## Contents



## Lab 3

```
lib = make_lib();
```

## Problem 1

```
disp(sprintf('Problem 1 Table:'));
disp(sprintf('Initial Interval \t Approximation \t \t Error \t\t Iterations'));

x0 = 0.0;
x1 = 3.0;
[it_count, root, xn] = bisect(lib.f,x0,x1,lib.accuracy,100);
disp(sprintf('%g to %g \t\t\t %0.10f \t\t %0.10f\t %g', x0, x1, root, ...
                lib.the_root - root, it_count));

x0 = 0.5;
x1 = 2.0;
[it_count, root, xn] = bisect(lib.f,x0,x1,lib.accuracy,100);
disp(sprintf('%g to %g \t\t %0.10f \t\t %0.10f\t %g', x0, x1, root, ...
                lib.the_root - root, it_count));

x0 = 0.9;
x1 = 1.2;
[it_count, root, xn] = bisect(lib.f,x0,x1,lib.accuracy,100);
disp(sprintf('%g to %g \t\t %0.10f \t\t %0.10f\t %g', x0, x1, root,...
                lib.the_root - root, it_count));
```

```
Problem 1 Table:
Initial Interval   Approximation     Error   Iterations
0 to 3   0.9999990463   0.0000009537  19
0.5 to 2   1.0000009537   -0.0000009537  18
0.9 to 1.2   0.9999984741   0.0000015259  15
```

a. The second interval needs exactly one fewer iterations because bisect, by definition, halves the interval for every iteration. The second interval has been effectively halved compared to the first interval so it starts one iteration ahead of the first.

```
disp(sprintf('\n'));
disp(sprintf('Problem 1 Table for part a:'));
x0 = 1;
x1 = 2;
[it_count, root, xn] = bisect(lib.f,x0,x1,lib.accuracy,100);
disp(sprintf('%g to %g \t\t\t %0.10f \t\t %0.10f\t %g', x0, x1, root, ...
                lib.the_root - root, it_count));

x0 = 0.5;
x1 = 1.5;
[it_count, root, xn] = bisect(lib.f,x0,x1,lib.accuracy,100);
disp(sprintf('%g to %g \t\t %0.10f \t\t %0.10f\t %g', x0, x1, root, lib.the_root - root, it_
```

```
Problem 1 Table for part a:
1 to 2   1.0000038147   -0.0000038147  17
0.5 to 1.5   1.0000038147   -0.0000038147  17
```

b. No it doesn't help. Bisection chooses either the left or the right side of the interval so when the root is on the end of an interval it will just choose whichever side the root is on. This doesn't help speed anything up.

## Problem 2

```
disp(sprintf('\n'));
disp(sprintf('Problem 2 Table:'));
disp(sprintf('Initial Value \t Approximation \t \t Error \t\t Iterations'));
x0 = -100;
[it_count, root, xn] = newton(lib.f, lib.fp, x0, lib.accuracy, 100);
disp(sprintf('%g \t\t %0.10f \t\t %0.10f\t\t %g', x0,root, lib.the_root - root, it_count));

x0 = 0;
[it_count, root, xn] = newton(lib.f, lib.fp, x0, lib.accuracy, 100);
disp(sprintf('%g \t\t %0.10f \t\t %0.10f\t\t %g', x0,root, lib.the_root - root, it_count));

x0 = 0.9;
[it_count, root, xn] = newton(lib.f, lib.fp, x0, lib.accuracy, 100);
disp(sprintf('%g \t\t %0.10f \t\t %0.10f\t\t %g', x0,root, lib.the_root - root, it_count));

x0 = 0.99;
[it_count, root, xn] = newton(lib.f, lib.fp, x0, lib.accuracy, 100);
disp(sprintf('%g \t\t %0.10f \t\t %0.10f\t\t %g', x0,root, lib.the_root - root, it_count));

x0 = 1.1;
[it_count, root, xn] = newton(lib.f, lib.fp, x0, lib.accuracy, 100);
```

```
disp(sprintf('%g \t\t %0.10f \t\t %0.10f\t\t %g', x0,root, lib.the_root - root, it_count));

x0 = 1.4;
[it_count, root, xn] = newton(lib.f, lib.fp, x0, lib.accuracy, 100);
disp(sprintf('%g \t\t %0.10f \t\t %0.10f\t\t %g', x0,root, lib.the_root - root, it_count));

x0 = 1000000;
[it_count, root, xn] = newton(lib.f, lib.fp, x0, lib.accuracy, 100);
disp(sprintf('%g \t\t %0.10f \t\t %0.10f\t\t %g', x0,root, lib.the_root - root, it_count));
```

```
Problem 2 Table:
Initial Value   Approximation     Error   Iterations
-100   1.0000000000   0.0000000000  28
0   1.0000000000   0.0000000000  2
0.9   1.0000000000   -0.0000000000  4
0.99   1.0000000000   -0.0000000000  3
1.1   1.0000000000   -0.0000000000  4
1.4   1.0000000000   -0.0000000000  6
1e+06   1.0000000000   -0.0000000000  67
```

a. Newton is much better at finding the root when the initial guess is close to the root.

b. The accuracy of the estimation surpases the required accuracy because on the last iteration Newton makes the error very small very fast, surpassing the required accuracy. Newton converges to the root at a quadratic rate (xˆ2), so ending with the accuracy at 10ˆ-12 rather than 10ˆ-6 makes sense: i.e. (10ˆ-6)ˆ2 == (10ˆ-12)

c. You can calculate the number of iterations by observing: (1/2)ˆn * 1,000,000 <= 1.3x10ˆ-12 Solving for n will give you the number of iterations. In this case n comes out to be 60. Newton takes 67 iterations, so in this case bisection is better.

## Problem 3

```
disp(sprintf('\n'));
disp(sprintf('Problem 3 Table:'));
disp(sprintf('Initial Interval \t Approximation \t \t Error \t\t Iterations'));
x0 = 0.0;
x1 = 3.0;
[it_count, root, xn] = secant(lib.f,x0,x1,lib.accuracy,100);
disp(sprintf('%g to %g \t\t\t %0.10f \t\t %0.10f\t %g', x0, x1, root, lib.the_root - root, i
```

```
x0 = 0.5;
x1 = 2.0;
[it_count, root, xn] = secant(lib.f,x0,x1,lib.accuracy,100);
disp(sprintf('%g to %g \t\t %0.10f \t\t %0.10f\t %g', x0, x1, root, lib.the_root - root, it_

x0 = 0.9;
x1 = 1.2;
[it_count, root, xn] = secant(lib.f,x0,x1,lib.accuracy,100);
disp(sprintf('%g to %g \t\t %0.10f \t\t %0.10f\t %g', x0, x1, root, lib.the_root - root, it_
```

```
Problem 3 Table:
Initial Interval   Approximation     Error   Iterations
0 to 3   1.0000000000   -0.0000000000  6
0.5 to 2   1.0000000000   0.0000000000  10
0.9 to 1.2   1.0000000000   -0.0000000000  6
```

a. When the interval is close to the root secant is better than bisection (by 11 iterations) and worse than Newton (by 4 iterations).

b. The initial size of the interval doesn't effect secant as much as it does bisection. Secant is very simliar to Newton; secant uses derivative information. It accelerates toward the root super-linearly so will make up for a large initiall interval very quickly.

## Problem 4

The errors in the roots start to become large after roots function tries to evaluate the roots higher than 7. This is happens because the polynomial evaluation a the higher roots yeilds very large numbers and as the approximate root get's close to the actual root you start to loose accuracy in your calculations.