

Root Finding and Fixed Point Methods:

- **Due:** 11 Sep 2019
- Implement the Bisection Method, Newton's Method, and Secant Method to find roots of $f(x) = x^2 - e^{2-x^2}$.
- Implement the Fixed Point Method to find where $f(x) = (x + 10)^{\frac{1}{4}}$.
- Check accuracy and order of convergence of each method for the specified examples.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 2 | Bisection Method | 3 |
| 2.1 | Code | 3 |
| 2.2 | Output | 5 |
| 2.3 | Accuracy and Order of Convergence | 5 |
| 3 | Newton's Method | 7 |
| 3.1 | Code | 7 |
| 3.2 | Output | 8 |
| 3.3 | Accuracy and Order of Convergence | 9 |
| 4 | Secant Method | 10 |
| 4.1 | Code | 10 |
| 4.2 | Output | 11 |
| 4.3 | Accuracy and Order of Convergence | 12 |
| 5 | Fixed Point Method | 13 |
| 5.1 | Code | 13 |
| 5.2 | Output | 15 |
| 5.3 | Accuracy and Order of Convergence | 15 |

1 Introduction

The following sections will describe the implementation of four numerical methods: Bisection, Newton's, Secant, and Fixed Point. All methods were implemented with Python 3.7 and run in an Anaconda command line prompt. Each root finding method approximates the root of $f(x) = x^2 - e^{2-x^2}$ and the Fixed Point Method approximates the intersection of $f(x) = x$ and $g(x) = (x + 10)^{\frac{1}{4}} = x$.

Order of convergence is approximated for each example by

$$p \approx \frac{\log \left| \frac{x_{n+1} - x_n}{x_n - x_{n-1}} \right|}{\log \left| \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}} \right|}$$

2 Bisection Method

The following code defines a function called "find_root" that takes in a user specified interval, i.e. $[a, b]$, and an allowable error ϵ . A while loop is used to iterate until the calculated error is less than or equal to the allowable error.

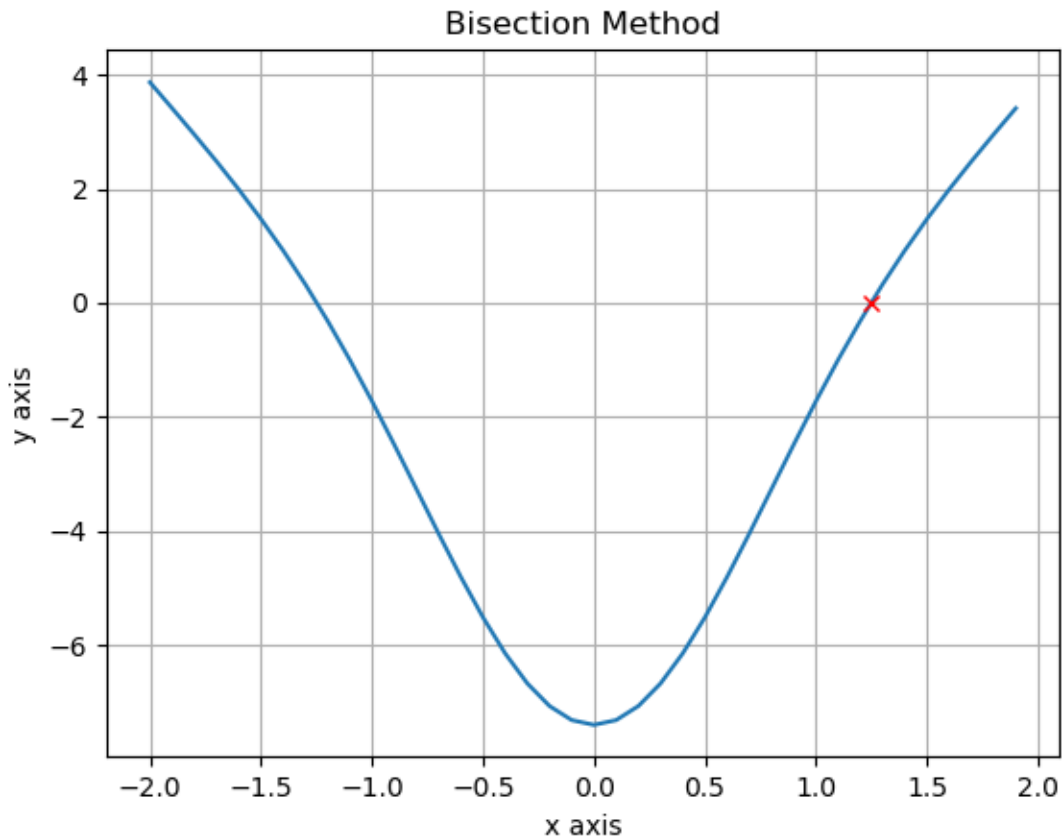
2.1 Code

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 my_function = lambda x: (x)**2 - np.exp(2-x**2)
5 true_root = 1.24786
6
7 def find_root(a, b, epsilon):
8     error = (b-a)/2
9     c = (a+b)/2
10    n = 0
11    n_array = []
12    true_error_array = []
13    while (error > epsilon):
14        if (my_function(b)*my_function(c) <= 0):
15            a = c
16            error = (b-a)/2
17            c = (a+b)/2
18            n = n + 1
19            n_array.append(n)
20            true_error = abs(true_root - c)
21            true_error_array.append(true_error)
```

```
22         print('iteration:', n, " - error:", error)
23         continue
24     else:
25         b = c
26         error = (b-a)/2
27         c = (a+b)/2
28         n = n + 1
29         n_array.append(n)
30         true_error = abs(true_root - c)
31         true_error_array.append(true_error)
32         print('iteration:', n, " - error:", error)
33         continue
34     print('error:', error)
35     print('approximate root:', c)
36     converge_rate = (np.log(abs((true_error_array[n-1] -
37                             true_error_array[n-2])/
38                             (true_error_array[n-2] -
39                             true_error_array[n-3]))))/
40                     (np.log(abs((true_error_array[n-2] -
41                             true_error_array[n-3])/
42                             (true_error_array[n-3] -
43                             true_error_array[n-4]))))
44     print("order of convergence:", converge_rate)
45     return c, n_array, true_error_array
46
47
48 # Plot of function and root
49 root = find_root(-10,11,0.000001)
50 root_0 = 0
51 x = np.arange(-2,2,0.1)
52 y = (x)**2-np.exp(2-x**2)
53 plt.title("Bisection Method")
54 plt.xlabel("x axis")
55 plt.ylabel("y axis")
56 plt.plot(x, y)
57 plt.plot(root[0], root_0, 'rx')
58 plt.grid(True)
59 plt.show()
60
61
62 # Plot of error vs iterations
63 plt.title("Bisection Method Error")
64 plt.xlabel("n iterations")
65 plt.ylabel("true error")
66 plt.plot(root[1], root[2], 'ro')
```

```
67 plt.grid(True)
68 plt.show()
```

2.2 Output

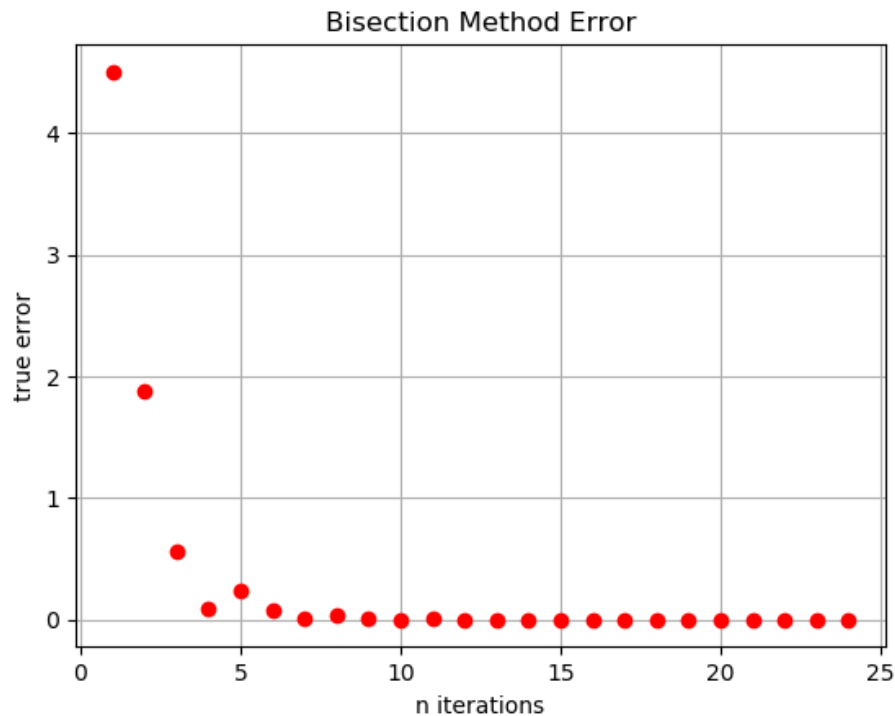


2.3 Accuracy and Order of Convergence

The last approximation for the root was within 6.258487×10^{-7} of the true value. The approximated value for the root of the function is 1.247856. The order of convergence for this method is 1.0.

```
Anaconda Prompt
(base) C:\Users\saeth\Documents\Courses\2019\Fall\MATH 5610>python Bisection_Method.py
iteration: 1 - error: 5.25
iteration: 2 - error: 2.625
iteration: 3 - error: 1.3125
iteration: 4 - error: 0.65625
iteration: 5 - error: 0.328125
iteration: 6 - error: 0.1640625
iteration: 7 - error: 0.08203125
iteration: 8 - error: 0.041015625
iteration: 9 - error: 0.0205078125
iteration: 10 - error: 0.01025390625
iteration: 11 - error: 0.005126953125
iteration: 12 - error: 0.0025634765625
iteration: 13 - error: 0.00128173828125
iteration: 14 - error: 0.000640869140625
iteration: 15 - error: 0.0003204345703125
iteration: 16 - error: 0.00016021728515625
iteration: 17 - error: 8.0108642578125e-05
iteration: 18 - error: 4.00543212890625e-05
iteration: 19 - error: 2.002716064453125e-05
iteration: 20 - error: 1.0013580322265625e-05
iteration: 21 - error: 5.0067901611328125e-06
iteration: 22 - error: 2.5033950805664062e-06
iteration: 23 - error: 1.2516975402832031e-06
iteration: 24 - error: 6.258487701416016e-07
error: 6.258487701416016e-07
approximate root: 1.2478561103343964
order of convergence: 1.0
(base) C:\Users\saeth\Documents\Courses\2019\Fall\MATH 5610>
```

The distance of each approximation to the true value with each iteration n :



3 Newton's Method

The following code was adapted from the Bisection Method. It was changed to only require one initial guess instead of an interval containing a root. A while loop iterates until $x_n = x_n + \frac{f(x_n)}{f'(x_n)}$ is within the user specified allowable error ϵ .

3.1 Code

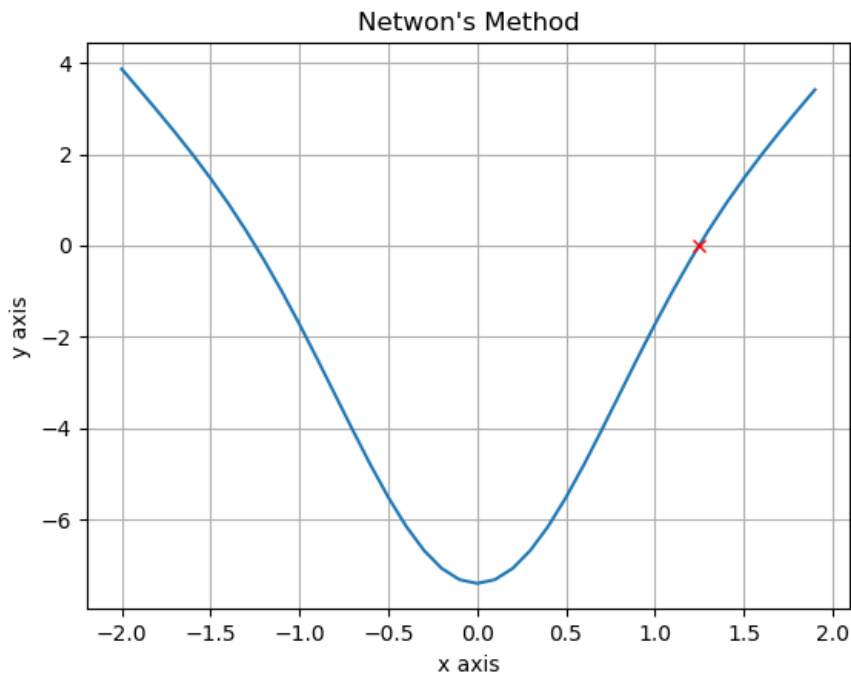
```

1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 my_function = lambda x: (x)**2-np.exp(2-x**2)
5 d_my_function = lambda x: 2*x*(1 + np.exp(2-x**2))
6 true_root = 1.24786
7
8 def find_root_newton(x_0, epsilon):
9     error = 1
10    n = 0
11    n_array = []
12    true_error_array = []
13    while (error > epsilon):
14        x_n = x_0 - (my_function(x_0)/d_my_function(x_0))
15        error = abs(x_0 - x_n)
16        x_0 = x_n
17        n = n + 1
18        n_array.append(n)
19        true_error = abs(true_root - x_0)
20        true_error_array.append(true_error)
21        print('iteration:', n, " - error:", error)
22        continue
23    print('error:', error)
24    print('approximate root:', x_n)
25    converge_rate = (np.log(abs((true_error_array[n-1] -
26        true_error_array[n-2])/
27        (true_error_array[n-2] -
28        true_error_array[n-3]))))/
29        (np.log(abs((true_error_array[n-2] -
30        true_error_array[n-3])/
31        (true_error_array[n-3] -
32        true_error_array[n-4])))))
33    print("order of convergence:", converge_rate)
34    print("order of convergence:", converge_rate)
35    return x_n, n_array, true_error_array

```

```
36
37
38 # Plot the function and root
39 root = find_root_newton(1.2,0.000001)
40 root_0 = 0
41 x = np.arange(-2,2,0.1)
42 y = (x)**2-np.exp(2-x**2)
43 plt.title("Netwon's Method")
44 plt.xlabel("x axis")
45 plt.ylabel("y axis")
46 plt.plot(x, y)
47 plt.plot(root[0], root_0, 'rx')
48 plt.grid(True)
49 plt.show()
50
51 # Plot of error vs iterations
52 plt.title("Bisection Method Error")
53 plt.xlabel("n iterations")
54 plt.ylabel("true error")
55 plt.plot(root[1], root[2], 'ro')
56 plt.grid(True)
57 plt.show()
```

3.2 Output



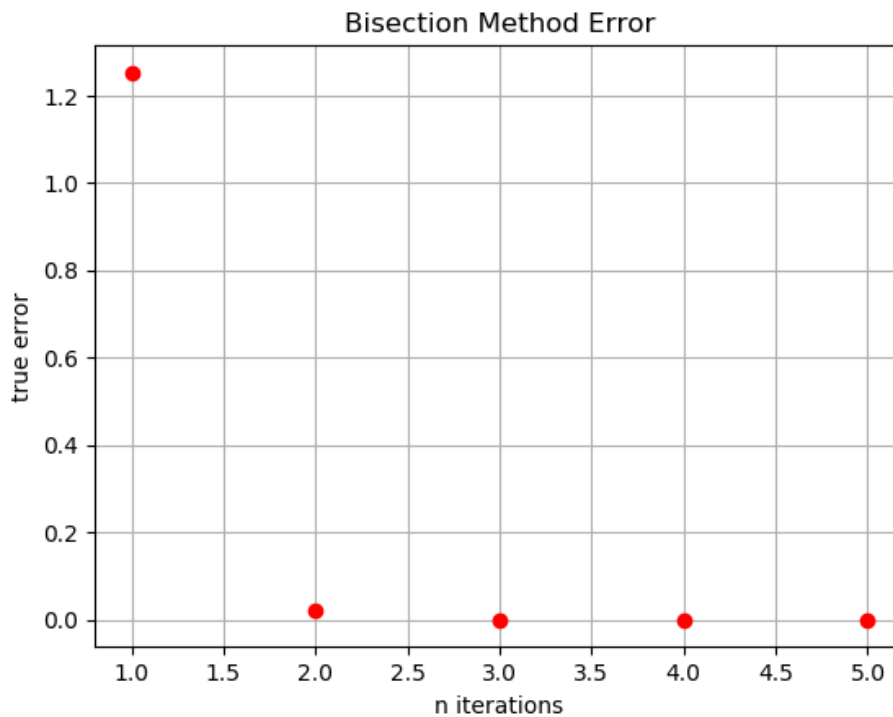
3.3 Accuracy and Order of Convergence

The last approximation for the root was within 1.224921×10^{-8} of the true value. The approximated value for the root of the function is 1.247856. The order of convergence is 2.006120.

Anaconda Prompt

```
(base) C:\Users\saeth>cd C:\Users\saeth\Documents\Courses\2019\Fall\MATH 5610
(base) C:\Users\saeth\Documents\Courses\2019\Fall\MATH 5610>python Newtons_Method.py
iteration: 1 - error: 2.499999999733191
iteration: 2 - error: 1.229607740945944
iteration: 3 - error: 0.02272053813909114
iteration: 4 - error: 0.00018466816241025086
iteration: 5 - error: 1.2249209202508382e-08
error: 1.2249209202508382e-08
approximate root: 1.247856401593393
order of convergence: 2.0061201921518266
```

The distance of each approximation to the true value with each iteration n :



4 Secant Method

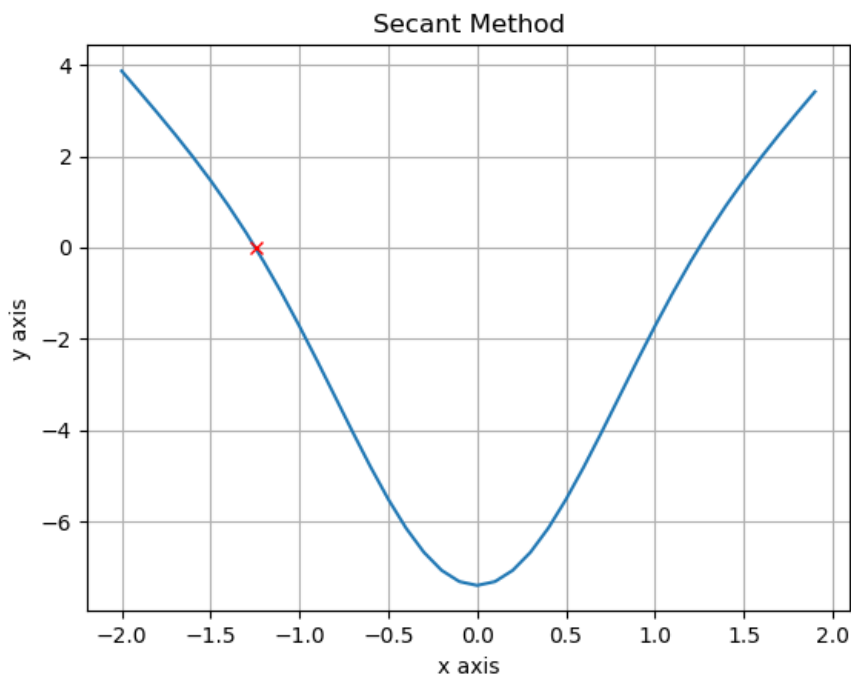
The Secant Method was adapted from Newton's Method by approximating $f'(x)$ with $\frac{f(x_{n+1}) - f(x_n)}{x_{n+1} - x_n}$.

4.1 Code

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 my_function = lambda x: (x)**2 - np.exp(2 - x**2)
5 true_root = -1.24786
6
7 def find_root_secant(x_0, x_1, epsilon):
8     error = 1
9     n = 0
10    n_array = []
11    true_error_array = []
12    while (error > epsilon):
13        x_n = x_0 - (my_function(x_0)*(x_1 - x_0))/
14            (my_function(x_1) - my_function(x_0))
15        error = abs(x_0 - x_n)
16        x_0 = x_1
17        x_1 = x_n
18        n = n + 1
19        n_array.append(n)
20        true_error = abs(true_root - x_0)
21        true_error_array.append(true_error)
22        print('iteration:', n, " - error:", error)
23        continue
24    print('error:', error)
25    print('approximate root:', x_n)
26    converge_rate = (np.log(abs((true_error_array[n-1] -
27        true_error_array[n-2])/
28        (true_error_array[n-2] -
29        true_error_array[n-3]))))/
30        (np.log(abs((true_error_array[n-2] -
31        true_error_array[n-3])/
32        (true_error_array[n-3] -
33        true_error_array[n-4])))))
34    print("order of convergence:", converge_rate)
35    return x_n, n_array, true_error_array
36
37
38 # Plot the function and the root
```

```
39 root = find_root_secant(-1.2,1,0.000001)
40 root_0 = 0
41 x = np.arange(-2,2,0.1)
42 y = (x)**2-np.exp(2-x**2)
43 plt.title("Secant Method")
44 plt.xlabel("x axis")
45 plt.ylabel("y axis")
46 plt.plot(x, y)
47 plt.plot(root[0], root_0, 'rx')
48 plt.grid(True)
49 plt.show()
50
51 # Plot of error vs iterations
52 plt.title("Secant Method Error")
53 plt.xlabel("n iterations")
54 plt.ylabel("true error")
55 plt.plot(root[1], root[2], 'ro')
56 plt.grid(True)
57 plt.show()
```

4.2 Output

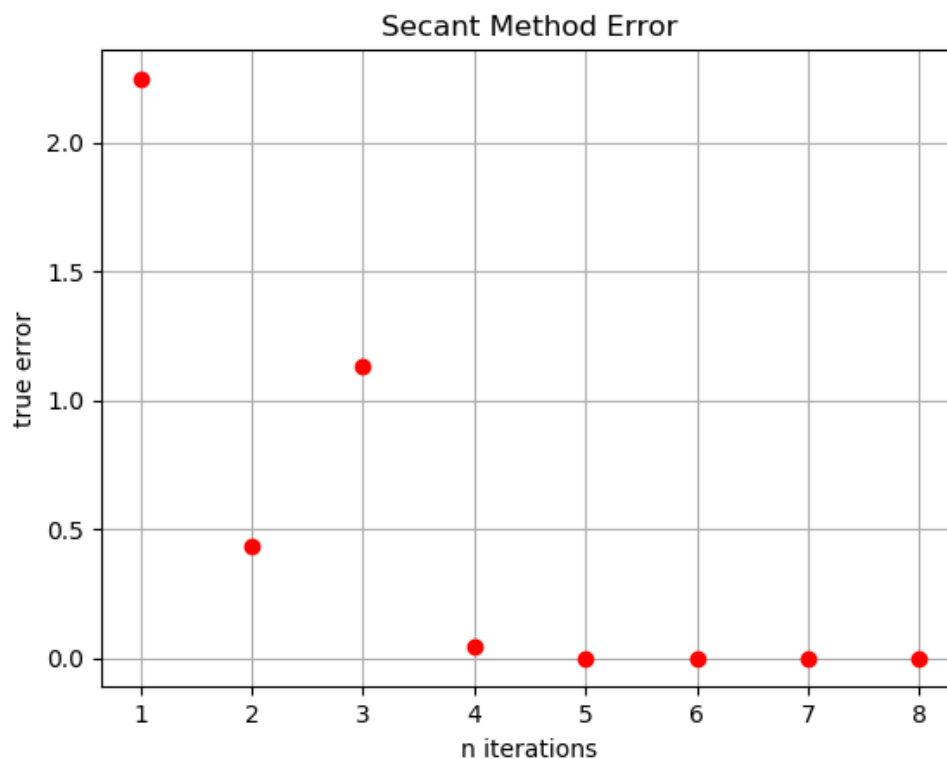


4.3 Accuracy and Order of Convergence

The last approximation for the root was within 8.302587×10^{-9} of the true value. The approximated value for the root of the function is -1.247856 . The order of convergence is 1.930753 .

```
Anaconda Prompt
(base) C:\Users\saeth\Documents\Courses\2019\Fall\MATH 5610>python Secant_Method.py
iteration: 1 - error: 0.48556050814427665
iteration: 2 - error: 1.117820722186887
iteration: 3 - error: 0.39016790166402604
iteration: 4 - error: 1.1311915574356324
iteration: 5 - error: 0.04755619626864749
iteration: 6 - error: 0.0011558697265390272
iteration: 7 - error: 1.999138184949345e-05
iteration: 8 - error: 8.302587284347851e-09
error: 8.302587284347851e-09
approximate root: -1.247856401593393
order of convergence: 1.9307527410602716
```

The distance of each approximation to the true value with each iteration n :



5 Fixed Point Method

The Fixed Point Method approximates the value at which the function $g(x) = (x + 10)^{\frac{1}{4}}$ intersects $f(x) = x$.

5.1 Code

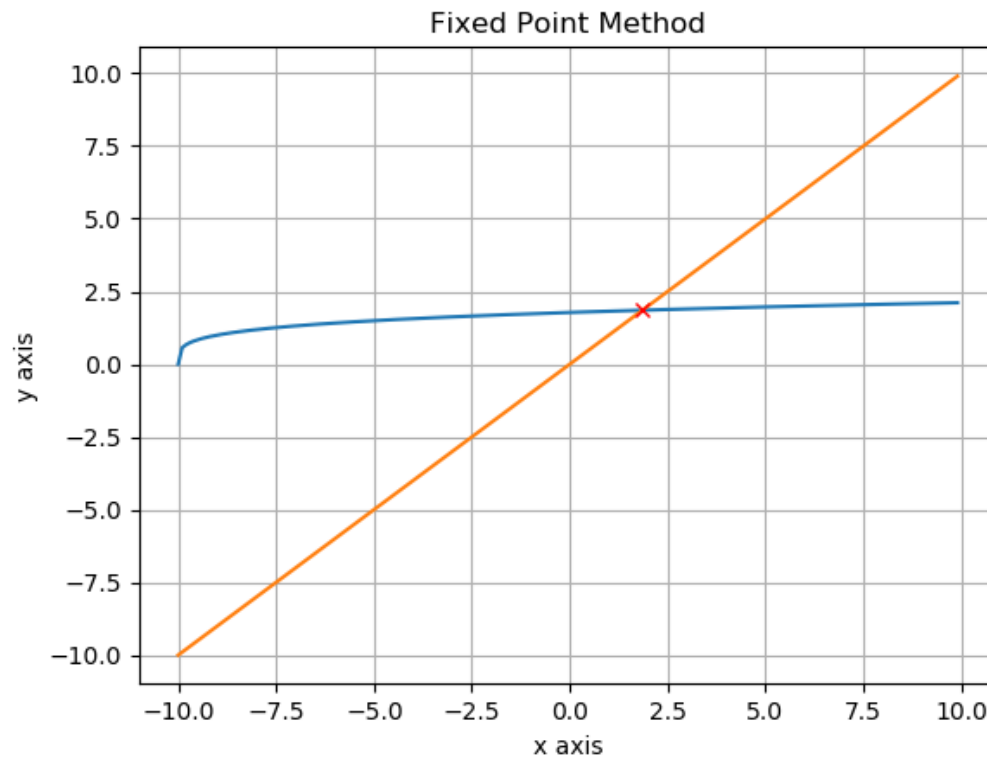
```

1  import numpy as np
2  from matplotlib import pyplot as plt
3
4  my_function = lambda x: (x + 10)**(1/4)
5  true_val = 1.855584528640937863760250564887986956416868
6              9613121880283886119879295797258305894223388
7              7913894052622171808602448788374623181292369
8              9694373052175808992515659707499952288816271
9              1608283807536494470806341316485129583883331
10             3556622920833781356730295328749934909901069
11             0823009
12
13  def fixed_point(x_0, epsilon):
14      error = 1
15      n = 0
16      n_array = []
17      true_error_array = []
18      while (error > epsilon):
19          x_n = my_function(x_0)
20          error = abs(x_0 - x_n)
21          x_0 = x_n
22          n = n + 1
23          n_array.append(n)
24          true_error = abs(true_val - x_0)
25          true_error_array.append(true_error)
26          print('iteration:', n, " - error:", error)
27          continue
28      print('error:', error)
29      print('approximate fixed point:', x_n)
30      converge_rate = (np.log(abs((true_error_array[n-1] -
31                                true_error_array[n-2]))/
32                      (true_error_array[n-2] -
33                      true_error_array[n-3])))/
34                      (np.log(abs((true_error_array[n-2] -
35                                true_error_array[n-3]))/
36                      (true_error_array[n-3] -
37                      true_error_array[n-4]))))

```

```
38     print("order of convergence:", converge_rate)
39     return x_n, n_array, true_error_array
40
41
42 # plot function and y = x
43 f_point = fixed_point(1,0.000001)
44 x = np.arange(-10,10,0.1)
45 y = (x + 10)**(1/4)
46 x_line = np.arange(-10,10,0.1)
47 y_line = x
48 plt.title("Fixed Point Method")
49 plt.xlabel("x axis")
50 plt.ylabel("y axis")
51 plt.plot(x, y)
52 plt.plot(x_line, y_line)
53 plt.plot(f_point[0], f_point[0], 'rx')
54 plt.grid(True)
55 plt.show()
56
57 # Plot of error vs iterations
58 plt.title("Fixed Point Method Error")
59 plt.xlabel("n iterations")
60 plt.ylabel("true error")
61 plt.plot(f_point[1], f_point[2], 'ro')
62 plt.grid(True)
63 plt.show()
```

5.2 Output



5.3 Accuracy and Order of Convergence

The last approximation for the fixed point was within 7.762710×10^{-8} of the true value. The approximated value for the fixed point is 1.855585. The order of convergence is 1.000000.

Anaconda Prompt

```
(base) C:\Users\saeth\Documents\Courses\2019\Fall\MATH 5610>python Fixed_Point_Method.py
iteration: 1 - error: 0.8211602868378718
iteration: 2 - error: 0.0330757894829774
iteration: 3 - error: 0.001295686596320511
iteration: 4 - error: 5.070105492266386e-05
iteration: 5 - error: 1.983880593137144e-06
iteration: 6 - error: 7.762709586245364e-08
error: 7.762709586245364e-08
approximate fixed point: 1.8555845254797814
order of convergence: 1.0000005140567554

(base) C:\Users\saeth\Documents\Courses\2019\Fall\MATH 5610>
```