$
$$x = 2 \tag{3}$$

Thus, $\lim_{n\to\infty} x_n = 2$. Further, if we let $g(x) = -16+6x+\frac{12}{x}$, then $g'(x) = 6-\frac{12}{x^2}$. In this case, $g'(x_*) = g'(2) = 6-3 = 3 > 1$, so by Theorem 2 the iteration will not converge because $|g'(x_*)| \not< 1$.

4.b

Given the sequence $x_{n+1} = \frac{2}{3}x_n + \frac{1}{x_n^2}$ and letting $\lim_{n\to\infty} x_n = x$, we know:

$$\lim_{n\to\infty} x_{n+1} = \lim_{n\to\infty} \frac{2}{3}x_n+\frac{1}{x_n^2} \tag{4}$$
$$x = \frac{2}{3}x+\frac{1}{x^2} \tag{5}$$
$$x = 3^{\frac{1}{3}} \tag{6}$$

Thus, $\lim_{n\to\infty} x_n = 3^{\frac{1}{3}}$. Further, if we let $g(x) = \frac{2}{3}x+\frac{1}{x^2}$, then $g'(x) = \frac{2}{3}-\frac{2}{x^3}$. In this case, $g'(x_*) = g'(3^{\frac{1}{3}}) = \frac{2}{3}-\frac{2}{3} = 0 < 1$, so by Theorem 2 the iteration converges because $|g'(x_*)| < 1$.

To test for convergence, we will manipulate Taylor's theorem. There exists c in the interval between c and x such that:

$$g(x) = g(x_*)+g'(c)(x-x_*)+\frac{g''(c)(x-x_*)^2}{2!} \tag{7}$$
$$= x_*+\frac{g''(c)(x-x_*)^2}{2!} \tag{8}$$
$$x_{n+1} = x_*+\frac{g''(c)(x-x_*)^2}{2!} \tag{9}$$
$$x_{n+1}-x_* = \frac{g''(c)(x-x_*)^2}{2!} \tag{10}$$
$$\frac{x_{n+1}-x_*}{(x-x_*)^2} = \frac{g''(c)}{2!} \tag{11}$$
$$= const \tag{12}$$

To determine the order of convergence, we can use the given definition:

$$\lim_{n\to\infty} \frac{|p_{n+1}-p|}{|p_n-p|^{\alpha}} = \lambda$$

with $\alpha = 2$ to test for suspected quadratic convergence.

$$\lim_{n\to\infty} \frac{|p_{n+1}-p|}{|p_n-p|^2} = \lim_{n\to\infty} \frac{|x_{n+1}-x_*|}{|x_n-x_*|^2} \tag{13}$$
$$= \lim_{n\to\infty} \frac{g''(c)}{2!} \tag{14}$$
$$= const \tag{15}$$

Therefore, this sequence is quadratically convergent.

4.c

Given the sequence $x_{n+1} = \frac{12}{1+x_n}$ and letting $\lim_{n\to\infty} x_n = x$, we know:

$$\lim_{n\to\infty} x_{n+1} = \lim_{n\to\infty} \frac{12}{1+x_n} \tag{16}$$
$$x = \frac{12}{1+x} \tag{17}$$
$$x = 3 \tag{18}$$

Thus, $\lim_{n\to\infty} x_n = 3$. Further, if we let $g(x) = \frac{12}{1+x_n}$, then $g'(x) = \frac{-12}{(1+x_n)^2}$. In this case, $g'(x_*) = g'(2) = \frac{12}{16} < 1$, so by Theorem 2 the iteration converges because $|g'(x_*)| < 1$.

To test for convergence, we will again manipulate Taylor's theorem. There exists c in the interval between c and x such that:

$$g(x) = g(x_*) + g'(c)(x-x_*) \tag{19}$$
$$x_{n+1} = x_* + g'(c)(x-x_*) \tag{20}$$
$$x_{n+1} - x_* = g'(c)(x-x_*) \tag{21}$$
$$\frac{x_{n+1}-x_*}{x-x_*} = g'(c) \tag{22}$$
$$= const \tag{23}$$

To determine convergence, we can test for non-quadratic, linear convergence:

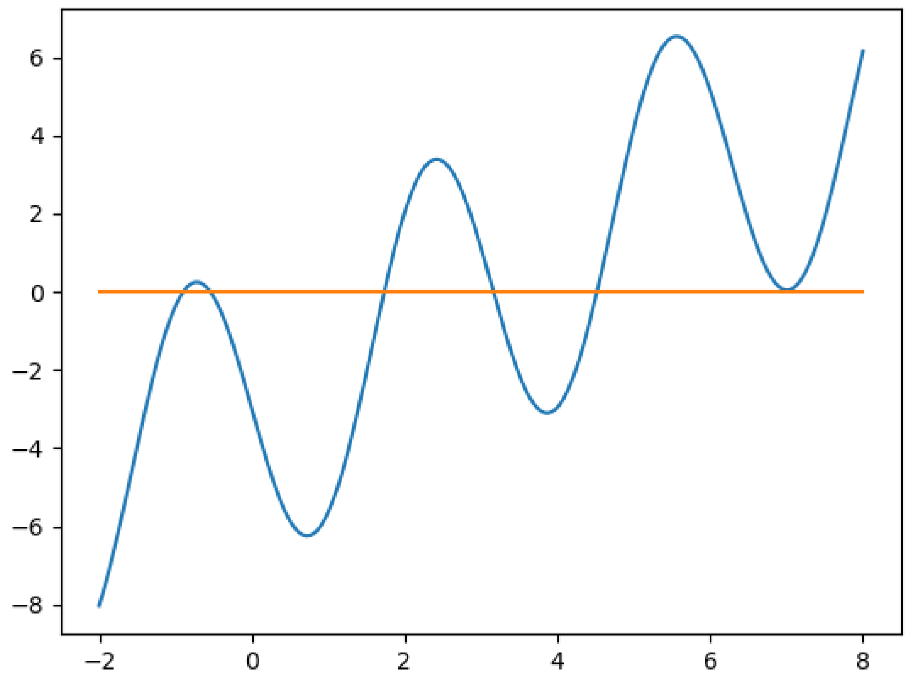
$$\lim_{n\to\infty} \frac{|p_{n+1}-p|}{|p_n-p|} = \lim_{n\to\infty} \frac{|x_{n+1}-x_*|}{|x_n-x_*|} \tag{24}$$
$$= \lim_{n\to\infty} g'(c) \tag{25}$$
$$= const \tag{26}$$

Therefore, this sequence is linearly convergent.

5 Problem 5

5.a

Figure 2: Plot of $x - 4sin(2x) - 3 = 0$



Using external graphing software (Desmos), I determined the roots to be r ∈ (-0.898, -0.544, 1.732, 3.162, 4.518).

5.b

I used the fixed point iteration on $g(x) = -sin(2x) + \frac{5x}{4} - \frac{3}{4}$. Additionally, I printed the value of $g'(x) = -2cos(2x) + \frac{5}{4}$ for each initial guess $x0$ in order to determine the behavior of the algorithm.

Using the footnote provided, I set the error tolerance level to be $tol = 0.5 * 10^{-10}$. The results of the iterations in python are provided below:

x0	root found	g'(x)
-.88	-0.5444424006959225	1.6261536777857604
-.54	-0.5444424006612157	0.30734327165252007
1.8	3.1618264865314685	3.0435168326682938
3.1	3.161826486535402	-0.743084194046435
4.6	diverges	3.199687242808327

Evidently, in cases where $|g'(x)| < 1$, the fixed point iteration will converge and find the proper root. However, when $|g'(x)| > 1$ the iteration diverges and will not find the root.

All code for this question can be found in 6.

6 Appendix

6.1 Code

Description of Code Here:

```python
import numpy as np
import math
import matplotlib.pyplot as plt

# Question 1

def question1():
    print("-----Question 1------")

    x = np.linspace(-1,math.pi,100)
    y_1 = np.array([2*X - 1 for X in x])
    y_2 = np.array([math.sin(X) for X in x])

    plt.plot(x,y_1)
    plt.plot(x,y_2)
    plt.savefig("HW3.1.a.png")
    plt.clf()

# use routines

    # Adjust inputs here here
    f = lambda x: (2*x) - 1 - math.sin(x)
    a = 0
    b = math.pi/2

#     f = lambda x: np.sin(x)
#     a = 0.1
#     b = np.pi+0.1

    tol = 1e-8

    [astar,ier,count] = bisection(f,a,b,tol)
    print('the approximate root is',astar)
    print('the error message reads:',ier)
    print('f(astar) =', f(astar))
    print('number of iterations = ', count)




# define routines
def bisection(f,a,b,tol):

#     Inputs:
#      f,a,b       - function and endpoints of initial interval
#       tol  - bisection stops when interval length < tol

#     Returns:
#       astar - approximation of root
#       ier   - error message
```

```
#             - ier = 1 => Failed
#             - ier = 0 == success

#     first verify there is a root we can find in the interval
    count = 0

    fa = f(a)
    fb = f(b)
    if (fa*fb>0):
       ier = 1
       astar = a
       return [astar, ier, count]

#   verify end points are not a root
    if (fa == 0):
      astar = a
      ier =0
      return [astar, ier, count]

    if (fb ==0):
      astar = b
      ier = 0
      return [astar, ier, count]

    d = 0.5*(a+b)
    while (abs(d-a)> tol):
      fd = f(d)
      if (fd ==0):
        astar = d
        ier = 0
        return [astar, ier, count]
      if (fa*fd<0):
         b = d
      else:
        a = d
        fa = fd
      d = 0.5*(a+b)
      count = count +1
#      print('abs(d-a) = ', abs(d-a))

    astar = d
    ier = 0
    return [astar, ier, count]

# question1()

# Question 2

def question2():
    print("-----Question 2-----")

# use routines
    print("Function 1:")
    # Adjust inputs here here
    f = lambda x: (x-5)**9
    a = 4.82
    b = 5.2

```

```
#     f = lambda x: np.sin(x)
#     a = 0.1
#     b = np.pi+0.1

    tol = 1e-4

    [astar,ier,count] = bisection(f,a,b,tol)
    print('the approximate root is',astar)
    print('the error message reads:',ier)
    print('f(astar) =', f(astar))
    print('f(a), f(b) =', f(a), f(b))
    print('number of iterations = ', count)

    print("Function 2 (Expanded Version):")
    # Adjust inputs here here
    f = lambda x: x**9 - 45*(x**8) + 900*(x**7) - 10500*(x**6) + ...
        78750*(x**5) - 393750*(x**4) + 1312500*(x**3) -2812500*(x**2) ...
        + 315625*x - 1953125
    a = 4.82
    b = 5.2

#     f = lambda x: np.sin(x)
#     a = 0.1
#     b = np.pi+0.1

    tol = 1e-4

    [astar,ier,count] = bisection(f,a,b,tol)
    print('the approximate root is',astar)
    print('the error message reads:',ier)
    print('f(astar) =', f(astar))
    print('f(a), f(b) =', f(a), f(b))
    print('number of iterations = ', count)

# question2()

# Question 3

def question3():
    print("-----Question 3-----")

# use routines
    print("Function 1:")
    # Adjust inputs here here
    f = lambda x: x**3 + x - 4
    a = 1
    b = 4

#     f = lambda x: np.sin(x)
#     a = 0.1
#     b = np.pi+0.1

    tol = 1e-3

    n = math.ceil(math.log2((b-a)/tol) - 1)
    print('expected upper bound on number of iterations: ', n)

    [astar,ier,count] = bisection(f,a,b,tol)
```

```
    print('the approximate root is',astar)
    print('the error message reads:',ier)
    print('f(astar) =', f(astar))
    print('f(a), f(b) =', f(a), f(b))
    print('number of iterations = ', count)

    print("Function 2 (Expanded Version):")
    # Adjust inputs here here
    f = lambda x: x**9 - 45*(x**8) + 900*(x**7) - 10500*(x**6) + ...
        78750*(x**5) - 393750*(x**4) + 1312500*(x**3) -2812500*(x**2) ...
        + 315625*x - 1953125
    a = 4.82
    b = 5.2

# question3()

# Question 5

def question5():
    print("-----Question 5-----")
    # function
    f1 = lambda x: x - (4*math.sin(2*x)) - 3

    x = np.linspace(-2,8,200)
    y = np.array([f1(X) for X in x])

    plt.plot(x,y)
    plt.plot(x,[0 for X in x])
    plt.savefig("HW3.5.a.png")
    plt.clf()




    f2 = lambda x: -np.sin(2*x) + ((5*x)/4) - (3/4)
    f2_deriv = lambda x: -2*math.cos(2*x) + (5/4)

    Nmax = 100
    tol = 0.5*(10**(-10))

# test f1 '''
    x0 = 4.6
    [xstar,ier] = fixedpt(f2,x0,tol,Nmax)
    print('the approximate fixed point is:',xstar)
    print('f1(xstar):',f2(xstar))
    print('f1_deriv(x0):', f2_deriv(x0))
    print('Error message reads:',ier)



# define routines
def fixedpt(f,x0,tol,Nmax):

    ''' x0 = initial guess'''
    ''' Nmax = max number of iterations'''
    ''' tol = stopping tolerance'''

    count = 0
```

```
    while (count <Nmax):
        count = count +1
        x1 = f(x0)
        if (abs(x1-x0) < tol):
            xstar = x1
            ier = 0
            return [xstar,ier]
        x0 = x1

    xstar = x1
    ier = 1
    return [xstar, ier]

# question5()
```

NOTE: CODE FOR THIS ASSIGNMENT IS ORIGINAL, WRITTEN BY JESSE HETTLEMAN. THE BISECTION METHOD AND FIXED POINT ITERATION CODE IS ADAPTED FROM CLASS EXAMPLES.