WRITEUP - ASGN2

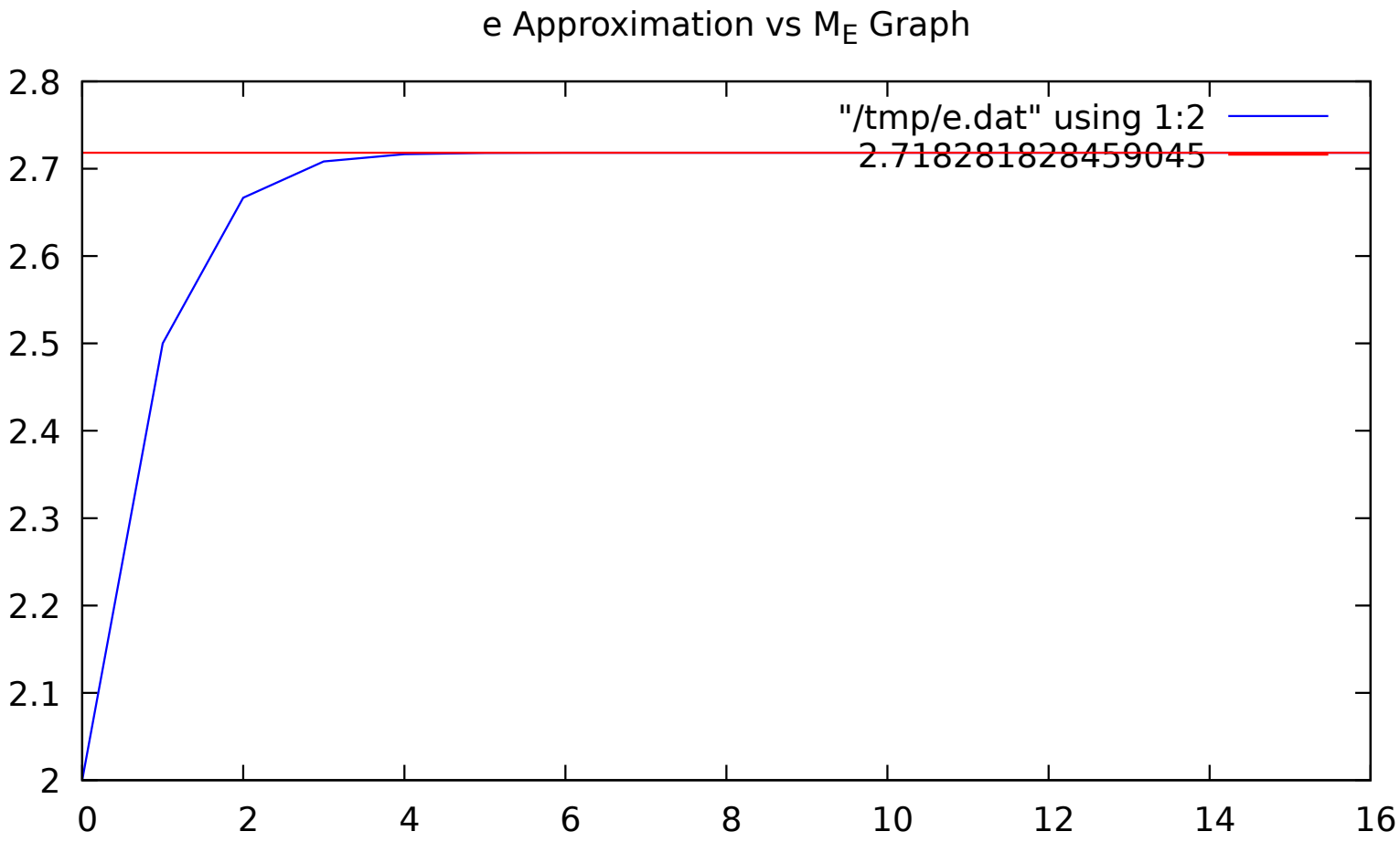
Viveka Agrawal

CSE13S, Winter 2023

Brief Summary of Assignment:

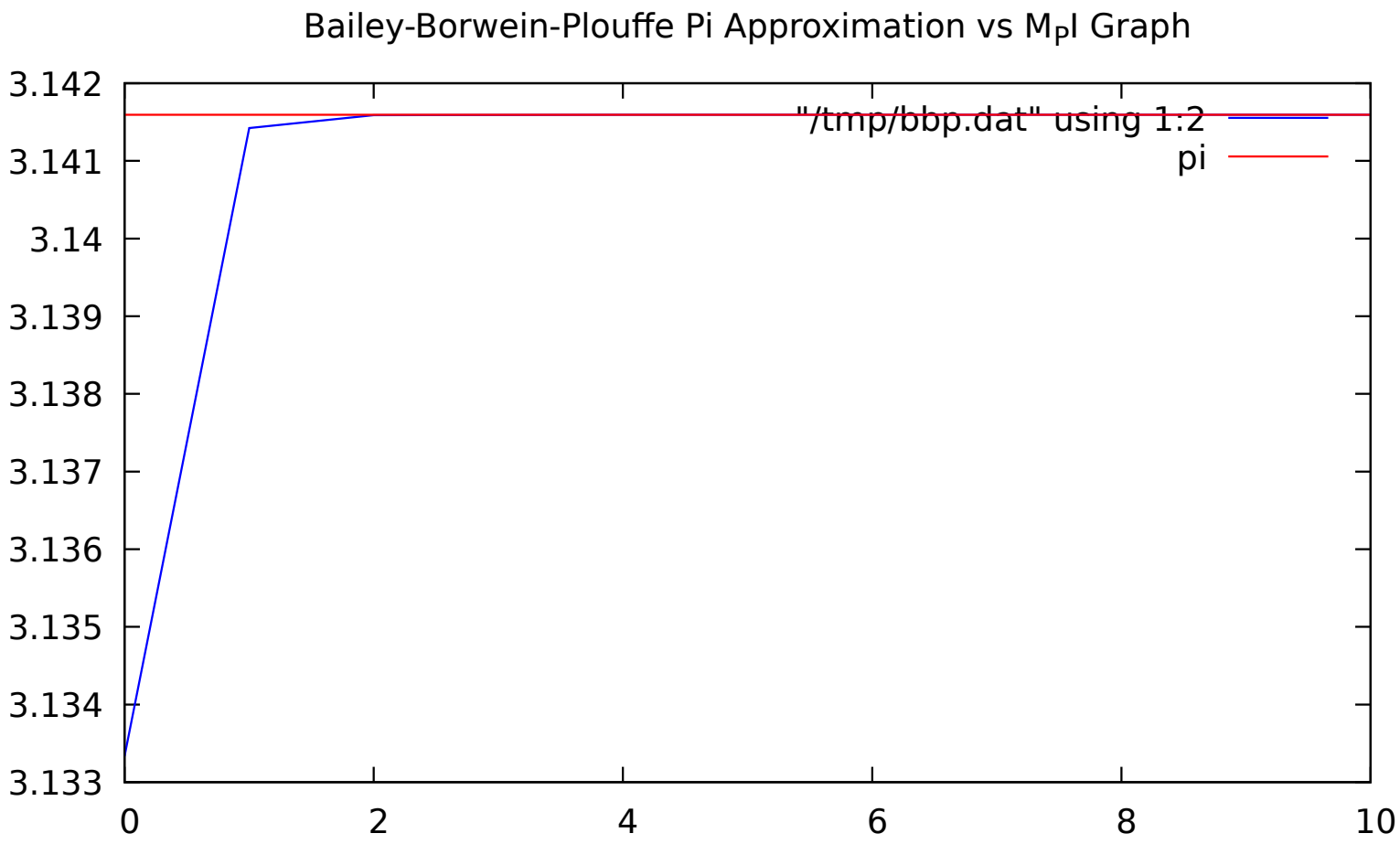
In this assignment, I learned how to code different math equations in C without using the math library. I coded the e approximation test, the Bailey-Borwein-Plouffe pi approximation test, the Madhava pi approximation test, the Euler sequence pi approximation test, the Viète pi approximation test, and the Newton-Raphson square root approximation tests. In my graphs, the blue line represents my approximation of the formulas and the red line represents the actual values.

Here is my gnuplot graph of the e approximation test VS the Math Library e value Graph pdf (next page):


e Approximation vs M_E Graph
"/tmp/e.dat" using 1:2
2.718281828459045
2.8
2.7
2.6
2.5
2.4
2.3
2.2
2.1
2
0
2
4
6
8
10
12
14
16

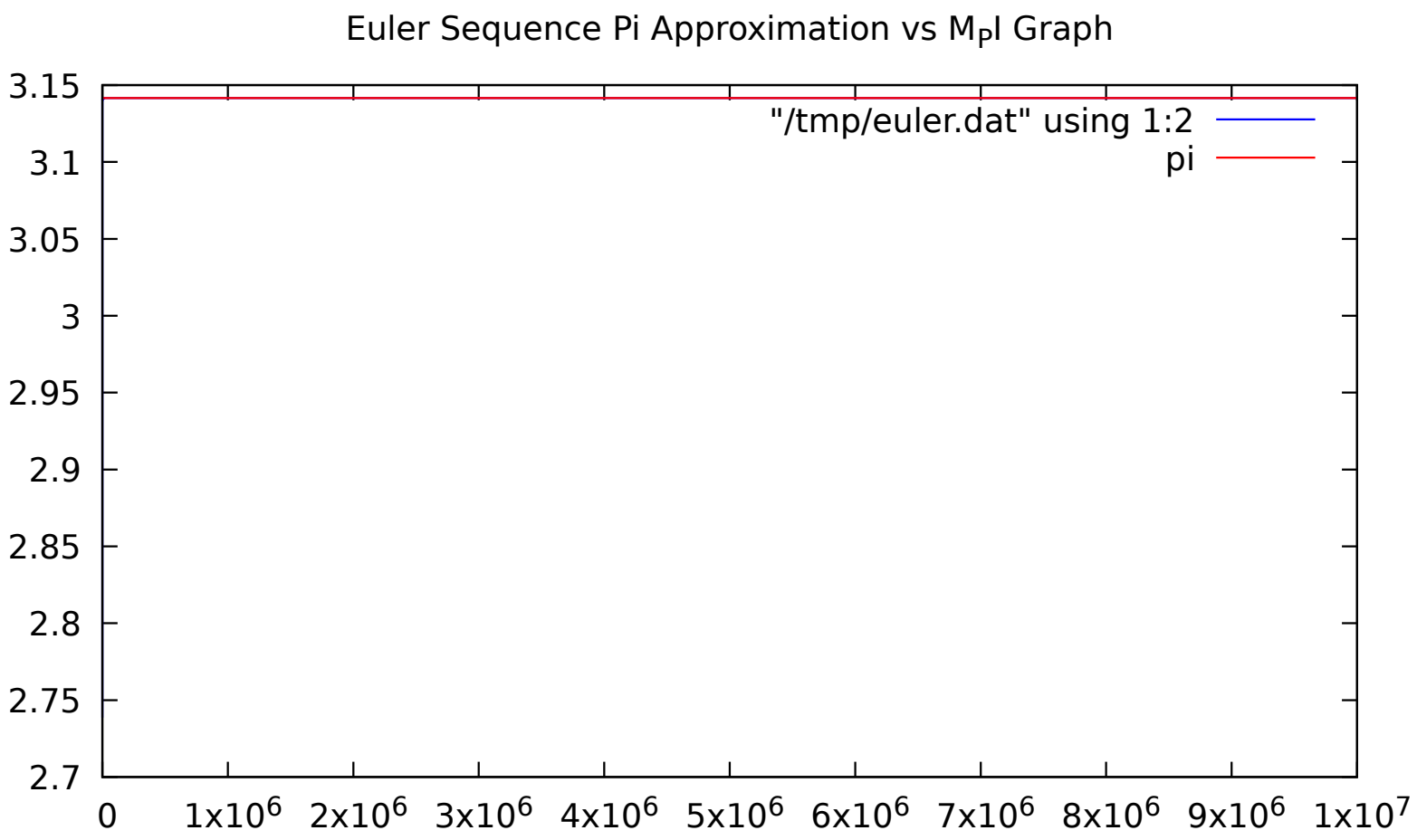
My e approximation function takes around 4 iterations to get very close to the actual e value. It starts at one because the first term is 1/0! and then the series slowly increases. The difference is so large at first because initially, the first calculated value of e is 1 based on my code, but as the while loop keeps running, the approximation gets closer and closer to the actual value of e. My code cannot be able to get the actual or close to the actual value of e at first since each term in the formula is calculated one at a time and then added to the previous value, so when the denominator in my code gets larger and larger, the approximation of e gets closer and closer to the actual value of e. The difference between my approximation of e and the actual value of e after 18 terms is 0.000000000000000, and this is shown in the graph.

Here is my gnuplot graph of the Bailey-Borwein-Plouffe pi Approximation Test VS the Math Library pi value Graph pdf (next page):


Bailey-Borwein-Plouffe Pi Approximation vs MPI Graph
"/tmp/bbp.dat" using 1:2
pi
3.142
3.141
3.14
3.139
3.138
3.137
3.136
3.135
3.134
3.133
0
2
4
6
8
10

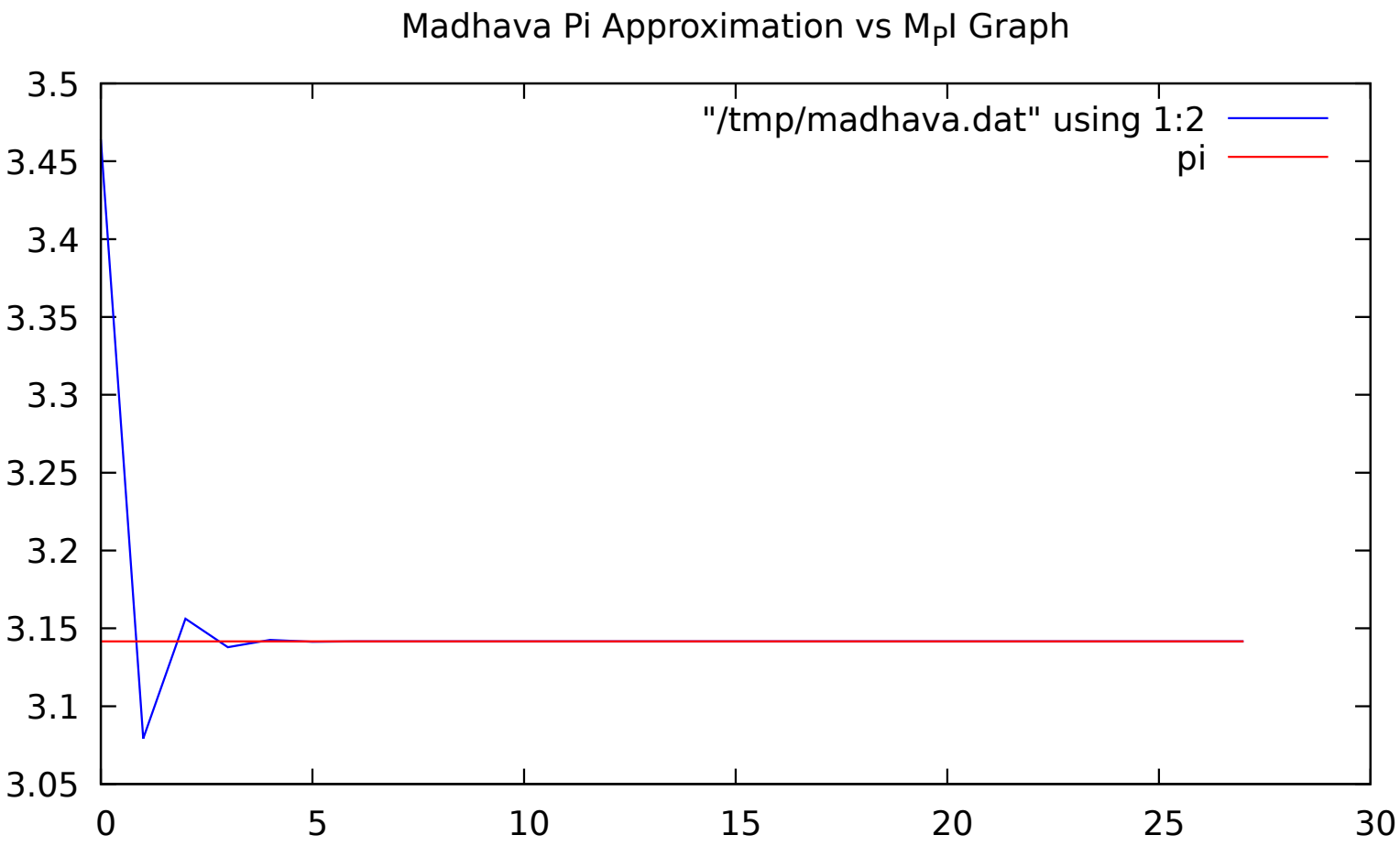
My bbp approximation takes around 2 iterations to get very close to the actual value of pi. This is because the bbp formula is already very close to pi, so it doesn't take many iterations for my approximation to become very close to pi. The difference between my approximation of pi from bbp and the actual value of pi is 0.000000000000000, and this is shown in the graph.

Here is my gnuplot graph of the Euler Sequence pi Approximation Test VS the Math Library pi value Graph pdf (next page):


Euler Sequence Pi Approximation vs MpI Graph
"/tmp/euler.dat" using 1:2
pi
3.15
3.1
3.05
3
2.95
2.9
2.85
2.8
2.75
2.7
0
1x10^6
2x10^6
3x10^6
4x10^6
5x10^6
6x10^6
7x10^6
8x10^6
9x10^6
1x10^7

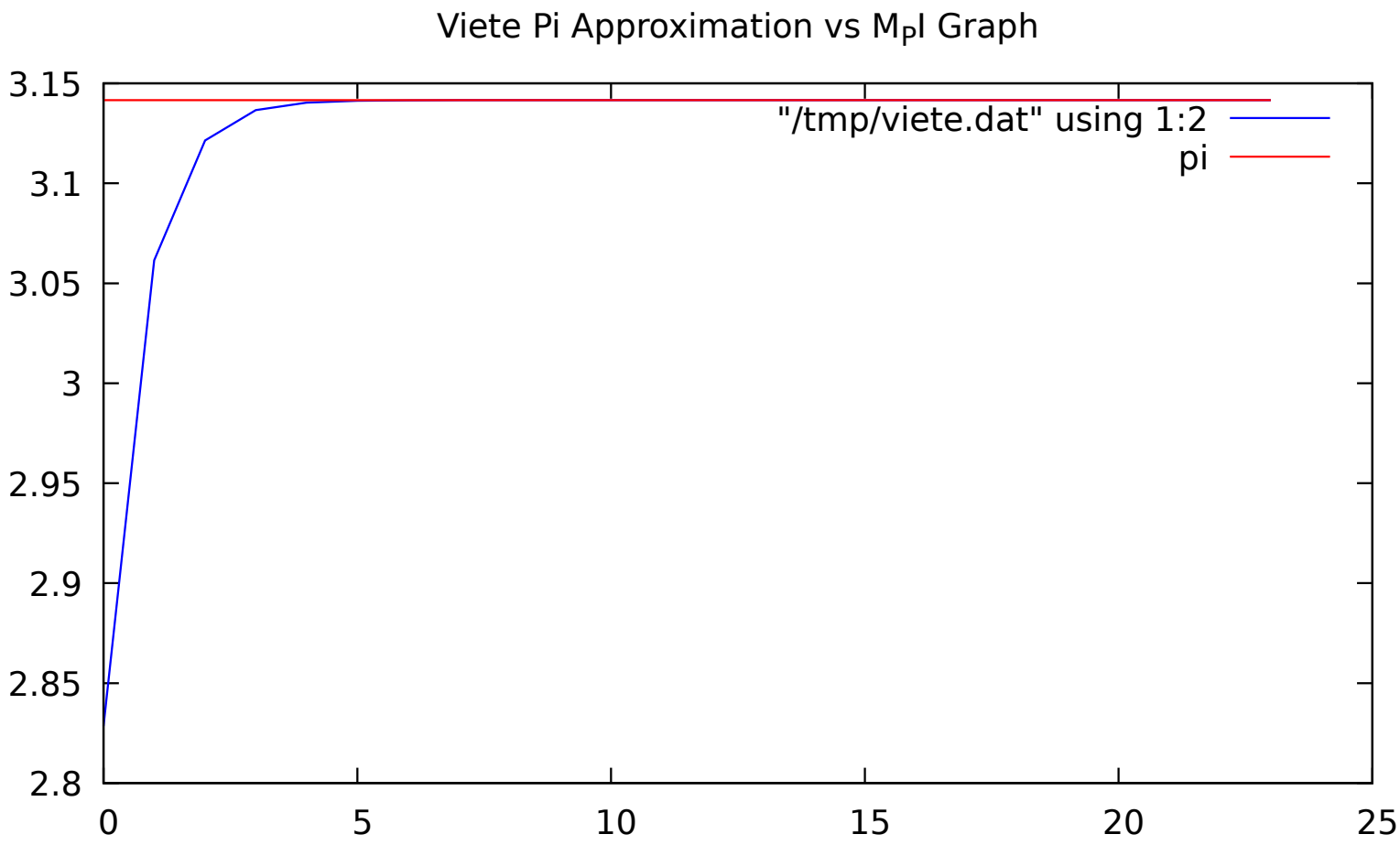
My euler approximation first starts at 0, but after about a few 1000 iterations, it comes extremely close to pi. It is not shown on the graph, but based on the fact that it takes 10000000 terms, I can infer that the number of iterations it takes for my approximation to get close to pi is 1000 or more. This is because the series starts at 0, then slowly increments by k to the power of 2 in the denominator of the function. My code cannot be able to get the actual or close to the actual value of pi at first since each term in the formula is calculated one at a time and then added to the previous value, so when the denominator in my code gets larger and larger, the approximation of pi gets closer and closer to the actual value of pi. The difference between my approximation of pi from euler and the actual value of pi is 0.000000095493881. There is no blue line shown for my approximation since the value of my approximation and pi are extremely close.

Here is my gnuplot graph of the Madhava Sequence pi Approximation Test VS the Math Library pi value Graph pdf (next page):


Madhava Pi Approximation vs MPI Graph
"/tmp/madhava.dat" using 1:2
pi
3.5
3.45
3.4
3.35
3.3
3.25
3.2
3.15
3.1
3.05
0
5
10
15
20
25
30

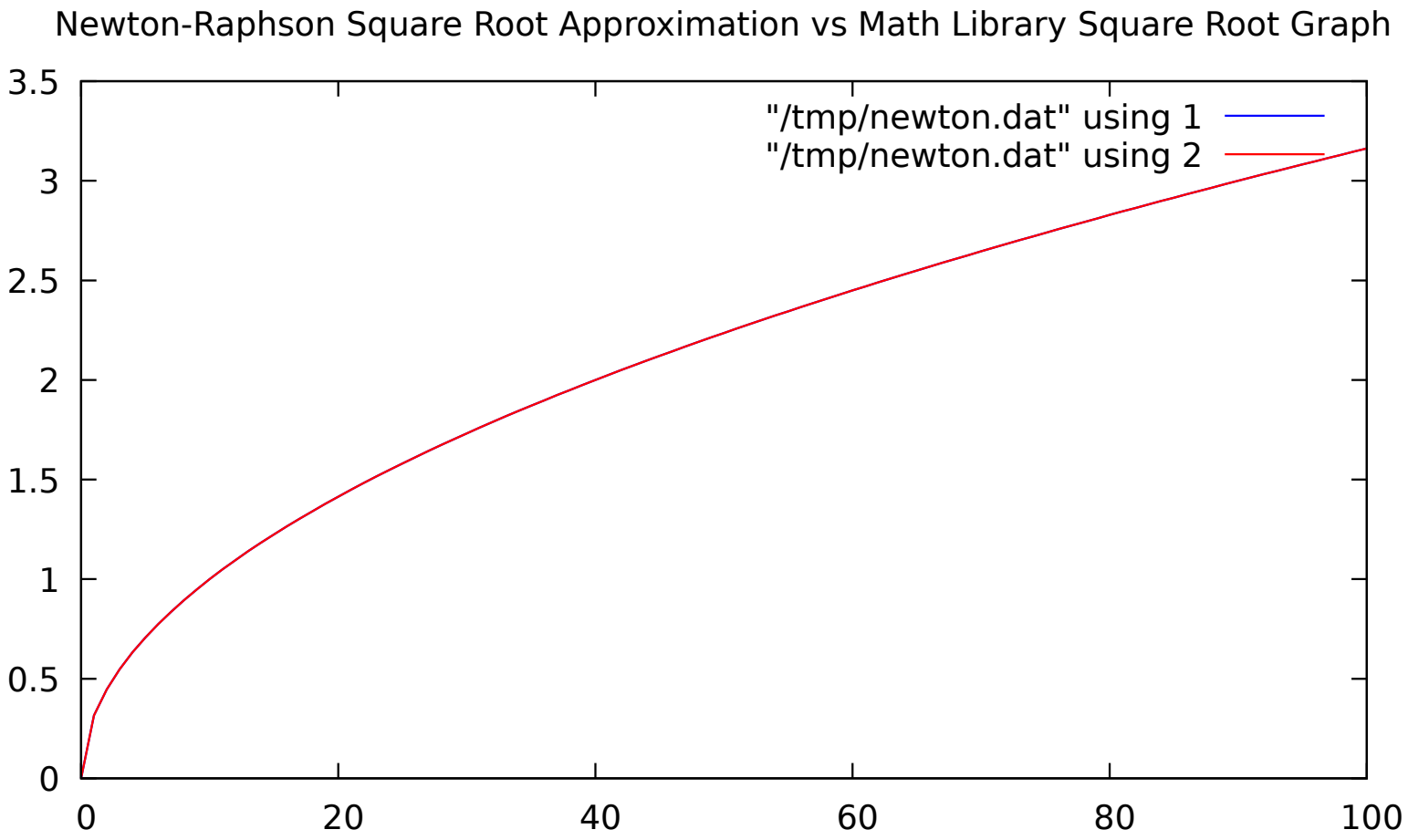
My madhava approximation at first is not close to pi at all. The function initially oscillates around pi (which is shown in the graph), but then the difference between the functions is very minimal around the fourth iteration. This is because the series initially begins at 0, where the first term is unaffected by negative 3 to the power of negative k. The second term, however, which is 1 is affected, therefore creating that oscillating look. My code cannot be able to get the actual or close to the actual value of pi at first since each term in the formula is calculated one at a time and then added to the previous value, so when the denominator in my code gets larger and larger, the approximation of pi gets closer and closer to the actual value of pi. The difference between my approximation of pi from madhava and the actual value of pi is 0.000000000000002.

Here is my gnuplot graph of the Viete pi Approximation Test VS the Math Library pi value Graph pdf (next page):


Viete Pi Approximation vs MPI Graph
"/tmp/viete.dat" using 1:2
pi
3.15
3.1
3.05
3
2.95
2.9
2.85
2.8
0
5
10
15
20
25

My viete approximation at first is not close to pi at all. The function rapidly increases towards pi, so the difference between the functions is very minimal around the fourth iteration. This is because the formula uses multiplication which quickly allows the function to approximate very close to pi. My code cannot be able to get the actual or close to the actual value of pi at first since each term in the formula is calculated one at a time and then multiplied by the previous value, so when the denominator in my code gets larger and larger, the approximation of pi gets closer and closer to the actual value of pi. The difference between my approximation of pi from viete and the actual value of pi is 0.000000000000004.

Here is my gnuplot graph of the Newton-Raphson Square Root Approximation Test VS the Math Library square root values Graph pdf (next page):


Newton-Raphson Square Root Approximation vs Math Library Square Root Graph
"/tmp/newton.dat" using 1
"/tmp/newton.dat" using 2
3.5
3
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100

My newton approximation is basically the same as the math library's square root function. This is shown on my graph which only has the red line, not even showing the blue line. This shows that my approximation of the Newton-Raphson Square Root approximation, for which Professor Long provided pseudocode for, is exactly the same as the actual square root function from the math library. The difference between my approximation of the square root function from newton and the actual value of the square root from the math library is first 0.000000000000007 then goes down to 0.000000000000000.