Homework #2

2019042497 송정명

```
wjdaud@LAPTOP-1NG9U1J0:~/NA/homework2$ ./homework2
Bracketing routine of the Bessel function J0's Root 1: [2.200000, 2.500000]
Bracketing routine of the Bessel function J0's Root 2: [5.500000, 5.800001]
Bracketing routine of the Bessel function J0's Root 3: [8.500002, 8.800002]

Roots of the Bessel function J0 by using Bisection:
Root 1:   2.404826, Convergence speed(Iteration): 19
Root 2:   5.520078, Convergence speed(Iteration): 19
Root 3:   8.653728, Convergence speed(Iteration): 19
Total Convergence speed(Iteration): 57

Roots of the Bessel function J0 by using Linear interpolation:
Root 1:   2.404826, Convergence speed(Iteration): 5
Root 2:   5.520078, Convergence speed(Iteration): 4
Root 3:   8.653728, Convergence speed(Iteration): 4
Total Convergence speed(Iteration): 13

Roots of the Bessel function J0 by using Secant:
Root 1:   2.404826, Convergence speed(Iteration): 4
Root 2:   5.520078, Convergence speed(Iteration): 4
Root 3:   8.653728, Convergence speed(Iteration): 4
Total Convergence speed(Iteration): 12

Roots of the Bessel function J0 by using Newton-Raphson:
Root 1:   2.404825, Convergence speed(Iteration): 3
Root 2:   5.520078, Convergence speed(Iteration): 3
Root 3:   8.653728, Convergence speed(Iteration): 2
Total Convergence speed(Iteration): 8

Roots of the Bessel function J0 by using Newton with bracketing:
Root 1:   2.404825, Convergence speed(Iteration): 3
Root 2:   5.520078, Convergence speed(Iteration): 3
Root 3:   8.653728, Convergence speed(Iteration): 2
Total Convergence speed(Iteration): 8
```

일단 zbrak.c에 있는 zbrak() 함수를 사용해서 bracketing routine을 사용해 [1.0, 10.0] 구간에서 Bessel function J0의 해가 있는 구간들의 정보를 받아왔습니다. (이때 구간은 30개로 나눴습니다.)

이후 Bisection (rtbis.c), Linear interpolation (rtflsp.c), Secant (rtsec.c), Newton-Raphson (rtnewt.c), Newton with bracketing (rtsafe.c) 방법들을 사용해서 각 해를 구해주었습니다. 해를 구할 때 xacc를 (1.0e-6)로 설정해서 상대 오차가 10^-6보다 작아지면 수렴했다고 판단하도록 했습니다.

```
Roots of the Bessel function J0 by using Muller:
Root 1:   2.404825, Convergence speed(Iteration): 3
Root 2:   5.520078, Convergence speed(Iteration): 3
Root 3:   8.653728, Convergence speed(Iteration): 3
Total Convergence speed(Iteration): 9
```

또한, 수업 시간에 배운 Muller method를 토대로 하여 muller.c에 Muller method를 만들고 해를 찾아보았습니다.

그리고 각 방법들의 convergence speed을 알아내기 위해 함수들의 코드에 count를 추가해주었습니다. 이를 통해 하나의 해를 찾을 때 몇 번의 반복을 하는지 다 계산해서 최종적으로 모든 해를 찾는 동안 몇 번의 반복을 하는지 알아보았습니다. 그 결과 이론 시간에 배웠던 것과 비슷하게 Bisection, Linear interpolation, Secant, Newton-Raphson 순서로 더 빨라진다(더 적은 횟수의 반복을 한다)는 사실을 알 수 있었습니다.

```
Bracketing routine of my nonlinear equation's Root 1: [1.000000, 1.009000]
Bracketing routine of my nonlinear equation's Root 2: [2.998001, 3.007001]
Bracketing routine of my nonlinear equation's Root 3: [4.995988, 5.004988]
Bracketing routine of my nonlinear equation's Root 4: [7.992930, 8.001930]

Roots of my nonlinear equation by using Newton with bracketing:
Root 1:   1.000000
Root 2:   3.000000
Root 3:   5.000000
Root 4:   8.000000
```

마지막으로 나만의 nonlinear equation(= (x - 1) * (x - 3) * (x - 5) * (x - 8))를 만들어 rtsafe.c에 있는 Newton with bracketing 함수를 사용하기 위해 먼저 bracketing routine을 사용해 이전과 똑같이 해가 있는 구간들을 찾아주었고, 이후 rtsafe() 함수를 사용해 해를 구했습니다.