Jaime Frade
Computational Finance: Dr. Kopriva
Homework #3

0. Executive Summary

In this assignment there is a need to determine the minimal interest rate, i, at which a person can invest \$1000 per month for 15 years to ensure retirement nest egg of \$1 million. In order to calculate i three root-finding algorithms, bisection method, Newton-Rapshon, and a combination, were programmed in C++.

It will be shown that even though the bisection method does provide solutions, it suffers from a slow error linear convergence. On the other hand, using Newton's method will show fast quadratic convergence, however, highly depends on the choosing initial starting point appropriately. A combination of these methods will combine the speed and convergence stability needed to solve the problem. In order to reassure the person of my calculations and show a comparison of each of the three algorithms, I will test several functions with known roots. Results will be shown. This was done by advice of my collegue.

From the calculations, it was found that an monthly interest rate, i, of approximately 1.5796708% under the IEEE standard of single precision. Therefore, to create this "nest egg," the investor will need to find a fund offering an annual interest rate of 18.95% compounded monthly. Unfortunately, as of today, retirement will be impossible due to investments, such as CDs/savings, only offering rates 3-4%.

1. Statement of Problem

Determine the interest rate which will satisfy the following equation:

$$A = \frac{P}{i}\left[(1+i)^n - 1\right], \tag{1}$$

where $A \Rightarrow$ amount in the account, $P \Rightarrow$ deposit amount at each month, and $i \Rightarrow$ interest rate per period.

For the "nest egg," $n = 15$ years (180 month since compounded monthly), $P = 1,000$ and $A = 1,000,000$.

To determine i, will find a root using the bisection method, Newton-Rapshon, and a combination of the two.

2. Description of the Mathematics

From lecture, the problem is to find a root for a function, $f(i)$:

$$f(i) = A - \frac{P}{i}\left[(1+i)^n - 1\right], \quad i \neq 0 \tag{2}$$

such that it solves $f(i) = 0$. The root exists by the Intermediate value theorem, if $f(i)$ is a continuous function on an closed interval, [a,b], and $f(a)f(b) < 0$. Then there $\exists$ an i such that it satisfies, (2), $f(i) = 0$ for some $i \in [a, b]$.

Bisection method will repeatedly check for an i, by making an initial guess such that $i = \frac{a+b}{2}$. Newton's method will use functional values to determine the i which satisfies $f(i) = 0$. If i is an approximation to $\hat{i}$ satisfying $f(\hat{i}) = 0$, then by using simplification on the Taylor series expansion on the function (2) there exists a better approximation for $\hat{i}$ such that

$$\hat{i} = i_n - \frac{f(i_n)}{f\,'(i_n)}, \quad \text{where} \tag{3}$$

$$f\,'(i) = \frac{Pni(1+n)^{n-1} - P(1+i)^n + 1}{i^2} \tag{4}$$

3. **Description of the Algorithm**

Bisection Method

If trying to determine a root in a closed interval [a,b],
denote $AbsTol \Rightarrow$ absolute tolerance and $RelTol \Rightarrow$ denote the relative tolerence.
From lecture, will determine a root for a sufficent number of steps N, such that

$$N = log_2 \left[\frac{b-a}{AbsTol + \left| \frac{a+b}{2} \right| RelTol} \right] + 1 \tag{5}$$

Then the bisection algothrim can be set up as follows:

$$\begin{aligned} For \quad i \quad &= 1 \text{ to } N \\ &p = a + \frac{b-a}{2} \\ &\text{if } \frac{b-a}{2} \leq AbsTol + |p| \cdot RelTol \\ &\quad \text{if } \text{sign}(f(a)) \cdot \text{sign}(f(b)) > 0 \\ &\quad\quad b = p \\ &\quad \text{else} \\ &\quad\quad a = p \end{aligned}$$

Newton's Method From lecture, the algothrim can be improved as follows. Let $\alpha \approx .01$

$$\begin{aligned} For \quad i \quad &= 1 \text{ to } N \\ &\delta = -\frac{f(x)}{f\,'(x)} \\ &\text{if } |\delta| \leq AbsTol + |x| \cdot RelTol \\ &\quad \text{if } |\delta| > \alpha |x| \text{ then} \\ &\quad\quad x = x + |x| \alpha \text{sign}(\delta) \\ &\quad \text{else} \\ &\quad\quad x = x + \delta \end{aligned}$$

Combination: Bisection and Newton

In this method, will use the bisection method first to assure convergence stability until within obtain a value for estimate within one significant digit. Since error doubles for every power of 10, the goal is to get to 10^{-13}. To obtain goal faster, switch to Netwon's method and after 5 or 6 iterations will obtain convergence to the solution.

4. **Results**

The following table will illustrate several functions in which the exact root(s) was either known or unknown (as in the case of the problem at hand). The roots were determined using each of the three methods, Bisection, Newton, and a combination method.

Functions	soln(s)	Bisection soln	Newton soln	Combination soln
$y = x - 1$	$1.00E+00$	$1.00E+00$	$1.00E+00$	$1.00E+00$
$y = x^2 - 1$	$1.00E+00$	$1.00E+00$	$1.00E+00$	$1.00E+00$
$y = e^x - e$	$1.00E+00$	$1.00E+00$	$1.00E+00$	$1.00E+00$
$y = \log(x)$	$1.00E+00$	$1.00E+00$	$1.00E+00$	$1.00E+00$
$y = \sin(x - 1)$	$1.00E+00$	$1.00E+00$	4.1415925	$1.00E+00$
rate prob $\rightarrow$ (2)	0.015796708	0.015796708	0.015796708	0.015796708

The relative error **(RE)** using each method was determined,

Functions	Bisection **(RE)**	Newton **(RE)**	Combination **(RE)**
$y = x - 1$	$0.00E+00$	$0.00E+00$	$0.00E+00$
$y = x^2 - 1$	$5.96E-08$	$1.19E-07$	$0.00E+00$
$y = e^x - e$	$0.00E+00$	$0.00E+00$	$0.00E+00$
$y = \log(x)$	$5.96E-08$	$0.00E+00$	$0.00E+00$
$y = \sin(x - 1)$	$0.00E+00$	3.14	$0.00E+00$
rate prob $\rightarrow$ (2)	$0.00E+00$	$0.00E+00$	$0.00E+00$

The number of iterations **(IT)** taken to solve the problem with each method is shown below.

Functions	Bisection **(IT)**	Newton **(IT)**	Combination **(IT)**
$y = x - 1$	26	1	10
$y = x^2 - 1$	26	6	11
$y = e^x - e$	26	8	11
$y = \log(x)$	23	5	8
$y = \sin(x - 1)$	24	5	10
rate prob $\rightarrow$ (2)	25	5	11

Assumptions of functions used for tests

- Each function was chosen to have a root at $x = 1$.
  Note: Trigonemtric function is periodic, thus multiple roots
- Each function satisifies conditions for Theorems $\Longrightarrow$ Continuity
- To achieve IEEE single precison, the relative tolerance (RelTol) was set to single point machine epsilion.

5. Conclusion

Anaylsis of Bisection Method

From lecture, the bisection method descreases error by a factor of 2. I can calculate the number of iterations that are needed for obtaining the IEEE standard.

$$\frac{|e_{n+1}|}{|e_n|} = \frac{1}{2} \implies |e_{n+1}| = \left(\frac{1}{2}\right)^n |e_n| \tag{6}$$

Setting e_n= machine epsilon to approximate the solution to single precision. In (6), can solve for n by applying the following:

$$n = \frac{\log\left(\frac{e_0}{e_n}\right)}{\log 2} \approx \frac{\log\left(\frac{10}{10^{-7}}\right)}{\log 2} = 26.6 \tag{7}$$

From the results, each bisection method took between 23-26 iterations to approximate a solution. This corroborate with the estimate in (7), however, my overestimate can still not be determined. It may be due to my choice in a starting error which is less than 10. I also choose a myriad of functions, which shows that determining the solution is independent of the number interations taken to derive the solution. This may because the only information that this method uses is the sign of the values.

Anaylsis of Newton Method

From lecture, Newton's method has quadratic convergence as follows:

$$\frac{|e_{n+1}|}{|e_n|^2} = \frac{1}{2}|g''(\xi)| \implies |e_{n+1}| = \left(\frac{|g''(\xi)|}{2}\right) e_n^2 \tag{8}$$

Newton's method is dependent on the initial values chosen, as illustrated in lecture. However, if chosen correctly and once the error approached one sigfig, converges at a quadratic speed.

Anaylsis of Combination

The combination method seems slower than Newton's method as seen in the number of iterations. However, Netwon's method may lose the quadratic speed if the intial values chosen are not "close enough" to an estimated solution. Bisection method will obtain a better intial values to start up the alogorithm and Netwon's method will obtian the solution faster.

Homework problem

I tested the program using functions with known roots. The results show that I obtained the solution with a certain amount of error (RelTol) and a certain number of iterations. The monthly interest rate, i, is estimated about 1.5796708%. By multiplying by the number of months (12), the annual monthly interest rate is 18.95605%. You cannot reach retirement with this little monthly deposit, P at today's rates. This is why I stop teaching high school.

APPENDIX: 1 Main file

```
// HW3.cpp
//Jaime Frade
//need to open excel file to store
//values.


#include <iostream>
#include <cmath>
#include "float.h"
#include "rootprog.h"
#include "testfunc.h"
#include "stdio.h"

// calculate an estimate for the relative error in calculation

float estimate (float exact_sol, float est_sol)
{
return fabs(exact_sol-est_sol)/fabs(exact_sol);
}

void reporting(FILE *fp, float exact_sol,float root_bi,float root_nt,float root_combo,
int iter_bi,int iter_nt, int iter_combo)
{
float error_bi;
float error_nt;
float error_combo;

error_bi=estimate(exact_sol,root_bi);
error_nt=estimate(exact_sol,root_nt);
error_combo=estimate(exact_sol,root_combo);

fprintf(fp,"%15.7e %15.7e %15.7e %15.7e %8.2e %8.2e %8.2e %6u %6u %6u \n",
exact_sol,root_bi,root_nt,root_combo,error_bi,error_nt,error_combo, iter_bi,iter_nt,iter_combo);

}

void main()
{
float a;
float b;
float startvalue;
float AbsTol;
float RelTol;
float exact_sol;

float root_bi;
float root_nt;
float root_combo;
int iter_bi;
int iter_nt;
int iter_combo;
```

```
FILE *fp;
fp=fopen("HW3.xls","w"); //prints output in excel file


AbsTol=FLT_MIN;
RelTol=FLT_EPSILON;

//  Test Case 1: Linear Function

a=0.1;
b=10.0;
startvalue=5.0;
exact_sol=1.0;
root_bi=bisect(f1,a,b,AbsTol,RelTol,iter_bi);
root_nt=newt(f1,fp1,startvalue,AbsTol,RelTol,iter_nt);
root_combo=combo(f1,fp1,a,b,AbsTol,RelTol,iter_combo);
reporting(fp,exact_sol,root_bi,root_nt,root_combo,iter_bi,iter_nt,iter_combo);

//  Test Case 2: Quadratic Function

a=0.1;
b=10.0e5;
startvalue=10.0e5;
exact_sol=1.0;
root_bi=bisect(f2,a,b,AbsTol,RelTol,iter_bi);
root_nt=newt(f2,fp2,startvalue,AbsTol,RelTol,iter_nt);
root_combo=combo(f2,fp2,a,b,AbsTol,RelTol,iter_combo);

reporting(fp,exact_sol,root_bi,root_nt,root_combo,iter_bi,iter_nt,iter_combo);

//  Test Case 3: Exponential Function

a=0.1;
b=10.0;
startvalue=5.0;
exact_sol=1.0;
root_bi=bisect(f3,a,b,AbsTol,RelTol,iter_bi);
root_nt=newt(f3,fp3,startvalue,AbsTol,RelTol,iter_nt);
root_combo=combo(f3,fp3,a,b,AbsTol,RelTol,iter_combo);
reporting(fp,exact_sol,root_bi,root_nt,root_combo,iter_bi,iter_nt,iter_combo);

//  Test Case 4: Log Function

a=0.1;
b=1.0;
startvalue=2.0;
exact_sol=1.0;
root_bi=bisect(f4,a,b,AbsTol,RelTol,iter_bi);
root_nt=newt(f4,fp4,startvalue,AbsTol,RelTol,iter_nt);
root_combo=combo(f4,fp4,a,b,AbsTol,RelTol,iter_combo);
reporting(fp,exact_sol,root_bi,root_nt,root_combo,iter_bi,iter_nt,iter_combo);
```

```
//  Test Case 5: Cosine Function (Multiple Solutions)

a=0.1;
b=5.0;
startvalue=3.0;
exact_sol=1.0;
root_bi=bisect(f5,a,b,AbsTol,RelTol,iter_bi);
root_nt=newt(f5,fp5,startvalue,AbsTol,RelTol,iter_nt);
root_combo=combo(f5,fp5,a,b,AbsTol,RelTol,iter_combo);
reporting(fp,exact_sol,root_bi,root_nt,root_combo,iter_bi,iter_nt,iter_combo);


//  Test Case 6: Interest rate Problem Function

a=FLT_MIN;
b=0.1;
startvalue=0.01;
root_bi=bisect(frate,a,b,AbsTol,RelTol,iter_bi);
root_nt=newt(frate,fprate,startvalue,AbsTol,RelTol,iter_nt);
root_combo=combo(frate,fprate,a,b,AbsTol,RelTol,iter_combo);
exact_sol=root_bi;
reporting(fp,exact_sol,root_bi,root_nt,root_combo,iter_bi,iter_nt,iter_combo);
fclose(fp);

}
```

APPENDIX: 2 Root-finding.h

```
# include "stdio.h"


//Need to derive a function that returns
//the sign of a number,
// number is postive = 1
// number is negative = -1
//number is zero = 0
//Not needed in FORTRAN


int sign (float x)
{
int sign;
if (x > 0.0)
sign=1;
else if (x<0.0)
sign=-1;
else
sign=0;
return sign;
}

//Bisection algothrim (as noted in lecture)



float bisect(float (*f)(float), float a, float b,
float AbsTol, float RelTol, int & iteriations)

{
int N; //iteriations stop here
int i;
int beta=20;
float p;
float fa;
float fp;


N = fabs(b-a)/(AbsTol+RelTol*(a+b)/2.0);
N = log10(N)/log10(2)*beta;

for (i=1;i<=N;i++)
{

p=a+(b-a)/2.0;

if (((b-a)/2.0 <= AbsTol+fabs(p)*RelTol) )
{
iteriations=i-1;
break;
```

```
}

fa=f(a);
fp=f(p);

if (sign(fa)*sign(fp)>0.0)
a=p;
else
b=p;

}

return p;

}


//Newton-Rapshon Method.  algothrim from lecture


float newt (float (*f)(float), float (*fp)(float), float startvalue,
  float AbsTol, float RelTol, int & iteriation)

{
int N;
int i;
float p;
float dp;

N = 1e5;  //iteriations stop here
p=startvalue;

for (i=1;i<=N;i++)
{

dp=-f(p)/fp(p);
if ( fabs(dp) < AbsTol+fabs(p)*RelTol )
{
iteriation=i-1;
break;
}
p=p+dp;

}

return p;

}

//Combo: (besection and Netwon)  Within solution of (alpha) = .01

float combo (float (*f)(float), float (*fp)(float), float a, float b,
  float AbsTol, float RelTol, int &iter)
```

```
{
float first_RelTol=0.01;
float first_sol;
int first_iter;

first_sol=bisect(f,a,b,AbsTol,first_RelTol,first_iter);
first_sol=newt(f,fp,first_sol,AbsTol,RelTol,iter);
iter=iter+first_iter;

return first_sol;

}
```

APPENDIX: 3 Root-finding.cpp

```
#include <cmath>

//bisection method


float bisect(float (*f)(float), float a, float b, float AbsTol, float RelTol)

{
int N;
int i;
float p;
float fa;
float fb;

N = fabs(b-a)/(AbsTol+RelTol*(a+b)/2.0);
N = log10(N)/log10(2);

for (i=1;i<=N;i++)
{

p=a+(b-a)/2.0;

if ((b-a)/2.0 <= AbsTol+fabs(p)*RelTol)
break;

fa=f(a);
fb=f(b);

if (fa*fb>0)
a=p;
else
b=p;

}

return p;

}


//Newton Method

float newt(float (*f)(float), float (*fp)(float), float startvalue, float AbsTol, float RelTol)

{
int N;
int i;
float p;
float dp;

N = 1e10;
```

```
p=startvalue;

i=0;
do
{
i=i+1;
dp=-f(p)/fp(p);
p=p+dp;

} while (( fabs(dp) > AbsTol+fabs(p)*RelTol ) && i<=N)

return p;

}
```

APPENDIX: 4 Functions

```
//Functions to be tested

//Case 1: Linear Function
float f1(float x)
{
float f;
f=x-1;
return f;

}

float fp1(float x)
{
float f;
f=1;
return f;
}

//Case 2: Quadratic

float f2(float x)
{
float f;
f=pow(x,2)-1.0;
return f;

}

float fp2(float x)
{
float f;
f=2*x;
return f;
}

// Case 3: Exponential

float f3(float x)
{
float f;
f=exp(x)-exp(1);
return f;

}

float fp3(float x)
{
float f;
f=exp(x);
return f;
}
```

```
// Case 4: Logarithmic

float f4(float x)
{
float f;
f=log(x);
return f;

}

float fp4(float x)
{
float f;
f=1.0/x;
return f;
}


// Case 5: Sine Trig function

float f5(float x)
{
float f;
f=sin(x-1);
return f;

}

float fp5(float x)
{
float f;
f=cos(x-1);
return f;
}


// Case 6: Interest Rate problem

float frate(float x)
{
float f;
int n;
float A;
float P;

n=12*15;
A=1000000;
P=1000;

f=A-P/x*(pow((1.0+x),n)-1.0);
return f;
```

```
}

float fprate(float x)
{
float f;
int n;
float A;
float P;

n=12*15;
A=1000000;
P=100000;

f=P/pow(x,2)*(pow((1.0+x),n)-1.0)-P/x*(n*pow((1.0+x),n-1));
return f;
}
```