Julian Chan

Lab Project 4

CS 317: Algorithms Analysis

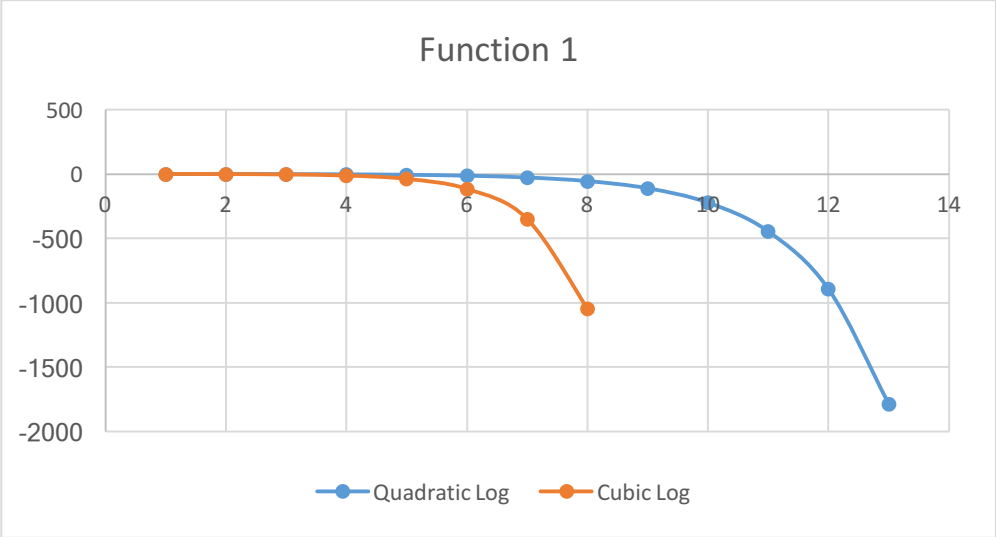
Miller

Lab Project 4: Newton's Method

After writing code for Newton's Algorithm with Quadratic, and then Cubic convergence. I found that the cubic convergence was "superior" in that it took less iterations to achieve an error tolerance of 10^{-1000}, as expected. I computed both algorithms through Java. There were no problems in terms of actually writing the code for Newton's Quadratic algorithm, however, there were a few problems regarding Newton's Cubic algorithm. Such as, how do I find $\sqrt{f'(x)^2 - 2f(x)f''(x)}$, since there is no square root function within the Big Decimal class. I eventually solved this issue by creating another Newton's Quadratic algorithm method within my Newton's Cubic algorithm. Every iteration for Newton's Cubic algorithm, I would do a separate set of iterations of Newton's Quadratic algorithm in a separate method in order to find the square root needed for the Newton's Cubic algorithm computation. Below are my results of each iteration and the error up until the error tolerance of 10^{-1000} was met. Expressed in terms of (n, $\log_{10}(E_n)$).

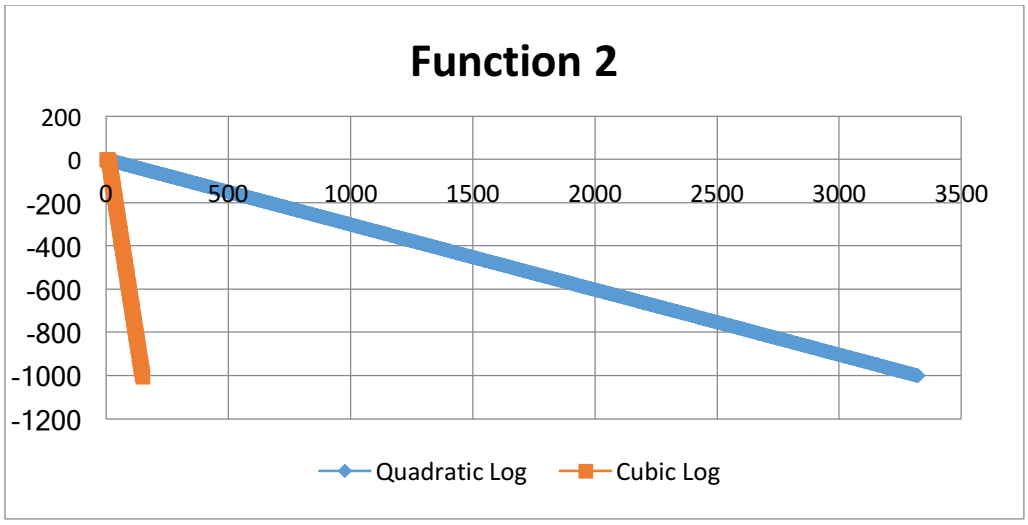
Function #1: $f(x) = x^3 - 8$

	Quadratic	Cubic
Root near:	2.0	2.0
Initial Guess:	3.0	3.0
Final Approximation:	2.00000	2.00000
# of Iterations to Achieve tol:	13 iterations	8 iterations



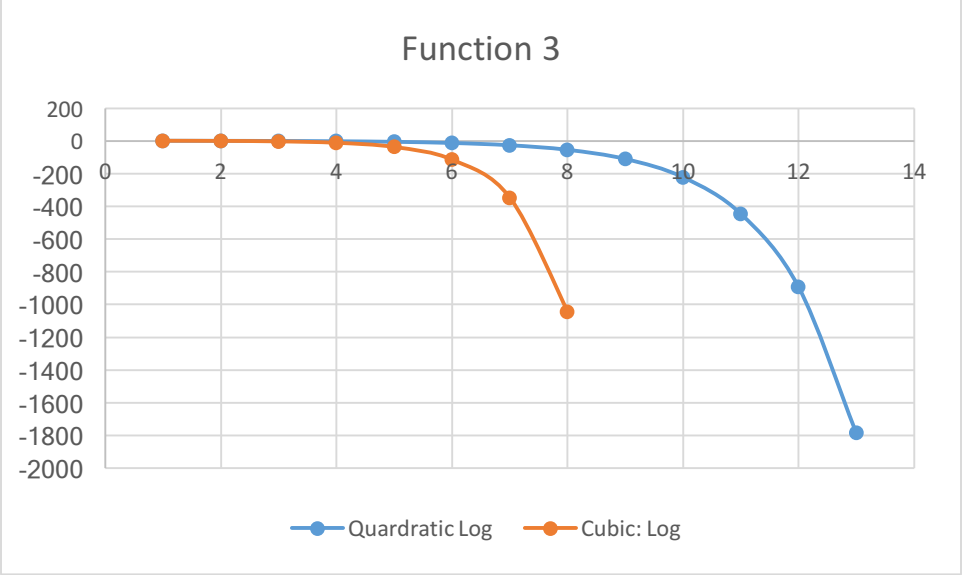
Function #2: $f(x) = x^3 - 2x^2 - 4x + 8$

	Quadratic	Cubic
Root near:	2.0	2.0
Initial Guess:	3.0	3.0
Final Approximation:	2.00000	1.99999
# of Iterations to Achieve tol:	3323 iterations	148 iterations



Function #3: f(x) = $-x^3 + 8$

	Quadratic	Cubic
Root near:	2.0	2.0
Initial Guess:	3.0	3.0
Final Approximation:	2.00000	2.00000
# of Iterations to Achieve tol:	13 iterations	8 iterations



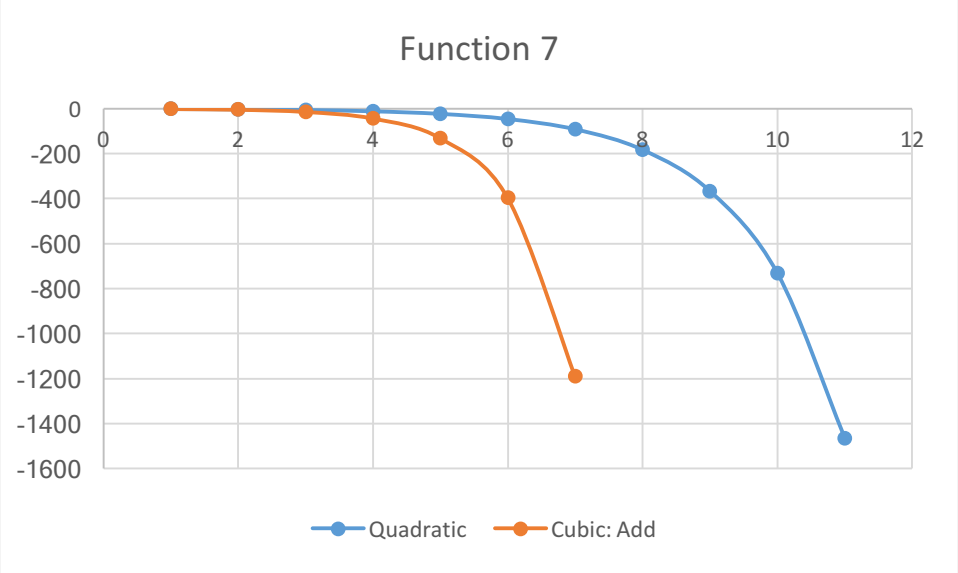
Function #6: $f(x) = -x^4 + 23$

	Quadratic	Cubic
Root near:	2.0	2.0
Initial Guess:	3.0	3.0
Final Approximation:	error	error
# of Iterations to Achieve tol:	∞	∞

Function #6 does not have a real root. The root includes an imaginary number. The algorithm will iterate forever and never converge to the root.

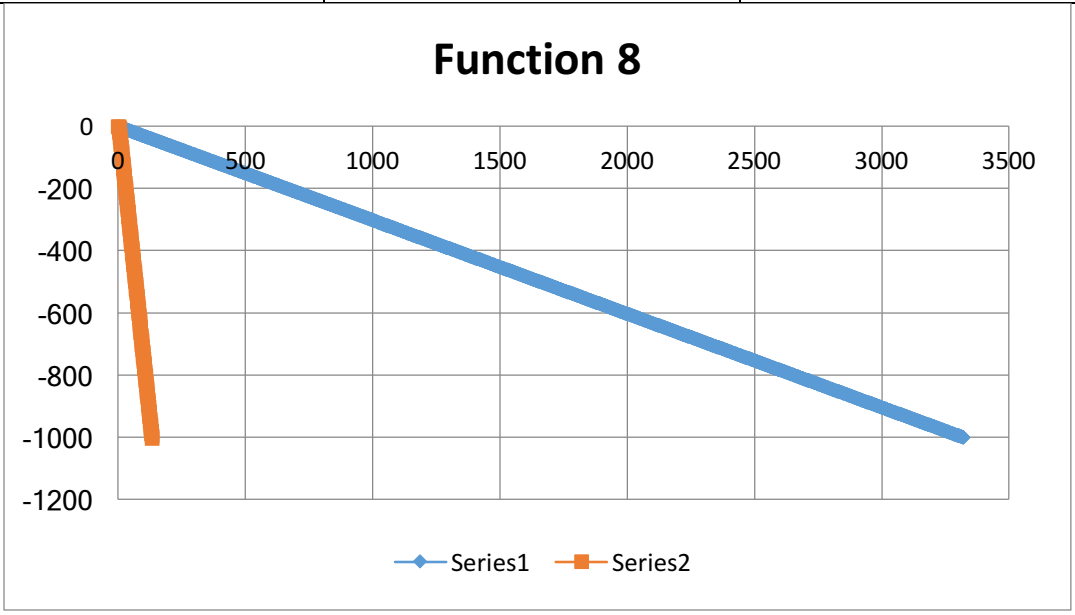
Function #7: $f(x) = x^5 - 26x^2$

	Quadratic	Cubic
Root near:	2.5	2.5
Initial Guess:	3.0	3.0
Final Approximation:	2.96249	2.96249
# of Iterations to Achieve tol:	11 iterations	7 iterations



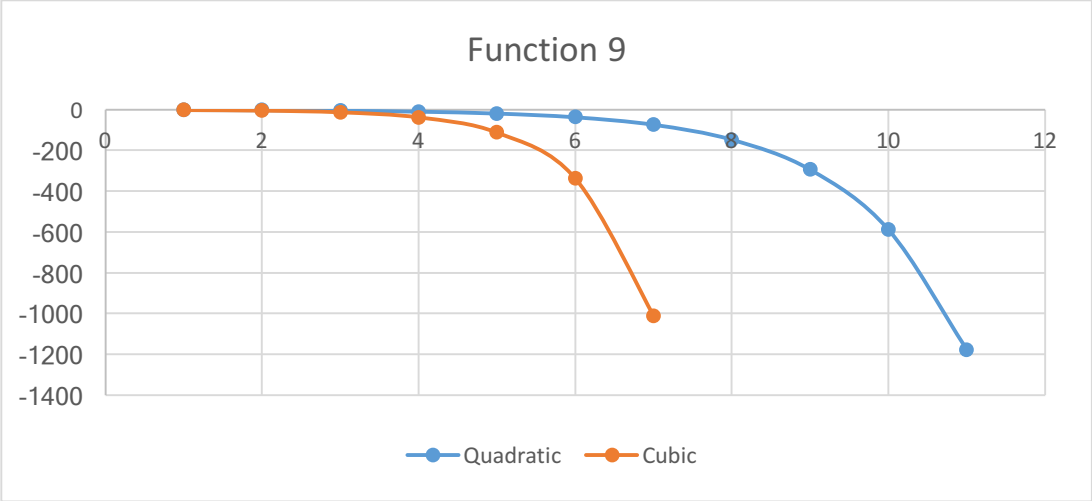
Function #8: $f(x) = x^5-26x^2$

	Quadratic	Cubic
Root near:	0	0
Initial Guess:	1	1
Final Approximation:	8.84596E-1001	3.15415E-1010
# of Iterations to Achieve tol:	3322 iterations	133 iterations



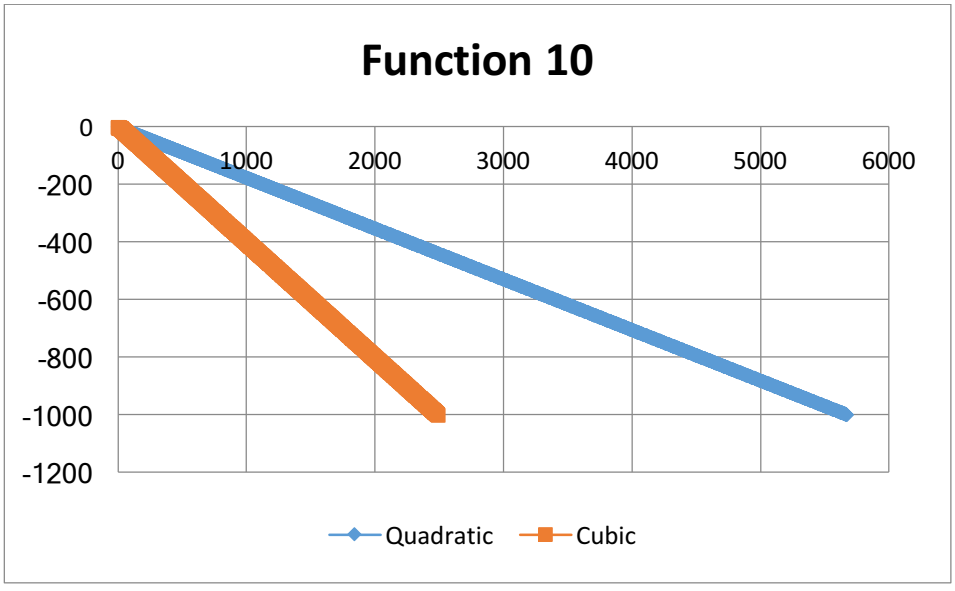
Function #9: $f(x) = x^5-26x^3$

	Quadratic	Cubic
Root near:	Near 5	Near 5
Initial Guess:	5	5
Final Approximation:	5.09902	5.09902
# of Iterations to Achieve tol:	11 iterations	7 iterations



Function #10: $f(x) = x^5-26x^3$

	Quadratic	Cubic
Root near:	0	0
Initial Guess:	0.1	0.1
Final Approximation:	1.62285E-100	-1.10883E-8
# of Iterations to Achieve tol:	5672 iterations	2490 iterations



Final Analysis

Factorization/multiplicity

1. $f(x) = x^3 - 8$ → $(x-2)(x^2+2x+4)$ Root near 2.0 – multiplicity = 1
2. $f(x) = x^3 - 2x^2 - 4x + 8$ → $(x+2)(x-2)^2$ Root near 2.0 – multiplicity = 2
3. $f(x) = -x^3 + 8$ → $(-x+2)(x^2+2x+4)$ Root near 2.0 – multiplicity = 1
4. –
5. –
6. $f(x) = x^4 + 23$ → *Doesn't factor Root near 2.0 – never converges
7. $f(x) = x^5 - 26x^2$ → $x^2(x^3-26)$ Root near 2.5 – multiplicity = 1
8. $f(x) = x^5 - 26x^2$ → $x^2(x^3-26)$ Root near 0 – multiplicity = 2
9. $f(x) = x^5 - 26x^3$ → $x^3(x^2-26)$ Root near 2.5 – multiplicity = 1
10. $f(x) = x^5 - 26x^3$ → $x^3(x^2-26)$ Root near 0 – multiplicity = 3

As observed by my data points and graphs, the order of the functions that took the least amount of iterations to converge towards the specified root at an error tolerance of 10^{-1000}, to the functions that took the most amount of iterations is as follows.

Least iterations

7 (Quadratic = 11 Cubic = 7)

9 (Quadratic = 11 Cubic = 7)

1 (Quadratic = 13 Cubic = 8)

3 (Quadratic = 13 Cubic = 8)

8 (Quadratic = 3322 Cubic = 133)

2 (Quadratic = 3323 Cubic = 148)

10 (Quadratic = 5672 Cubic = 2490)

6 (Quadratic = ∞ Cubic = ∞)

Most iterations

The order of these functions from least to most iterations is as expected and can be explained by each function's specified root's multiplicity. For functions 7 and 9 which took the least amount of iterations to converge towards their specified root of 2.0, both had a multiplicity of 1 for root 2.0 in their respective functions. Functions 1 and 3 took a few more iterations to converge towards their specified roots of 2.0 and 2.5, but were still quite close to the convergence rates of functions 7 and 9, this is due to their multiplicities of 1 as well for their roots of 2.0 and 2.5 within their functions. Functions 8 and 2 both had multiplicities of 2 for their specified roots of 0 and 2.0, which is why they took longer to converge than the previous four functions. Function 10 took the most amount of iterations to converge out of all of the functions because the 0 root in its function had a multiplicity of 3, making function 10's convergence rate the slowest out of all of them. And finally, function 6 had an infinite amount of iterations for both Quadratic Newton and Cubic Newton, simply because it never converges, it has no real root.

In regards to choosing a good "guess" for each algorithm, as long as the guess is close enough to the specified root, and assuming that the specified root does exist, both algorithms will converge towards the root. In function #6's case, although my guess was close to the specified root of 2.0, it never converges because the root does not exist. Moreover, as long as the initial guess is within the range of $-1 < g'(x) < 1$, g'(x) being the derivative of the Newton's

algorithm in question, the algorithm should converge towards the root. However, Newton's Cubic algorithm is more sensitive in terms of choosing a "guess," and thus requires more care. The farther away from the actual root the guess is, the more iterations Newton's Cubic algorithm will conduct in order to achieve the required tolerance. Whereas, Newton's Quadratic algorithm isn't quite as sensitive, choosing a farther guess from the actual root doesn't increase the amount of iterations it takes to achieve the error tolerance as much as Newton's Cubic algorithm. For example, for function 1, $f(x) = x^3 - 8$, with root 2.0, a guess of 3 causes Newton's Quadratic algorithm to iterate 13 times, and Newton's Cubic algorithm to iterate 8 times, before reaching error tolerance of 10^{-1000}. However, if we choose a guess of 5, Newton's Quadratic algorithm only takes one more iteration to achieve the error tolerance with 14, but Newton's Cubic algorithm jumps from 8 iterations to 13 iterations, thus highlighting the sensitivity of Newton's Cubic algorithm in choosing a good "guess."

For Newton's Cubic algorithm I used a plus sign if $f'(x) * f''(x) > 0$, and I used a minus sign if the $f'(x) * f''(x) < 0$, this ensures convergence.

I would go about deriving a Newton's Quartic algorithm by deriving the third derivative of f(x), $f'''(x)$ with the limit definition. Once I have that, I would solve for x_n. This is resembling how we derived Newton's Quadratic Algorithm in that the first derivative of f(x) can be represented as

$$f'(x) = \frac{f(\bar{x}) - f(x)}{x_n - \bar{x}}$$

with $f(\bar{x}) = 0$. This gives us a reduction of $x_n = \bar{x} - \frac{f(x)}{f'(x)}$ or Newton's Quadratic Algorithm. Theoretically, if we reduce the third derivative of f(x) for x_n, we should get Newton's Quartic Algorithm.

The advantages in creating a hybrid algorithm which starts off using the Quadratic Newton's algorithm for a few steps, then switching to the Cubic Newton's algorithm is that it minimizes the number of iterations it takes to achieve error tolerance of 10^{-1000}. This is a good idea because with a very poor guess (far from the actual root), Newton's Cubic algorithm takes much more iterations to converge to the root and hit tolerance. However, Newton's Cubic algorithm is very good at minimizing iterations to converge and achieving error tolerance of 10^{-1000} once close to the actual root. So by using Newton's Quadratic first and then switching to Newton's Cubic algorithm minimizes the amount of iterations it takes to converge towards a root while simultaneously achieving the desired error tolerance as opposed to just running one of the algorithms.