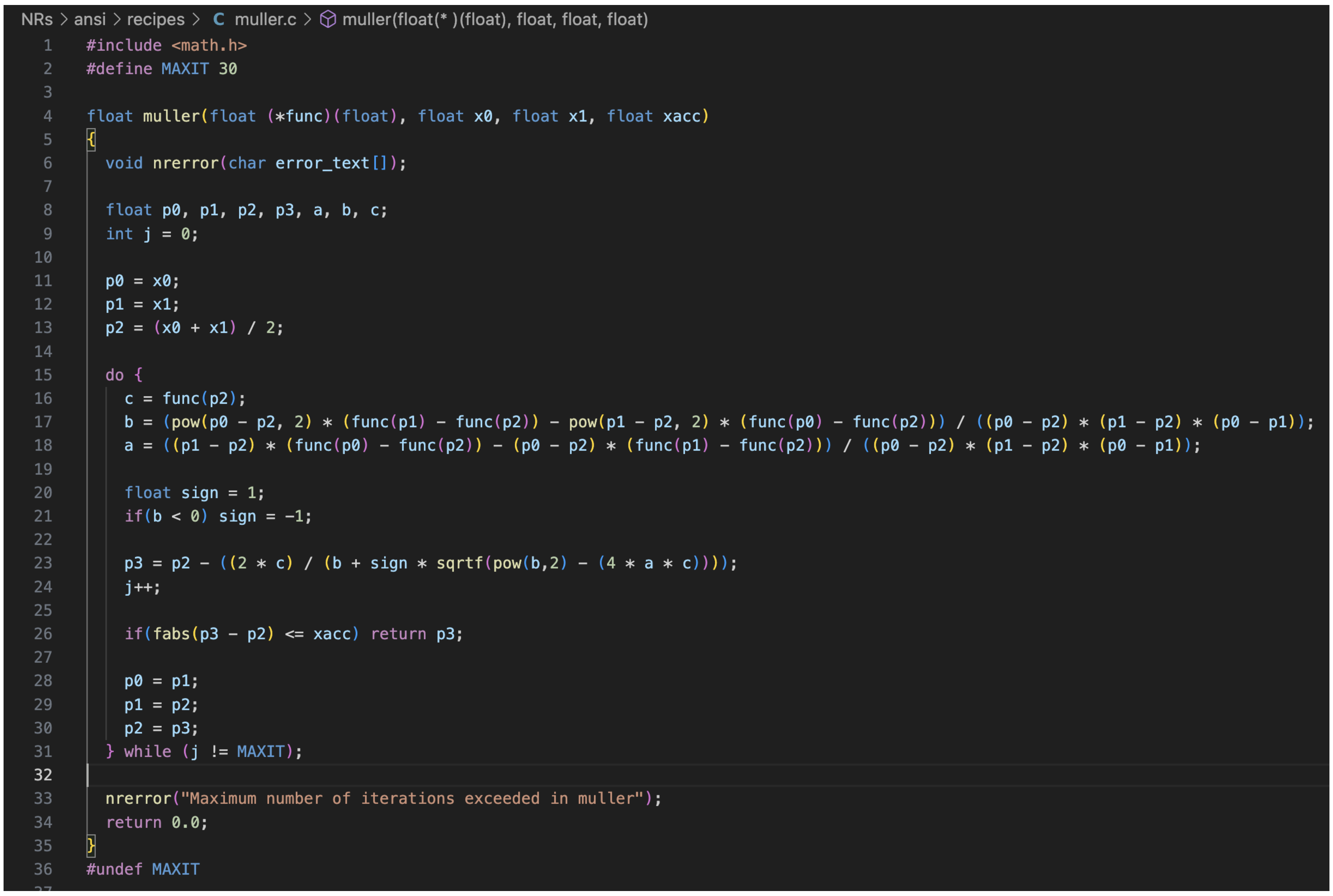
Homework#2

컴퓨터소프트웨어학부 2019030991 홍정범

Write source codes for Muller method (muller.c) and do the same job as above.

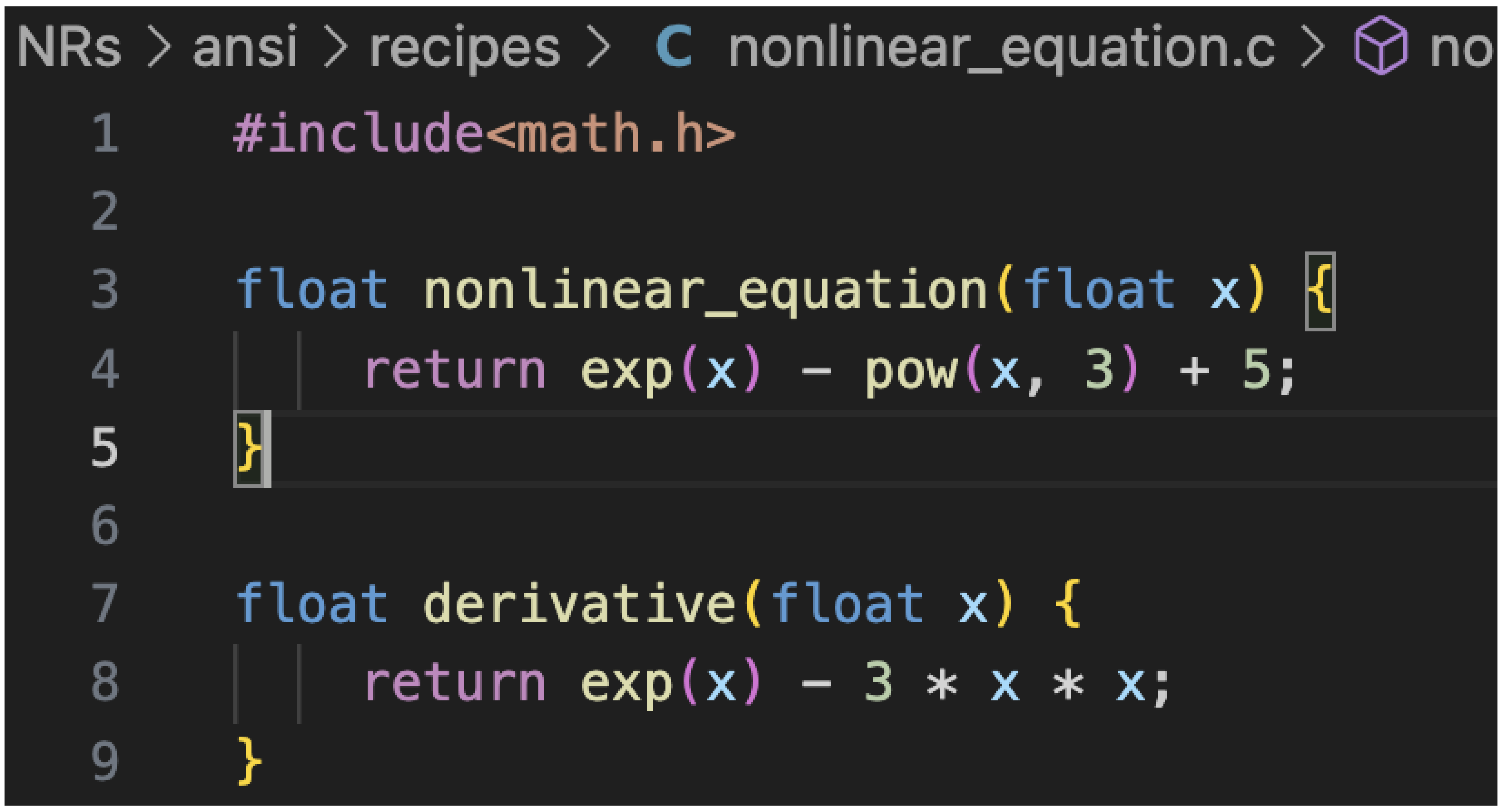


NRs > ansi > recipes > C muller.c > muller(float(*)(float), float, float, float)

```c
#include <math.h>
#define MAXIT 30

float muller(float (*func)(float), float x0, float x1, float xacc)
{
  void nrerror(char error_text[]);

  float p0, p1, p2, p3, a, b, c;
  int j = 0;

  p0 = x0;
  p1 = x1;
  p2 = (x0 + x1) / 2;

  do {
    c = func(p2);
    b = (pow(p0 - p2, 2) * (func(p1) - func(p2)) - pow(p1 - p2, 2) * (func(p0) - func(p2))) / ((p0 - p2) * (p1 - p2) * (p0 - p1));
    a = ((p1 - p2) * (func(p0) - func(p2)) - (p0 - p2) * (func(p1) - func(p2))) / ((p0 - p2) * (p1 - p2) * (p0 - p1));

    float sign = 1;
    if(b < 0) sign = -1;

    p3 = p2 - ((2 * c) / (b + sign * sqrtf(pow(b,2) - (4 * a * c))));
    j++;

    if(fabs(p3 - p2) <= xacc) return p3;

    p0 = p1;
    p1 = p2;
    p2 = p3;
  } while (j != MAXIT);

  nrerror("Maximum number of iterations exceeded in muller");
  return 0.0;
}
#undef MAXIT
```

Discuss the convergence speed of the methods & Solve one interesting nonlinear equation you want to solve using the routine of rtsafe.c in NR in C

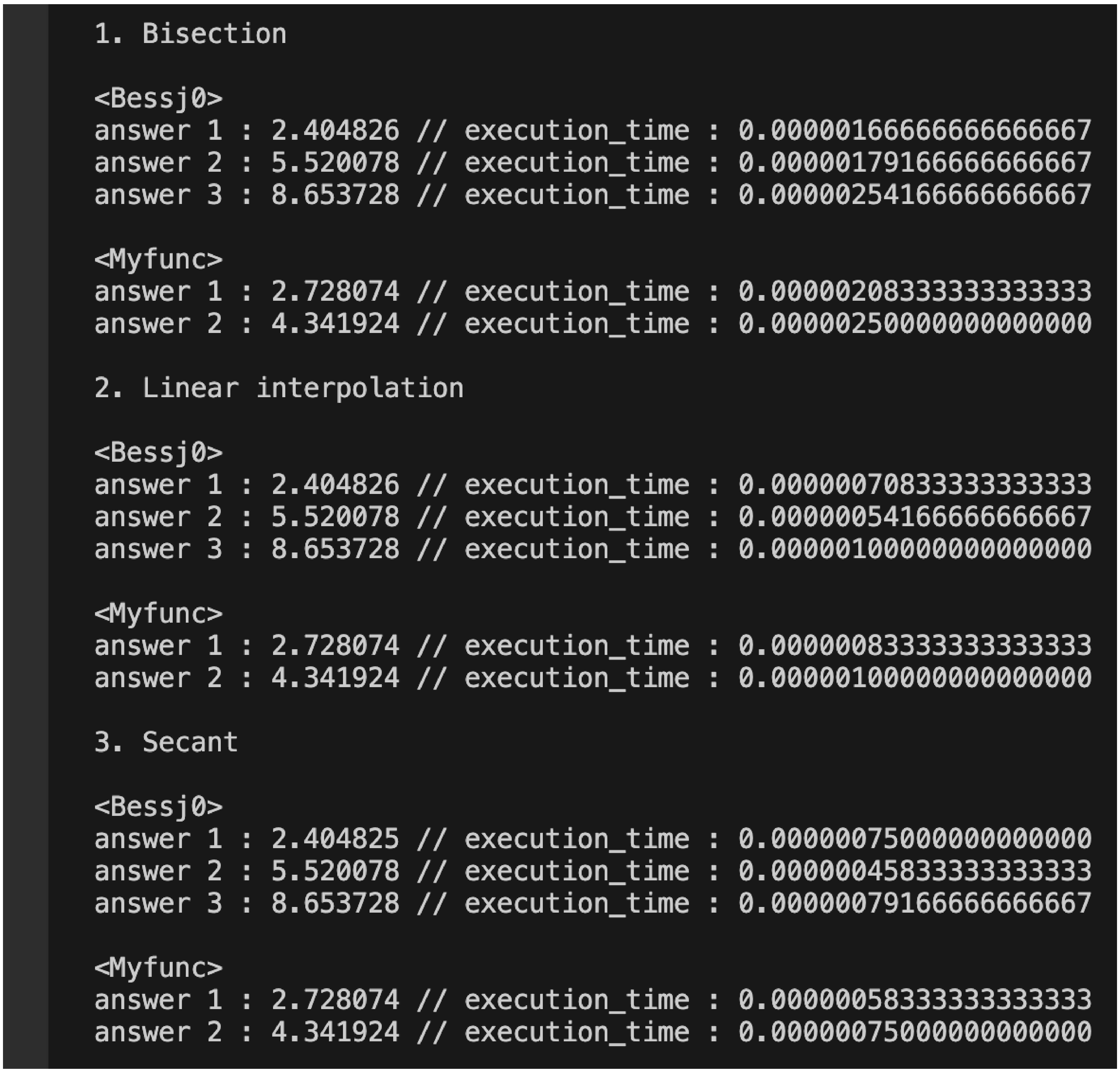
제가 선택한 함수와 그 함수의 도함수입니다.



NRs > ansi > recipes > C nonlinear_equation.c > no

```c
#include<math.h>

float nonlinear_equation(float x) {
    return exp(x) - pow(x, 3) + 5;
}

float derivative(float x) {
    return exp(x) - 3 * x * x;
}
```

각 함수들을 실행한 후 실행 시간을 MacOS 기반에서 mach/mach_time.h를 이용하여 연산하여 출력하였습니다.



```
1. Bisection

<Bessj0>
answer 1 : 2.404826 // execution_time : 0.00000166666666666667
answer 2 : 5.520078 // execution_time : 0.00000179166666666667
answer 3 : 8.653728 // execution_time : 0.00000254166666666667

<Myfunc>
answer 1 : 2.728074 // execution_time : 0.00000208333333333333
answer 2 : 4.341924 // execution_time : 0.00000250000000000000

2. Linear interpolation

<Bessj0>
answer 1 : 2.404826 // execution_time : 0.00000070833333333333
answer 2 : 5.520078 // execution_time : 0.00000054166666666667
answer 3 : 8.653728 // execution_time : 0.00000100000000000000

<Myfunc>
answer 1 : 2.728074 // execution_time : 0.00000083333333333333
answer 2 : 4.341924 // execution_time : 0.00000100000000000000

3. Secant

<Bessj0>
answer 1 : 2.404825 // execution_time : 0.00000075000000000000
answer 2 : 5.520078 // execution_time : 0.00000045833333333333
answer 3 : 8.653728 // execution_time : 0.00000079166666666667

<Myfunc>
answer 1 : 2.728074 // execution_time : 0.00000058333333333333
answer 2 : 4.341924 // execution_time : 0.00000075000000000000
```

```
4. Newtion-Raphson

<Bessj0>
answer 1 : 2.404825 // execution_time : 0.0000005416666666667
answer 2 : 5.520078 // execution_time : 0.0000004166666666667
answer 3 : 8.653728 // execution_time : 0.0000008333333333333

<Myfunc>
answer 1 : 2.728074 // execution_time : 0.0000009166666666667
answer 2 : 4.341924 // execution_time : 0.0000007083333333333

5. Newton with bracketing

<Bessj0>
answer 1 : 2.404825 // execution_time : 0.0000010833333333333
answer 2 : 5.520078 // execution_time : 0.0000007083333333333
answer 3 : 8.653728 // execution_time : 0.0000011250000000000

<Myfunc>
answer 1 : 2.728074 // execution_time : 0.0000006250000000000
answer 2 : 4.341924 // execution_time : 0.0000009583333333333

5. muller's method

<Bessj0>
answer 1 : 2.404825 // execution_time : 0.0000017916666666667
answer 2 : 5.520078 // execution_time : 0.0000012083333333333
answer 3 : 8.653728 // execution_time : 0.0000024166666666667

<Myfunc>
answer 1 : 2.728074 // execution_time : 0.0000020000000000000
answer 2 : 4.341924 // execution_time : 0.0000017083333333333
```