Coding Report

Joe Lenning

1 Problem Description:

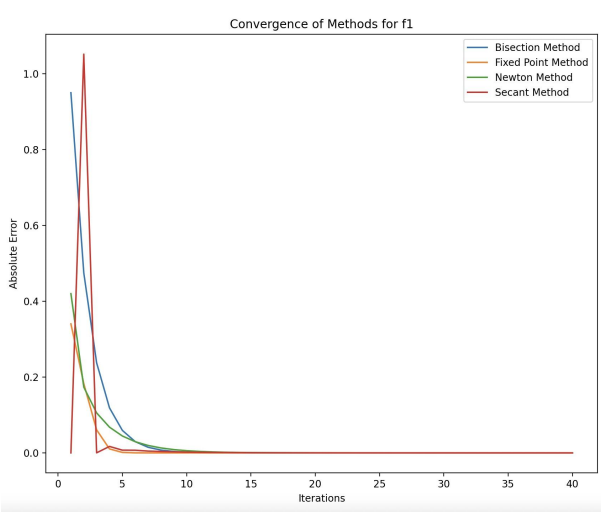
This report demonstrates a method for finding solutions to equations. We are going to be using the following methods: Bisection Method, Fixed Point, Newton, and Secant. The Bisection Method involves iterating over an interval from a/b, we will continuously find the midpoint of this interval and repeat this until we find the root. The Fixed Point Method involves picking a number somewhat near to the actual root of a function. Then plug that value into the function to come up with a new number. Then repeat this process not forgetting to add the previous term to the function. The Newton Method involves picking some initial point on the function and taking the tangent line to it, this may be somewhat close to the root. Then we will the end directly about the tangential line and repeat this process until we have found the root. And lastly, we have come to the secant method. This method is similar to the Newton Method but instead of taking tangent lines, we will take secant lines to find the root.

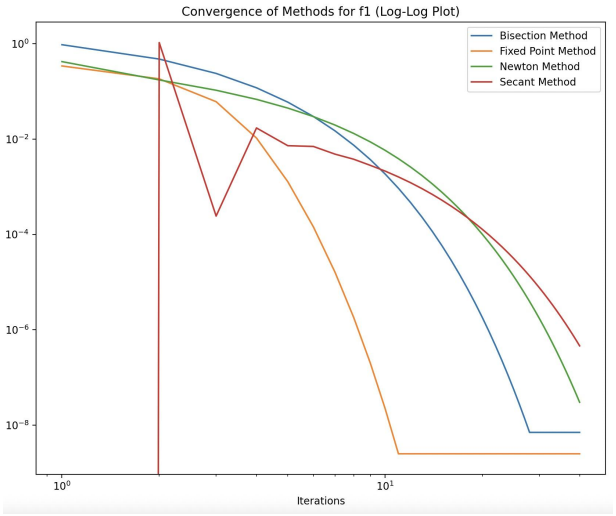
****Note: For all of these Methods we are looking for only 1 root. If there are multiple roots in the function, this could prove difficult or impossible for some of the methods.

2 Results

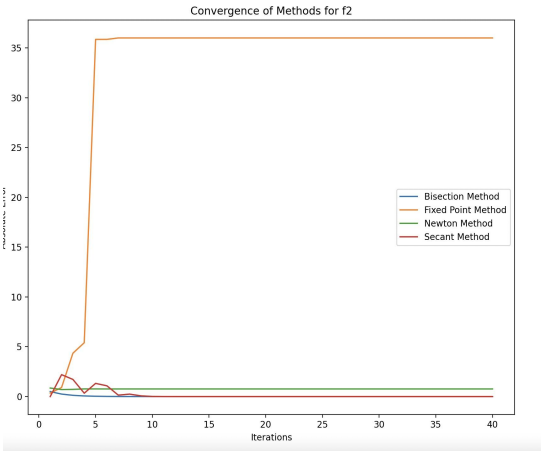
In all the root-finding methods for f1 and f2, we used 40 iterations and an tolerance of 1e-8. While all four methods certainly didn't need 40 iterations to converge, with respect to the graph it certainly did help the visualization process. With respect to the f1(x) = (1/2) * math.sin(x-1) ** 3 we saw that we did find an answer roughly within 5-10 iterations. However, with respect to f2(x) = 6**(-x) -2, I found that it converged within 10 iterations. However, I found that with the fixed-point method we could not find an answer that converged. The reason for this was(from the book) that there exist some domain issues on this function. To combat you must you some limit techniques in order to find a solution. However, with this algorithm, there was no way to find a numerical solution. We can see the graphical solutions below.

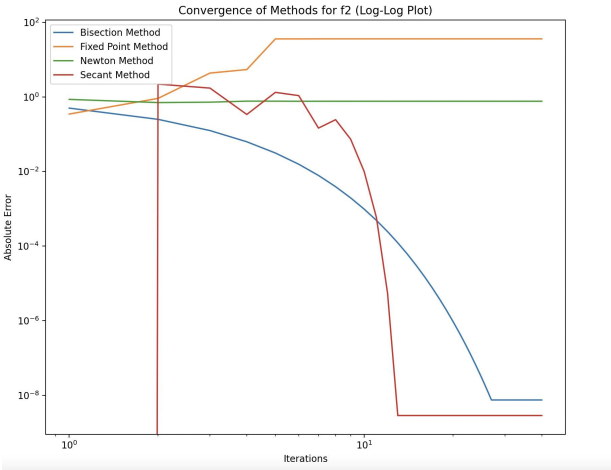
Graphs of Function 1





From the first graph on the left we can see that all four methods used converge within 10 iterations.





All method except for the fixed point method(as previously mentioned -> no numerical solution) convered within 10 iterations.

3 Collaboration:

None

4 Academic Integrity

On my personal integrity as a student and member of the UCD community, I have not

given nor received any unauthorized assistance on this assignment.

5 Appendix

```
import math
import matplotlib.pyplot as plt

def f1(x):
   f1x = (1/2) * math.sin(x-1) ** 3
   return f1x

def f2(x):
   f2x = (6 ** (-x)) - 2
   return f2x

def f1p(x):
   f1px = (3/2) * math.cos(x-1) * (math.sin(x-1) ** 2)
   return f1px

def f2p(x):
   fp2x = (math.log10(6) / (6 ** x)) * -1
   return fp2x


#-----------------------------------------------------------------------

def bisection_method(func, a, b, error_accept, iterations):
   fa = func(a)
   i = 1
   my_error = 0
   while i <= iterations:
       p = a + ((b-a)/2)
       fp = func(p)
       my_error = (b-a)/2
       if fp == 0 or my_error < error_accept:
           return p, my_error
       i = i + 1
       if fa * fp > 0:
           a = p
           fa = fp
       else:
           b = p
   return 'Failed', my_error
#-----------------------------------------------------------------------

#some issue with |g'(x)| ≤ k,
#f2p = -ln(6)6^
def fixed_point_method(func, x_start, error_accept, iterations):
```

```
    i = 1
    my_error = 0
    while i <= iterations:
        p = func(x_start)
        my_error = abs(p - x_start)
        if my_error < error_accept:
            return p, my_error
        i = i + 1
        x_start = p
    return 'Failed', my_error

#-----------------------------------------------------------------------

def newton_method(func, fprime, x_start, error_accept, iterations):
    i = 1
    my_error = 0
    while i <= iterations:
        p = x_start - (func(x_start) / fprime(x_start))
        my_error = abs(p - x_start)
        if my_error < error_accept:
            return p, my_error
        i = i + 1
        x_start = p
    return 'Failed', my_error

#-----------------------------------------------------------------------

def secant_method(func, po, p1, error_accept, iterations):
    i = 2
    q_0 = func(po)
    q_1 = func(p1)
    p = 0
    my_error = 0
    while i <= iterations:
        p = p1-q_1*((p1-po)/ (q_1 - q_0))
        my_error = abs(p-p1)
        if my_error < error_accept:
            return p, my_error
        i += 1
        po = p1
        q_0 = q_1
        p1= p
        q_1 = func(p)
    return 'Failed', my_error

def graph_errors(errors, title_func):
    plt.figure(figsize=(10, 8))
    plt.plot(x, errors["errors_bisect"], label='Bisection Method')
    plt.plot(x, errors["errors_fixed"], label='Fixed Point Method')
```

```
    plt.plot(x, errors["errors_newtons"], label='Newton Method')
    plt.plot(x, errors["errors_secant"], label='Secant Method')
    plt.xlabel('Iterations')
    plt.ylabel('Absolute Error')
    plt.title(f'Convergence of Methods for {title_func}')
    plt.legend()
    plt.show()

def graph_log_errors(errors, title_func):
    plt.figure(figsize=(10, 8))
    plt.loglog(x, errors["errors_bisect"], label='Bisection Method')
    plt.loglog(x, errors["errors_fixed"], label='Fixed Point Method')
    plt.loglog(x, errors["errors_newtons"], label='Newton Method')
    plt.loglog(x, errors["errors_secant"], label='Secant Method')
    plt.xlabel('Iterations')
    plt.ylabel('Absolute Error')
    plt.title(f'Convergence of Methods for {title_func} (Log-Log Plot)')
    plt.legend()
    plt.show()



if __name__ == "__main__":

    iterations = 40
    x = list(range(1, iterations + 1))

    # Define the error_accept value
    ERROR_ACCEPT = 1e-8

    # Calculate the errors for each method and each function
    f1_errors = {"errors_bisect": [], "errors_fixed": [], "errors_newtons": [],
"errors_secant": []}
    f2_errors = {"errors_bisect": [], "errors_fixed": [], "errors_newtons": [],
"errors_secant": []}

    for i in range(1, iterations + 1):
        f1_errors["errors_bisect"].append(
            bisection_method(f1, 0.1, 2, ERROR_ACCEPT, i)[1]
        )
        f1_errors["errors_fixed"].append(
            fixed_point_method(f1, 0.1, ERROR_ACCEPT, i)[1]
        )
        f1_errors["errors_newtons"].append(
            newton_method(f1, f1p, 0.1, ERROR_ACCEPT, i)[1]
        )
        f1_errors["errors_secant"].append(
            secant_method(f1, 0.1, 2, ERROR_ACCEPT, i)[1]
        )
```

```
print(f1_errors)
graph_errors(f1_errors, "f1")
graph_log_errors(f1_errors, "f1")

for i in range(1, iterations + 1):
    f2_errors["errors_bisect"].append(
        bisection_method(f2, 0, 1, ERROR_ACCEPT, i)[1]
    )
    f2_errors["errors_fixed"].append(
        fixed_point_method(f2, -.378, ERROR_ACCEPT, i)[1]
    )
    f2_errors["errors_newtons"].append(
        newton_method(f2, f2p, -1, ERROR_ACCEPT, i)[1]
    )
    f2_errors["errors_secant"].append(
        secant_method(f2, 0, 1, ERROR_ACCEPT, i)[1]
    )

print(f2_errors)
graph_errors(f2_errors, "f2")
graph_log_errors(f2_errors, "f2")
```