Exercise 1

Write a function that take into argument a function, an interval, a tolerance, and return zero of the function using the bisection method. Then, compute the root of $f(x) = x^6 - x - 1.0$ on the domain $[1, 2]$ using the bisection method with tolerance $5 \cdot 10^{-4}$. It should take 11 iterations and the root is 1.1342773437

Work

For this exercise I wrote the appropriate Python code for the bisection method and then called it to find the root of the function.

Python 3 Code

```
def bisect(f, a, b, tol, itmax):
    count = 0
    while abs(b - a) >= tol and abs(f((a + b)/2)) >= tol and count < itmax:
        c = (a + b)/2
        if f(c) == 0:
            return c
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
        count += 1
    return c, count


# Exercise 1
f = lambda x: x**6 - x - 1.0 # root at about -0.75
print("Root for exercise 1: ", bisect(f, 1, 2, 5*10**-4, 1000)[0], end="\n\n")
```

Solution

Root for exercise 1: 1.13671875 Iterations for exercise 1: 8

Exercise 2

Use bisection method and graph of $f(x)$ to find all the roots of $f(x) = 32x^6 - 48x^4 + 18x^2 - 1$ within 10^{-10} accuracy.

Work

For this problem, I first graphed the given function with matplotlib, and then used the graph to estimate it's roots. I then used the estimates to call the bisection method for a more accurate approximation.

Python 3 Code

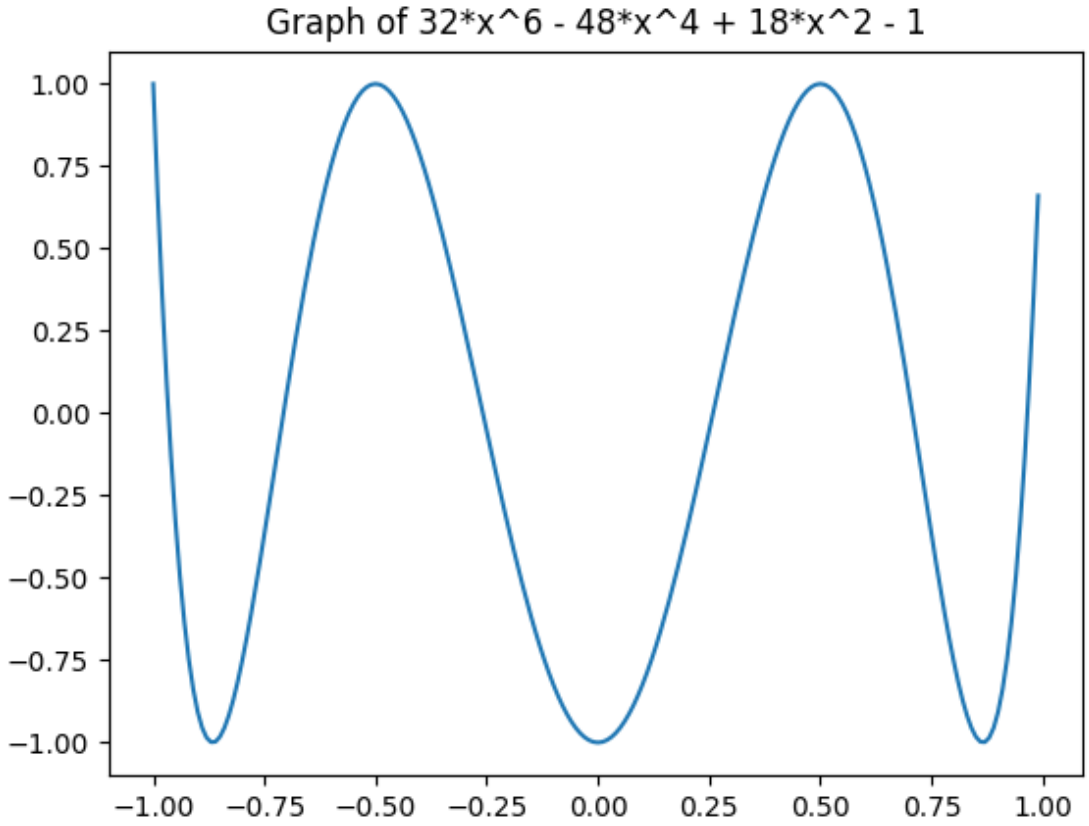
```
def graph(f, div, iter):
    x = [i/div for i in range(iter[0],iter[1])]
    y = [f(i) for i in x]
    plt.plot(x, y)
    plt.title("Graph of 32*x^6 - 48*x^4 + 18*x^2 - 1")
    plt.show()

 Exercise 2
# Estimated roots at -1, -0.75, -0.25, 0.25 0.75, 1
g = lambda x: 32*x**6 - 48*x**4 + 18*x**2 - 1
graph(g, 100, [-100, 100])

# List of ranges for roots
lst = [[-1.2, -0.8], [-0.95, -0.55], [-0.45, -0.05],
        [0.05, 0.45], [0.55, 0.95], [0.8, 1.2]]

print("Roots for exercise 2: ")
for i in range(len(lst)):
    print("Root Number ", i+1, ":\t", bisect(g, lst[i][0], lst[i][1], 10**-10, 1000)[0],
print()
```

Solution




```
Roots for exercise 2:
Root Number 1: -0.9659258262254298
Iterations for exercise 2: 32
Root Number 2: -0.7071067813783883
Iterations for exercise 2: 30
Root Number 3: -0.2588190450333059
Iterations for exercise 2: 32
Root Number 4: 0.2588190450333059
Iterations for exercise 2: 32
Root Number 5: 0.7071067813783883
Iterations for exercise 2: 30
Root Number 6: 0.9659258262254298
Iterations for exercise 2: 32
```

Exercise 3

Let be the largest root of $f(x) = e^x - x - 2$. Find an interval $[a, b]$ containing α and for which the bisection method will converge to α. Then compute the root using your bisection function within an accuracy of $5 \cdot 10^{-8}$.

Work

Here I use the already created bisection method and call it with new parameters to find our results.

Python 3 Code

```
# Exercise 3
h = lambda x: exp(x) - x - 2
print("Root for exercise 3:", bisect(h, 1, 1.3, 5*10**-8, 1000), end="\n\n")
```

Solution

Root for exercise 3: 1.1461931943893433 Iterations for exercise 3: 22

BONUS

Estimate the number of iterates needed to find α within an accuracy of $5 \cdot 10^{-8}$ using the formula $|\alpha - c| \leq \frac{1}{2^n}|b - a|$ and compare to what you found.

Work

$$-log_2(\frac{|x^* - C_n|}{|b - a|}) >= n => -log_2(\frac{|5 * 10^-8|}{|.3|}) >= n$$

Solution

$$n <= 22$$

Exercise 4

Write a function that take into argument a function, its derivative, an initial guess, a tolerance, a maximum number of iteration and return the zero of the function using Newtons method. Then, compute the root of $f(x) = x^6 - x - 1.0$ using Newtons method with tolerance 10^{-10} and with starting point 1.0. It should take 6 iterations and the root is 1.1347

Work

In this problem, I wrote up the code for the newtons method and placed it into our lib.py file. Once created, I call the function with it's proper parameters for the exercise.

Python 3 Code

```
def newtons(f, fprime, x, tol, itmax, its=1):
    if abs(f(x)) < tol or its > itmax:
        return x, its
    x = x - f(x)/fprime(x)
    return newtons(f, fprime, x, tol, itmax, its+1)


#Exercise 4
f = lambda x: x**6 - x - 1.0
fprime = lambda x: 6*x**5 - 1
print("Root for exercise 4:", newtons(f, fprime, 1.0, 10**-10, 1000)[0], end="\n\n")
```

Solution

Root for exercise 4: 1.134724138401536
Iterations for exercise 4: 6

Exercise 5

The equation $x + e^{-\beta \cdot x^2} cos(x)$ has a unique root in the interval $[-1, 1]$. Use Newtons method to find it as accurately as possible. Use the values of $B = 1, 5, 10, 25, 50$. What should be a good choice for $x0$? Explain the behavior observed in the iterates for the larger values of β.

Work

In this exercise, I begin by creating a list of the value B's that were given, along with estimates for the roots. I then loop through those while using the newtons function call to find the roots.

Python 3 Code

```
#Exercise 5
B = [1, 5, 10, 25, 50]
xs = [-0.5, -0.4, -0.3, -0.2, -0.15]

x = sp.symbols('x')
gs = [x + sp.exp(-i * x**2) * sp.cos(x) for i in B]
gPrimes = [sp.lambdify(x, sp.diff(i)) for i in gs]
gs = [sp.lambdify(x, g) for g in gs]

print("Roots for exercise 5: ")
lst = [newtons(g, gPrime, x, 10**-2, 100)[0] for g, gPrime, x in zip(gs, gPrimes, xs)]
for root in lst:
    print("Root Number ", i+1, ":\t", root, sep="")
print()

print("Exercise 5 Explanations")
print("A good choice for x0 is -0.5 since it is close to the actual 0")
print("As the values of B increases, the roots of the functions decrease while approachi
print()
```

Solution

Roots for exercise 5:
Root Number 1: -0.5891960863101156
Iteration Number for Root 1: 2

Root Number 2: -0.4048965911080112
Iteration Number for Root 2: 2

Root Number 3: -0.3256217639939549
Iteration Number for Root 3: 2

Root Number 4: -0.237408271591381
Iteration Number for Root 4: 3

Root Number 5: -0.18320828325981584
Iteration Number for Root 5: 3

Exercise 5 Explanations
A good choice for x0 is -0.5 since it is close to the actual 0
As the values of B increases, the roots of the functions decrease while approaching 0

Exercise 6

Solve the equation $x^3 - 3x^2 + 3x - 1 = 0$ using Newton's Method. (as accurately as possible, this means tolerance = 10^{-16}.

Work

This one was straightforward. I did another function call with Newton's using the right parameters.

Python 3 Code

```
#Exercise 6
h = lambda x: x**3 - 3*x**2 + 3*x - 1
hPrime = lambda x: 3*x**2 - 6*x + 3
print("Root for exercise 6:", newtons(h, hPrime, 1.2, 10**-16, 100)[0], end="\n\n")
```

Solution

Root for exercise 6: 1.0000039276336283
Iterations for exercise 6: 27

Exercise 7

Give Newtons method for finding $\sqrt[m]{a}$, with $a > 0$ and m a positive integer. Apply it to finding $\sqrt[m]{2}$ for $m = 3, 4, 5, 6, 7, 8$, say to six significant digits.

Work

Problem 7 was pretty neat. I created a function that turned a given root into $x^m - a = 0$ and used Newton's to solve for that root.

Python 3 Code

```
def solveRoot(m, a, guess):
    x = sp.symbols('x')

    f1 = x**m - a
    f1prime = sp.diff(f1)
    f1 = sp.lambdify(x, f1)
    f1prime = sp.lambdify(x, f1prime)

    return newtons(f1, f1prime, guess, 10**-16, 100)


# Exercise 7 x**m - a = 0
print("Root for exercise 7:", solveRoot(3, 2, 1)[0], end="\n\n")
```

Solution

Root for exercise 7: 1.2599210498948732
Iterations for exercise 7: 6

Exercise 8

Solve the equation $x^3 - 3x^2 + 3x - 1 = 0$ using the secants method. (as accurately as possible, this means tolerance = 10^{-16})

Work

For this problem I wrote the function for the Secant method and then called the function with the appropriate parameters.

Python 3 Code

```
def secant(f, x0, x1, tol, itmax, its=1):
    if abs(f(x1)) < tol or its > itmax or abs(x1-x0) < tol:
        return x1, its
    remainder = f(x1) * (x1 - x0)/(f(x1) - f(x0))
    x0 = x1
    x1 = x1 - remainder
    return secant(f, x0, x1, tol, itmax, its+1)


# Exercise 8
f = lambda x: x**3 - 3*x**2 + 3*x - 1
print("Root for exercise 8:", secant(f, 0.8, 1.1, 10**-15, 100)[0], end="\n\n")
```

Solution

Root for exercise 8: 1.00001100782391
Iterations for exercise 8: 33

Exercise 9

Convert the equation $x^2 - 5 = 0$ to the fixed-point problem $x = x + c(x^2 - 5)$ with c a non zero constant. Determine the possible values of c to ensure convergence. (The true solution is $\alpha = \sqrt{5}$.

Work

With this problem, the trickiest part was not writing the function, but instead looking for the working values of c. I had trouble finding the positive solution, until I used a negative value for c. Besides this, I simply wrote the code for the function, then made the function call.

Python 3 Code

```
def fixedPoint(g, x0, tol, itmax):
    x1 = g(x0)
    it = 0

    while abs(g(x1) - x1) >= tol and abs(x1 - x0) >= tol and it < itmax:
        x0, x1 = x1, g(x1)
        it += 1
    return x1, it


# Exercise 9
g = lambda x: x + 0.2238*(x**2 - 5)
print("Negative Root for exercise 9:", fixedPoint(g, 1.5, 10**-16, 1000)[0], end="\n\n")
g = lambda x: x + (-0.4*(x**2 - 5))
print("Positive Root for exercise 9:", fixedPoint(g, 2.4, 10**-16, 1000)[0], end="\n\n")
```

Solution

```
Negative Root for exercise 9: [0]
Iterations for exercise 9.1: 9

Positive Root for exercise 9: 2.2360679774997894
Iterations for exercise 9.2: 1000
```

BONUS: Exercise 10

For $x^6 - x - 1.0 = 0$, compute the convergence of the Newtons method, Secant Method, Bisection Method with initial guess 2.0 and comparing each of them to the solution 1.13472. Plot in the x axis the iteration number and in the y axis $\log|\alpha - x_n|$, what slope do you expect? (look mostly at the first 10 iterations)

Work

This was a plotting problem, and so I followed the instrutcions and went through Bisection, Netwons, and Secant from iterations 1 to 20 and graphed the appropriate values to determine the efficiency of each method, comparatively.

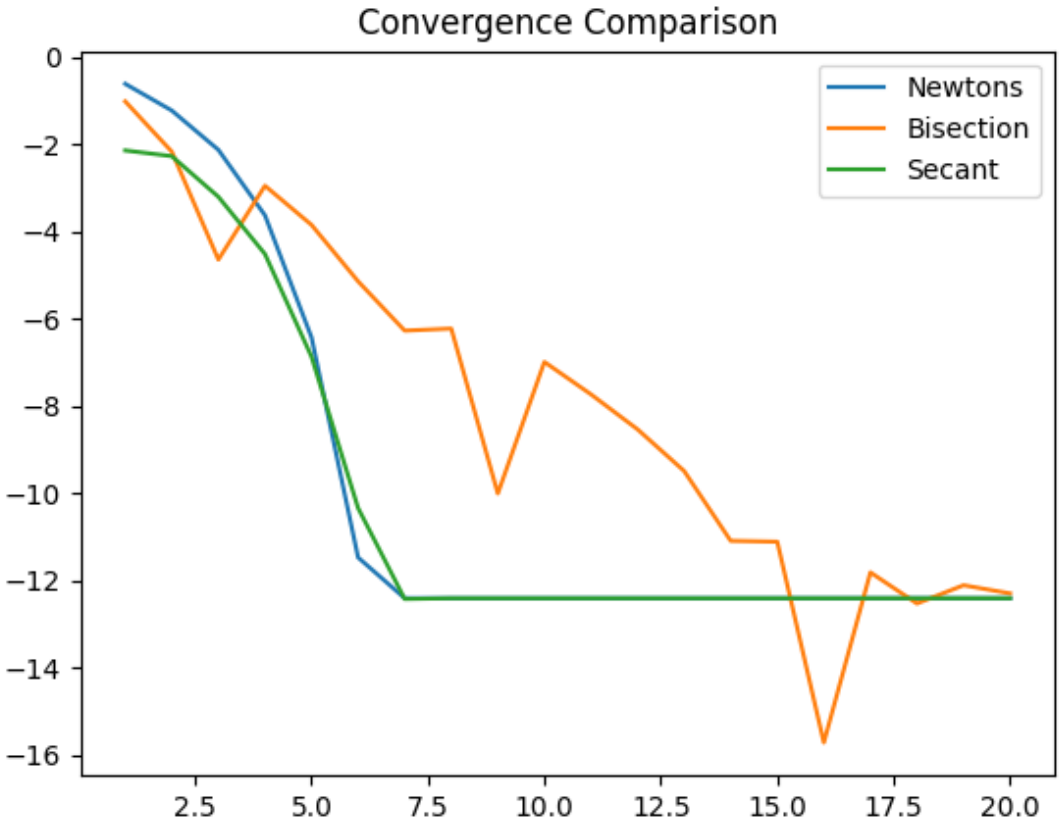
Python 3 Code

```
def addToPlot(f, params, name):
    dct={}
    for i in range(1, 21):
        dct[i] = log(abs(f(params[0], params[1], params[2], params[3], i)[0] - 1.13472))
    print(dct)
    import matplotlib.pyplot as plt
    x = list(dct.keys())
    y = list(dct.values())
    plt.plot(x, y)
    plt.legend(["Newtons", "Bisection", "Secant"])


# Exercise 10
f = lambda x: x**6 - x - 1
fprime = lambda x: 6*x**5 -1

addToPlot(newtons, [f, fprime, 2.0, 10**-16], "Newtons")
addToPlot(bisect, [f, 1.0, 2.0, 5*10**-16], "Bisection")
addToPlot(secant, [f, 1.0, 2.0, 5*10**-16], "Secant")
plt.title("Convergence Comparison")
plt.show()
```

Solution



By analysing the graph we can see that the lower the average slope, the faster it approaches the number we're looking for. When the graph flatlines, it means we have reached our root. From this we can infer that even though the Secant method starts out stronger, Newtons Method obtains the final result faster. We can also see that Bisection performs poorly overall, compared to the other two.