MAP 5611 Intro to Computational Finance HW2

Student: Sen Zhang

Problem 1 1

Problem 2 4

Problem 1

(a)

$$\nu = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - m\right)^2 \qquad (1)$$

$$\nu = \frac{1}{n}\sum_{i=1}^{n}x_i^{\,2} - m^2 \qquad (2)$$

To show the two formulas are algebraically equivalent:

$$\begin{aligned}
\nu &= \frac{1}{n}\sum_{i=1}^{n}\left(x_i - m\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(x_i^{\,2} - 2x_i m + m^2\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}x_i^{\,2} + m^2 - 2\frac{1}{n}\sum_{i=1}^{n}x_i m \\
&= \frac{1}{n}\sum_{i=1}^{n}x_i^{\,2} + m^2 - 2m^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}x_i^{\,2} - m^2
\end{aligned}$$

To show how the formula (2) can update continuously:

Assume the set of data is $\{x_i\}_{i=1}^{n}$, look at the condition of adding a x_{n+1}:

For formula (1), after the set $\{x_i\}_{i=1}^{n}$ was inputted the calculators will store n, m and $\sum_{i=1}^{n}(x_i-m)^2$. The m will change when adding a x_{n+1}. Then the values, $\sum_{i=1}^{n}(x_i-m)^2$ stored in calculators cannot be used to calculate the $\sum_{i=1}^{n+1}(x_i-m)^2$. Thus we cannot get the variance of $\{x_i\}_{i=1}^{n+1}$.

For formula (2), after the set $\{x_i\}_{i=1}^{n}$ was inputted the calculators will store n, m and $\sum_{i=1}^{n}x_i^2$. If adding a x_{n+1}, then $\sum_{i=1}^{n+1}x_i^2=\sum_{i=1}^{n}x_i^2+x_{n+1}^2$ and $m=\frac{nm+x_{n+1}}{n+1}$. Thus it is possible for formula (2) to update continuously.

(b)

It will cause errors when a real number is approximated by rounding off to a nearby floating point number. So we need to discuss the rounding error properties of algorithms.

For formula (1):

Nearly equal values:

It is likely to lose significant digits when a subtraction occurs between two nearly values.

For example $\beta=10, t=2, L=-1, U=1$

The real numbers are a=9.75, b=9.74, c=9.79 and m=9.76.

Then v=1.4*10^(-3).

The floating point number fl(a)=9.8, fl(b)=9.7, fl(c)=9.8, fl(m)=9.8

Then in formula (1) fl(v)=0.10,

Thus the relative error is very big.

Values with great difference:

The square of this difference might cause infinite. This might lead to an INF. For example in single precision one element of the set $x_1=10^{20}$ and m is very small, then v is INF.

For formula (2):

Small values

The substation and the square of small values might cause loss of significant digits.

Big values

The square of very large values might cause infinite.

(b)'

Meaningless result given by formula (2)

In particular, the formula (2) can give a meaningless result for very large numbers.

For example, in double precision $x_i = 10^{200}, \text{for } i = 1,\ 2,\ \ldots n;\ m = 10^{200}$

For formula (1), we can get the right variance of the set.

$$\begin{aligned} \nu &= \frac{1}{n}\sum_{i=1}^{n}(x_i - m)^2 \\ &= 0 \end{aligned}$$

But for formula (2), $x_i^2 = \text{INF}, \text{for } i = 1,\ 2,\ \ldots n;\ m^2 = \text{INF}$

$$\begin{aligned} & x_i^2 = \text{INF}, \text{for } i = 1,\ 2,\ \ldots n; \\ & m^2 = \text{INF} \\ & \text{Then } \nu = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - m^2 = \text{INF} - \text{INF} \\ & \qquad = \text{INF} \end{aligned}$$

Problem 2

I. Executive Summary

In this report, the root of the given function is found in three methods, Bisection Method, Newton-Raphson Method and Wrapper Method which is comprised of Bisection and Newton routines. The root is 0.29849060201 in double precision , and 0.29849049449 in float precision. In addition, the advantages and disadvantages of different methods were shown under the change of $[a,\ b]$ and startValue.

II. Statement of Problem

To add a new class with three methods to the toolbox, which are Bisection Method, Newton-Raphson Method and a Wrapper Method which is comprised of Bisection and Newton routines.

Then use these three methods to find all roots of the function:

$$f(v) = v(\frac{1}{R} + \mu\gamma) - \mu v^2 + \alpha(e^{v/\beta} - 1) - \frac{E}{R} = 0$$

$$\frac{E}{R} = 1.2\times10^{-4},\ \frac{1}{R} = 3\times10^{-4},\ \alpha = 10^{-12},\ \beta^{-1} = 40,\ \mu = 10^{-3},\ \gamma = 0.4$$

III. Description of The Mathematics

Bisection Method:

Theorem 1: If *f(x)* is continuous function on interval [*a, b*] and if *f(a)f(b)<0,* then *f(p) = 0*, for some $p \in [a,\ b]$.

From theorem 1, we can find the root of *f(x) = 0* when *f(x)* is continuous function on interval [*a, b*] and *f(a)f(b)<0.* The idea of finding the root is as following:

Assume $a_0 = a,\ b_0 = b$

Then $p_i = \dfrac{a_i + b_i}{2}$

If $f(a_i) f(p_i) < 0$

Let $b_{i+1} = p_i,\ a_{i+1} = a_i$

Else $b_{i+1} = b_i,\ a_{i+1} = p_i$

...

$p_n = \dfrac{a_n + b_n}{2} \to \text{root } x, \text{ as } n \to \infty$

However, it is possible that $f(p_n) \neq 0$ because of rounding errors. So a reasonable tolerance should be allowed for p_n to converge.

Considering the following two convergence conditions:

$$| p_n - p_{n-1} | < absTol \qquad (3)$$

$$\left| \frac{p_n - p_{n-1}}{p_n} \right| < relTol \qquad (4)$$

Because the absolute error is not good for large value of p_n and relative error is not good for small value of p_n. A possible test for convergence is to combine them

$$| p_n - p_{n-1} | \leq \text{absTol} + \text{relTol} | p_n |$$

$$| p_n - p_{n-1} | \leq \frac{b-a}{2^{n+1}}$$

$$\Rightarrow \frac{b-a}{2^{n+1}} \leq \text{absTol} + \text{relTol} | p_n |$$

$$\Rightarrow n + 1 \geq \log_2((b-a) / (absTol + relTol | p_n |)$$

$$\Rightarrow n + 1 \geq \log_2((b-a) / (absTol + relTol \min(| a |, | b |))$$

$$\Rightarrow n \geq \log_2((b-a) / (absTol + relTol \min(| a |, | b |)) - 1$$

Newton-Raphson Method:

Firstly, we use a linear Taylor polynomial to approximate the function and then find the root of the linear interpolation.

Assume *f(x)* is sufficiently smooth.

Then

$$0 = f(x) = f(p_n) + (x - p_n) f'(p_n) + \mathrm{O}((x - p_n)^2)$$

It is reasonable to ignore the $\mathrm{O}((x - p_n)^2)$ if p_n is near root *x.*

Then call the root p_{n+1}, $0 = f(p_{n+1}) = f(p_n) + (x - p_n) f'(p_n)$

$$\Rightarrow 0 = f(p_n) + (p_{n+1} - p_n) f'(p_n)$$
$$\Rightarrow p_{n+1} = p_n - \frac{f(p_n)}{f'(p_n)} \rightarrow x, \text{ as } n \rightarrow \infty$$

However, it is possible that $f(p_{n+1}) \neq 0$ because of rounding errors. So a reasonable tolerance should be allowed. A possible tolerance is

$$\left| \frac{f(p_n)}{f'(p_n)} \right| \leq \mathrm{absTol} + \mathrm{relTol} \,|\, p_n \,|$$

Wrapper Method:

This method can firstly get a solution to within one or two significant digits using Bisection Method. Then this solution can be passed to the Newton Method, which it to return the result to machine accuracy.

IV. Description of The Algorithm

Bisection Method:

$for\ i = 1\ to\ n\ \ do$

$p = a + \frac{b-a}{2};$

$if\ \ \frac{b-a}{2} \le absTol + relTol * abs(p)$ Exit

if sign($f(a)$)*sign($f(p)$)<0

b=p;

else

$a = p;$

end if

repeat

For n, we have $n \ge \log_2((b-a)/(absTol + relTol \min(|a|,|b|)) - 1,$

Then let $n = \log((b-a)/(absTol + relTol \min(|a|,|b|))/\log 2$

The detail of this algorithm is as following:

```
//Bisection Method
FM Solver::Bisect(FM f(FM p), FM a, FM b, FM absTol, FM relTol)
{
        FM p;
        int n;
        n= int( log( (b-a)/(absTol + relTol*min( abs(a), abs(b) )) ) / log(2.0));
        for (int i = 1; i <= n; i++)
        {
                p = a + (b - a)/2;
                if ( abs((b-a))/2 <= absTol + relTol* abs(p) )
                {
                        break;
                }
                else
                {
                    if ( sign(f(a)) * sign(f(p)) < 0 )
                    {
                                b = p;
                    }
                    else
                    {
                            a = p;
                    }
                }
                cout<<"The number of iteration by Bisection Method: "<<i<<endl;
        }
```

```
    return p;
}
```

Newton-Raphson Method:

$$
\begin{aligned}
&for\ i = 1\ to\ n\ \ do \\
&\Delta = -\frac{f(p)}{f'(p)}; \\
&p = p + \Delta \\
&if\ \ |\Delta| \le absTol + relTol * abs(p)\ \text{Exit} \\
&end\ \text{if} \\
&\text{repeat} \\
&\text{Note: } n \in [5,\ 10]
\end{aligned}
$$

The detail of this algorithm is as following:

```
// Newton-Raphson Method
FM  Solver::Newton(FM f(FM p), FM fp(FM p), FM startValue, FM absTol, FM relTol)
{
        FM p, delta;
        p = startValue;
        int n = 10;
        for (int i=1; i<=n; i++)
        {
                delta = -f(p)/fp(p);
                p = p + delta;
                if (abs(delta) <= absTol + abs(p)*relTol)
                {
                        break;
                }
                cout<<"The number of iteration by Newton Method: "<<i<<endl;
        }
        return p;
}
```

Wrapper Method:

The detail of this algorithm is as following:

```
//Combination of Bisection and Newton-Raphson Method
FM  Solver::Wrapper(FM f(FM p), FM fp(FM p), FM a, FM b, FM startValue, FM absTol, FM relTol)
{
    FM N;
        startValue = Bisect(f, a, b, 0.1, 0.1);
        N = Newton(f, fp, startValue, absTol, relTol);
        return N;
}
```

V. Results

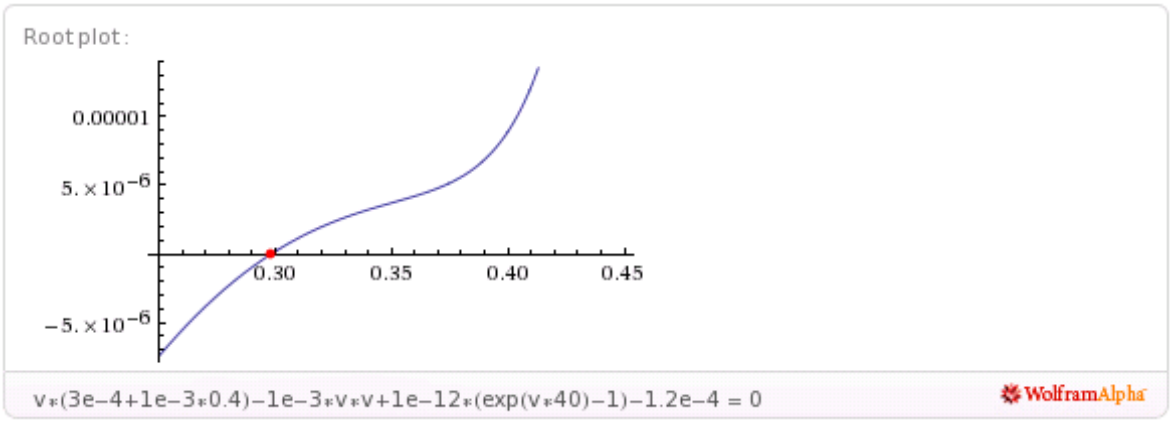
For the function

$$f(v) = v(\frac{1}{R} + \mu\gamma) - \mu v^2 + \alpha(e^{v/\beta} - 1) - \frac{E}{R} = 0$$

$$\frac{E}{R} = 1.2\times10^{-4},\ \frac{1}{R} = 3\times10^{-4},\ \alpha = 10^{-12},\ \beta^{-1} = 40,\ \mu = 10^{-3},\ \gamma = 0.4$$

To make sure the endpoints$[a,\ b]$, look at the graph of the function. We can observe that there is only one root for the function, and this root is in the range of [0, 1].

Note: this graph is gained on the website www.wolframalpha.com



Then let relTol=5×10^{-11}, absTol = machine epsilon in float and double precisions.

We got the results under different $[a,\ b]$ and startValues:

Table 1 Root found by Bisection Method

	$[a,\ b]$	Root	Iteration#
Double precision	[0.0, 1.0]	0.29849060201	35
Double precision	[-1.0, 1.0]	0.298490602	35
Double precision	[0.0, 2.0]	0.29849060201	36
Float precision	[0.0, 1.0]	0.29849028587	22
Float precision	[-1.0, 1.0]	0.29849028587	23
Float precision	[0.0, 2.0]	0.29849040508	23

Advantage: From this table, we can see that Bisection Method always converges.

Disadvantage: Form this table, we can see that Bisection Method is too slow.

Table 2 Root found by Newton-Raphson Method

	startValue	Root	Iteration#	Success
Double precision	0.0	0.29849060201	7	Yes
Double precision	1.0	0.75000023047	9	No
Double precision	-1.0	0.29849060201	9	Yes
Float precision	0.0	0.29849049449	6	Yes
Float precision	1.0	0.75000041723	9	No
Float precision	-1.0	0.29849049449	8	Yes

Advantage: From this table, we can see that Newton Method is fast.

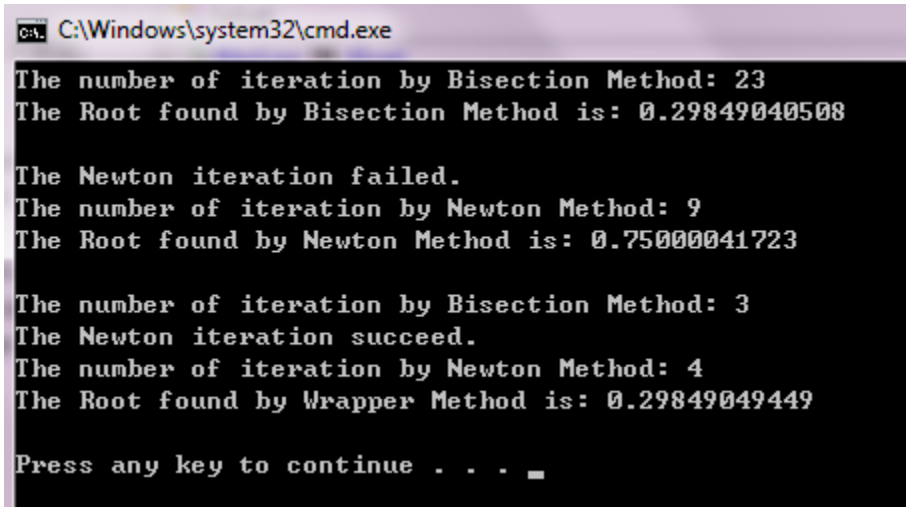
Disadvantage: From this table, we can see that Newton Method does not always converge.

Table 3 Root found by Wrapper Method

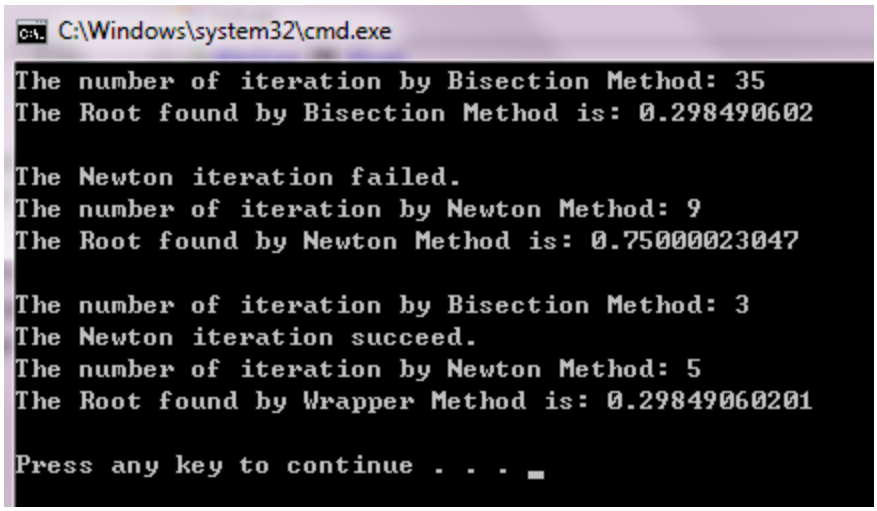
	$[a,\ b]$	Root	Iteration# in Bisection	Iteration# in Newton
Double precision	[0.0, 1.0]	0.29849060201	2	5
Double precision	[-1.0, 1.0]	0.29849060201	3	5
Double precision	[0.0, 2.0]	0.29849060201	3	5
Float precision	[0.0, 1.0]	0.29849049449	2	4
Float precision	[-1.0, 1.0]	0.29849049449	3	4
Float precision	[0.0, 2.0]	0.29849049449	3	4

Advantage: From this table, we can see that wrapper Method is fast, and it converges.

Graph 1 Results of running codes in Float precision



Graph 2 Results of running codes in Double precision



VI. Conclusions

The root is 0.29849060201 in double precision , and 0.29849049449 in float precision.

Analysis of advantages and disadvantages in different methods:

Bisection Method:

Advantage: From table 1, we can see that Bisection Method always converges after checking different $[a, b]$.

Disadvantage: From table1 and table 2, we can see that Bisection Method is much slower than Newton-Raphson Method comparing the numbers of iteration they need to get the root.

Newton-Raphson Method:

Advantage: From table1 and table 2, we can see that Newton Method is much faster than Bisection Method comparing the numbers of iteration they need to get the root.

Disadvantage: From table 2, we can see that Newton Method does not always converges since when let startValue =1.0, the iteration failed.

Wrapper Method:

Advantage: Comparing these three tables, we can find that the Wrapper Method combined the advantages of Bisection and Newton Methods, and avoid their disadvantages. That means the Wrapper Method is fast, and does converge.

VII. Program Listing

Solver.h

```
//head file: Solver.h
#define FM float
#pragma once
class Solver
{public:
        FM  Bisect(FM f(FM p), FM a, FM b, FM absTol, FM relTol);
        FM  Newton(FM f(FM p), FM fp(FM p), FM startValue, FM absTol, FM relTol);
        FM  Wrapper(FM f(FM p), FM fp(FM p), FM a, FM b, FM startValue, FM absTol, FM relTol);
};
```

Solver.cpp

```
//source file: Solver.cpp includes three methods
#include <iostream>
#include <cmath>
#include "Solver.h"
#include "FMfloat.h"
using namespace std;

// to return the sigh of a value
int sign(FM s)
{
        int h=0;
        if (s>0.0)
        {
                h = 1;
        }
        if (s<0.0)
        {
                h = -1;
        }
        return h;
}

//Bisection Method
FM Solver::Bisect(FM f(FM p), FM a, FM b, FM absTol, FM relTol)
{
        FM p;
        int n, k;
        n= int( log( (b-a)/(absTol + relTol*min( abs(a), abs(b) )) ) / log(2.0));
        for (int i = 1; i <= n; i++)
        {
                p = a + (b - a)/2;
                if ( abs((b-a))/2 <= absTol + relTol* abs(p) )
                {
                        break;
                }
                else
```

```
                {
                        if ( sign(f(a)) * sign(f(p)) < 0 )
                    {
                                b = p;
                    }
                    else
                    {
                            a = p;
                    }
                }
                k=i;
        }
        cout<<"The number of iteration by Bisection Method: "<<k<<endl;
    return p;
}


// Newton-Raphson Method
FM  Solver::Newton(FM f(FM p), FM fp(FM p), FM startValue, FM absTol, FM relTol)
{
        FM p, delta;
        FMfloat fm;
        FM s =0.0;
        p = startValue;
        int k;
        int n = 10;
        for (int i=1; i<=n; i++)
        {
                delta = -f(p)/fp(p);
                p = p + delta;
                if (abs(delta) <= absTol + abs(p)*relTol)
                {
                        cout<<"The Newton iteration succeed."<<endl;
                break;
                }
                if ( i>=10 && abs(delta) > abs(p)*fm.epsilon(s))
                {
                        cout<<"The Newton iteration failed."<<endl;
                        break;
                }
                k=i;
        }
        cout<<"The number of iteration by Newton Method: "<<k<<endl;
        return p;
}

//Combination of Bisection and Newton-Raphson Method
FM  Solver::Wrapper(FM f(FM p), FM fp(FM p), FM a, FM b, FM startValue, FM absTol, FM relTol)
{
    FM N;
        startValue = Bisect(f, a, b, 0.1, 0.1);
        N = Newton(f, fp, startValue, absTol, relTol);
        return N;
}
```

Test.cpp

```
//source file: Test.cpp  includes the function f(v), the first prime of function fp(v) and the
main function
#include <iostream>
#include <cmath>
#include <iomanip>
#include "Solver.h"
#include "FMfloat.h"
using namespace std;

//Define function f(v)
FM f(FM v)
{
        FM R_1,mu,gamma,alpha,beta_1, E_R;
        alpha= 1e-12;
        beta_1=40.0;
        gamma=0.4;
        mu=1e-3;
        E_R=1.2e-4;
        R_1=3.0e-4;
        return FM (v*(R_1 + mu*gamma)-mu*v*v + alpha*(exp(v*beta_1)-1.0) - E_R);
        //FM y;
        //y= FM( v*(3e-4+1e-3*0.4)-1e-3*v*v+1e-12*(exp(v*40)-1)-1.2e-4 );
        //return y;
}

//Define the first prime of function f(v)
FM fp(FM v)
{
        FM y;
        y= FM( 3e-4+1e-3*0.4-2e-3*v+40*1e-12*exp(v*40) );
        return y;
}

int main()
{
        FMfloat fm;
        Solver rt;
        FM a, b, startValue;
        FM x,y,z;
        startValue=1.0; a=0.0;  b=2.0;
        FM relTol =  5e-11;
        FM absTol = fm.epsilon(a);

        x = rt.Bisect(f, a, b, absTol, relTol);
        cout << "The Root found by Bisection Method is: " <<setprecision(11)<< x << endl<<endl;

        y = rt.Newton(f, fp, startValue, absTol, relTol);
        cout << "The Root found by Newton Method is: " <<setprecision(11)<< y << endl<<endl;

        z = rt.Wrapper(f, fp, a, b, startValue, absTol, relTol);
        cout << "The Root found by Wrapper Method is: " <<setprecision(11)<< z << endl<<endl;
    return 0;
}
```