Steve Harman
CIS541 HW#2

Results & Analysis of various root-finding methods

The goal of this lab was to show how three different root-finding methods performed when attempting to find the roots of three independent functions.

I began by implementing the bisection method, Newton's method, and the secant method in C++ on the standard Solaris environment. You can see the attached source code (hw2.cpp) for implementation details. After implementation was completed, I ran the executable (called "times") for each of the three different functions. I then compiled the results in a table (below). My analysis of results:

Function 1: cos(x+1) – ln((x/5)+1).

The bisection method found three roots within the range [0,10] for this function. After graphing the function of a TI-86 graphing calculator, these do indeed appear to be all of the roots that for this function (within the given range). It took an average of about 37 iterations to find each of these three roots, and to total of 710,800 nano-seconds for the bisection method to run on this range.

Newton's method also found three distinctive roots within the range [0,10]. The roots were the same to about the eleventh decimal place, so there was some discrepancy between the two methods. It took an average of only 5.111 iterations and a total of 357,100 nano-seconds for Newton's method to find these three distinctive roots. When starting with x=9 & x=10, Newton's method fell out of range (that is, it failed to converge fast enough), so I reported no root for those starting points.

The secant method was also able to find all three roots within the range [0,10]. This method found the three roots with an average of just 4.714 iterations, and so had the fastest convergence of the three. However, the total running time for this method was somewhere between the other two at 536,100 nano-seconds. Also for the intervals [2,3], [8,9], & [9,10] this method failed to converge on a root for unknown reason.

So, all three methods found the three roots for this function. However, I would say that Newton's method and the secant method performed better as they found the roots much more quickly than the bisection method (that is, they converged on the roots faster). If looking purely at the overall runtimes, then I would say that Newton's method performed the best for this function.

Function 2: tan(0.5 + (x/3)).

For this function, the bisection method found just one root in the range [0,10]. After looking at a graph of this function, that does indeed appear to be the only root in the given range. The bisection method was able to converge on this root in just 36 iterations. It also had a total runtime of 453,900 nano-seconds.

Newton's method was unable to find any roots for this function. For each starting point 0 – 10, this method went out of range (that is, it shot off to infinity, or –infinity), and so was unable to find any roots. Newton's method ran in just 191,400 nano-seconds, the fastest of the three methods.

The sSecant method actually reported finding two different roots, however only one of them fell within the range [0,10]. According to a graph of the function, the two points that this method reported were in fact roots of the function. It took an average of just 10 iterations to find the two roots, and an average of just 4.571 iterations to converge

on the root within the given range. The overall runtime for this method was 832,500 nano-seconds, which was the slowest of all of the methods.

Overall, I would say that the bisection method and the secant method preformed the best as they were both able to report the only root within the range. I would say the secant method performed better because even though the bisection method had a faster overall runtime, but the secant method converged much more quickly.

Function 3: $1000/(x+1) + 3^x - 2x^3$.

For the third function the bisection method was unable to find any roots in the range [0,10]. After consulting a graph of the function, this is correct as the only root even near the range lies at about x=-5. This method had an overall runtime of 264,300 nano-seconds, which was a power of 10 faster than the other two functions.

Newton's method reported one root, but it was outside of the given range. This method found a root near x=-5 for several starting points, and went out of range just once. The other starting points resulted in too many iterations (I had set a max of 100). It took an average of 57.714 iterations to find this root, and a total runtime of 2,531,500 nano-seconds, this was the slowest of the three methods.

The secant method was also able to find a root, however it was out of the given range. Like Newton's method, the secant method reported a root near x=-5, however it only took an average of 36.8 iterations. The overall runtime for this method was 2,323,600 nano-seconds, which isn't too far behind Newton's method. For all of the intervals which didn't report a root, the program reported an unknown reason for exiting. My guess is that the x shot off to some big number, and resulted in a DIV by Zero. When I looked into it, x was being reported as a NaN (Not a Number), by the C++ code.

Overall I would say that the bisection method performed the best for this function. It correctly reported no roots within the given range, and it ran much faster than the other two methods.

Function 1 = cos(x+1) - ln((x/5)+1)

Bisection Method

Roots	Iterations	Interval
0.4791551118124	39	[0,1]
none		[1,2]
none		[2,3]
none		[3,4]
4.3948783983215	36	[4,5]
5.9526646769446	37	[5,6]
none		[6,7]
none		[7,8]
none		[8,9]
none		[9,10]
Running Time (nano-seconds)	710,800	

Newton's Method

Roots	Iterations	Interval
0.4791551118131	4	0
0.4791551118131	4	1
4.3948783983204	10	2
4.3948783983204	6	3
4.3948783983204	4	4
4.3948783983204	5	5
5.9526646769453	3	6
5.9526646769453	5	7
4.3948783983204	5	8
out of range		9
out of range		10
Running Time (nano-seconds)	357,100	

Secant Method

Roots	Iterations	Interval
0.4791551118131	3	[0,1]
0.4791551118131	5	[1,2]
no root, unknown reason		[2,3]
4.3948783983197	4	[3,4]
4.3948783989204	6	[4,5]
5.9526646769453	5	[5,6]
5.9526646769458	4	[6,7]
5.9526646769453	6	[7,8]
no root, unknown reason		[8,9]
no root, unknown reason		[9,10]
Running Time (nano-seconds)	536,100	

Function 2 = tan(0.5+(x/3))

Bisection Method

Roots	Iterations	Interval
none		[0,1]
none		[1,2]
none		[2,3]
none		[3,4]
none		[4,5]
none		[5,6]
none		[6,7]
7.9247779607686	36	[7,8]
none		[8,9]
none		[9,10]
Running Time (nano-seconds)	453,900	

Newton's Method

Roots	Iterations	Interval
out of range		0
out of range		1
out of range		2
out of range		3
out of range		4
out of range		5
out of range		6
out of range		7
out of range		8
out of range		9
out of range		10
Running Time (nano-seconds)	191,400	

Secant Method

Roots	Iterations	Interval
-1.4999999999999	4	[0,1]
-1.5000000000000	6	[1,2]
-1.5000000000000	8	[2,3]
7.9247779607684	8	[3,4]
7.9247779607694	6	[4,5]
7.9247779607694	5	[5,6]
7.9247779607694	4	[6,7]
7.9247779607695	2	[7,8]
7.9247779607694	3	[8,9]
7.9247779607694	4	[9,10]
Running Time (nano-seconds)	832,500	

Function 3 = 1000/(x+1) + 3^x - 2x^3

Bisection Method

Roots	Iterations	Interval
none		[0,1]
none		[1,2]
none		[2,3]
none		[3,4]
none		[4,5]
none		[5,6]
none		[6,7]
none		[7,8]
none		[8,9]
none		[9,10]
Running Time (nano-seconds)	264,300	

Newton's Method

Roots	Iterations	Interval
too many iterations		0
too many iterations		1
-4.9999806337919	44	2
out of range		3
-4.9999806337919	50	4
-4.9999806337920	63	5
too many iterations		6
-4.9999806337921	41	7
-4.9999806337921	47	8
-4.9999806337920	63	9
-4.9999806337919	96	10
Running Time (nano-seconds)	2,531,500	

Secant Method

Roots	Iterations	Interval
-4.9999806337916	16	[0,1]
no root, unknown reason		[1,2]
-4.9999806337916	29	[2,3]
-4.9999806337916	28	[3,4]
no root, unknown reason		[4,5]
no root, unknown reason		[5,6]
-4.9999806337916	75	[6,7]
no root, unknown reason		[7,8]
-4.9999806337916	36	[8,9]
no root, unknown reason		[9,10]
Running Time (nano-seconds)	2,323,600	