Homework#3

2017029589 컴퓨터소프트웨어학부 류지범

1. Find the roots of the Bessel function

모든 코드는 main.cpp 에 구현되어있다.

```
Bisection Method
1'th root: 2.404826172
2'th root: 5.520077734
3'th root: 8.65372832
14ms passed

Linear interpolation
1'th root: 2.40482556
2'th root: 5.520078106
3'th root: 8.653727913
7ms passed

Secant Method
1'th root: 2.40482556
2'th root: 5.520078106
3'th root: 8.653727913
6ms passed

Newton-Raphson Method
1'th root: 2.40482556
2'th root: 5.520078106
3'th root: 8.653727913
7ms passed

Newton with bracketing Method
1'th root: 2.40482556
2'th root: 5.520078106
3'th root: 8.653727913
5ms passed

Muller Method
1'th root: 2.40482556
2'th root: 5.520078106
3'th root: 8.653727913
6ms passed
```

각 method에 대한 결과와 걸린 시간을 `ms`단위로 표현했다.

`zbrak`함수를 사용해서 root가 존재하는 구간을 `xb1``xb2` vector에 담고 각 method에 해당하는 함수를 사용해서 root를 구했다.

2. Muller method

Main.cpp 에 `muller` 함수를 같이 작성했다.

```
DP muller(DP func(const DP), const DP x1, const DP x2, const DP xacc) {
    const int MAXIT = 30;
    double a, b, c, p0, p1, p2, p3, h1, h2, d1, d2, rpe;
    p0 = x1;
    p1 = x2;
    p2 = (x1 + x2) / 2.0;

    for (int i = 0; i < MAXIT; ++i) {
        h1 = p1 - p0;
        h2 = p2 - p1;
        d1 = (func(p1) - func(p0)) / h1;
        d2 = (func(p2) - func(p1)) / h2;
        a = (d2 - d1) / h2 + h1;
        b = a * h2 + d2;
        c = func(p2);
        double temp = b * b - 4 * a * c;
        if (b < 0) {
            p3 = p2 + (-2.0 * c) / (b - sqrt(temp));
        } else {
            p3 = p2 + (-2.0 * c) / (b + sqrt(temp));
        }
        rpe = fabs((p3 - p2) / p3);
        if (rpe < xacc) return p3;
        p0 = p1;
        p1 = p2;
        p2 = p3;
    }
    nrerror("Maximum number of iterations exceeded in muller");
    return 0.0;
}
```

`p0`, `p1`, `p2`를 공식에 따라 계속 갱신해나가고, `relative error`가 `xacc`보다 작아질 때의 `p3`를 root로 반환하도록 했다.

3. Convergence speed

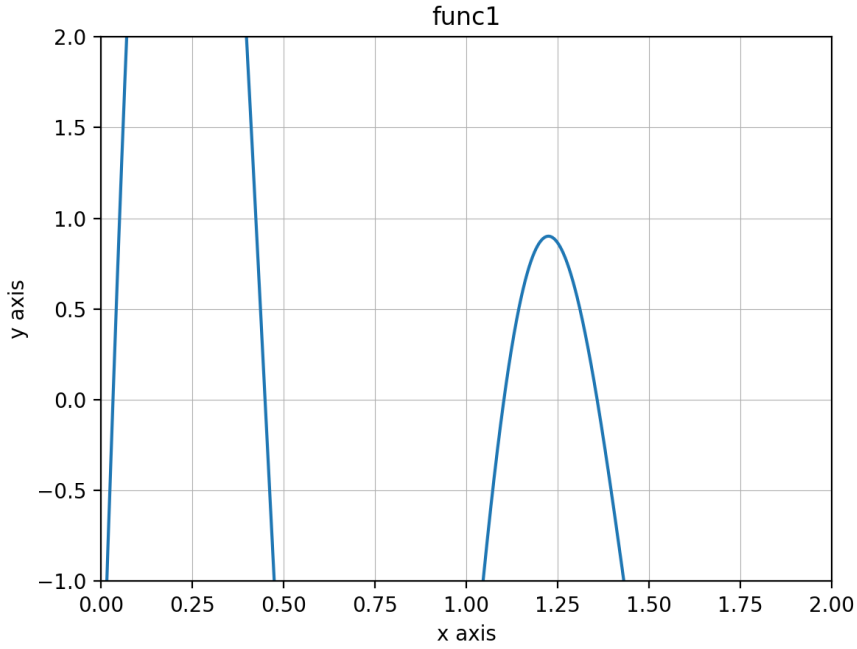
Method	Convergence Speed
Bisection	**14ms**
Linear Interpolation	7ms
Secant	6ms
Newton-Raphson	7ms
Newton with bracketing	5ms
Muller	6ms

method의 실행 전 후를 기점으로 `clock()`를 통해 시간을 측정했다. `Bisection method`를 제외한 나머지는 거의 다 `5~6ms`로 비슷한 수렴 속도를 보여주었고 `Bisection method`만 `14ms`로 비교적 느린 수렴 속도를 보여주었다.

4. Solve problem using rtsafe

모든 함수는 python matplotlib 의 결과와 비교했다.

1)
- f(x) = 10e^xsin(2 * pi * x) - 2 = 0 on [0.1, 1]



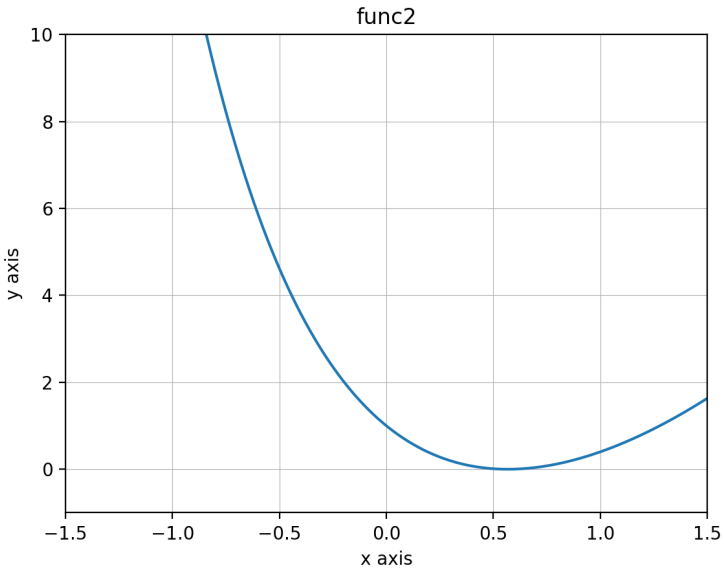
- f'(x) = 10e^(-x) * (2 * pi * cos(2 * pi * x) - sin(2 * pi * x))

result

```
func1 in [0.1, 1]
1'th root: 0.4492608307
```

2)

- f(x) = x^2 - 2xe^-2x + e^-2x = 0 on [0, 1]
- f''(x) = 2e^(-2x) * (e^x + 1) * (e^x * x - 1)

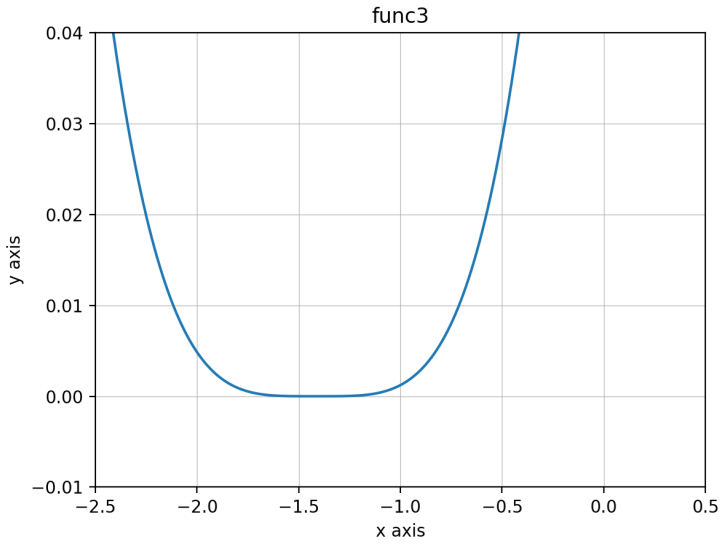


result

```
func2 in [0, 1]
1'th root: 0.5671432872
2'th root: 0.5671432927
```

3)

- f(x) = cos(x + sqrt(2)) + x(x/2 + sqrt(2)) = 0 on [-2, 1]
- f'(x) = x - sin(x + sqrt(2)) + sqrt(2)

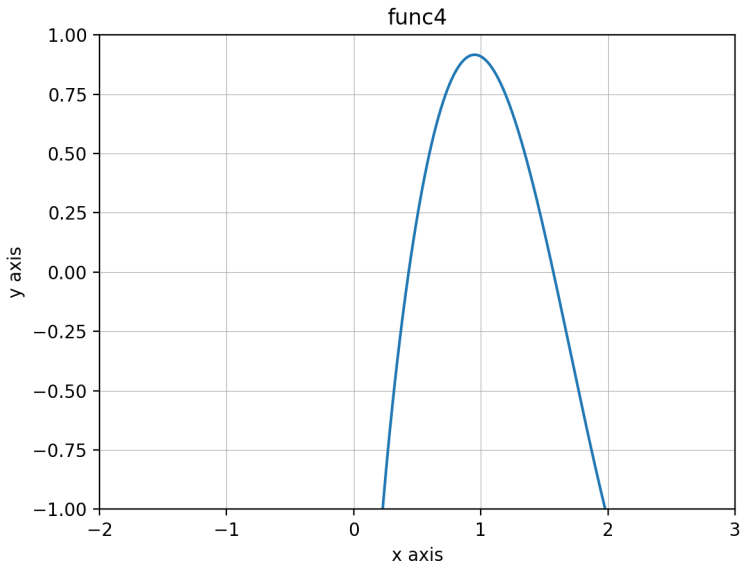


result

```
func3 in [-2, -1]
1'th root: 0.4324731422
2'th root: 1.570796327
```

Interesting nonlinear equation I want to solve

- f(x) = cos(x)ln(x) + sin(2x)
- f'(x) = cos(x)/x + 2cos(2x) - ln(x)sin(x)



result

```
func4 in [0, 2]
1'th root: 0.4324731422
2'th root: 1.570796327
```