UNITED STATES MILITARY ACADEMY

PROJECT 1

MA386: NUMERICAL ANALYSIS

SECTION G2

COL LINDQUIST

By

CADET EZRA HARRIS '22, CO E2

WEST POINT, NEW YORK

15 SEPTEMBER 2021

EH MY DOCUMENTATION IDENTIFIES ALL SOURCES USED AND ASSISTANCE RECEIVED IN COMPLETING THIS ASSIGNMENT.

_____ I DID NOT USE ANY SOURCES OR ASSISTANCE REQUIRING DOCUMENTATION IN COMPLETING THIS ASSIGNMENT.

SIGNATURE: Ezra Harris

Analyzing the Interest Rate Function With the Secant Method

Abstract

In this paper I analyze the interest rate function using the secant method. I compare and contrast the secant method with Newton's method and the bisection method, analyze the limits of the secant method, and provide a recommendation for the purchase of an RV given a set time frame. I discover that the secant method is good balance between ease of use and speed when calculating a desired interest rate.

1 Introduction

This project seeks to explore the utility of the secant method in finding the roots of various functions. I will provide a comparison of several other root finding methods. I will analyze how various initial conditions effect the outcome of the function. Finally, I will evaluate the strengths and weaknesses of the method as a numerical analysis technique.

The reason why applying this method is important is because it is a powerful tool for finding the value of the independent variable when the desired value is know. In this project we analyze the interest rate function as well as two toy functions to validate the method and demonstrate it's applicability to real world problem solving.

The interest rate function will be explored using the example of a major purchase being made sometime in the future. In this case COL Lindquist needs to save 200,000 dollars to purchase an RV before retirement. He will contribute 2,200 dollars a month and he has approximately 30 months to purchase the RV in time for his retirement. We are asked to find the interest rate required to purchase the RV.

2 Methods

In order to determine the interest rate required to meet COL Lindquist's goal we construct an interest rate function which has a root at the desired interest rate. This function is 200,000 subtracted from the superposition of the amount accrued from the compound interest of a principle investment and the amount accrued from regular periodic investments.

$$f(x) = P(1+i)^n + (\frac{d}{i})((1+i)^n - 1) - 200000 \tag{1}$$

Where p = principle investment, i = interest rate, d = periodic contributions, and n = the number of months.

In order to evaluate this function we use the secant method. The secant method is similar to newtons method except that it replaces the derivative with an approximation of the derivative by calculating the function at two points very close together. This gives us the proverbial rise over run and an approximation of the derivative. By calculating the root of the secant line and then replacing one of the fixed points with the evaluation of the function at that point the secant line can converge on the root of the function. A more thorough comparison of Newton's method and the secant method will be provided in the discussion section of this report.

First, we test our method using two toy functions.

$$f(x) = cos(x) \tag{2}$$

$$f(x) = 1 - cos(x) \tag{3}$$

Using the secant method we find the root of $cos(x)$ to be 1.5707963 and the root of $1 - cos(x)$ to be 0.0 within a tolerance of .00001. Evaluating the functions at these points does indeed reveal that they are

1.5707963 and 0 respectively. Given the limits of this test we opt to use a third test function that more closely resembles the interest rate function. This is to validate that the method works for polynomials and not just trigonometric functions

$$f(x) = x^2 - 10 \tag{4}$$

Using the secant method we find the root of this function to be 3.162277 again within a tolerance of .00001. Evaluating this function again reveals that it is 3.162277. Given the theoretical basis and three test cases we conclude that the method should work for the interest rate function.

3 Results and Analysis

Using the secant method we identified that given 32 months, COL Lindquist could buy his RV if he had a 1.455% monthly interest rate or a 17.46% yearly interest rate. The secant method calculated this value by first calculating the secant line of two points $f(p_0)$ and $f(p_1)$ on the function. It uses the evaluation of the function at the root of this secant line to replace the previous value at which the function was calculated. This brings the two points closer together. The method repeats until $p_0 - p_1$ is less than the tolerance. Because one point is the root of the secant line and the other point is on the function we know that the point on the function must be very close to zero.

Yearly Interest Rate	Fund value
1.2	143777.2613
2.4	147248.3864
3.6	150816.0422
4.8	154482.9706
6.0	158251.9906
7.2	162126.0003
8.4	166107.9795
9.6	170200.9912
10.8	174408.1845
12.0	178732.7968
13.2	183178.1560
14.4	187747.6832
15.6	192444.8954
16.8	197273.4077
18.0	202236.9360
19.2	207339.3004
20.4	212584.4273
21.6	217976.3524
22.8	223519.2243
24.0	229217.3066
25.2	235074.9818
26.4	241096.7543
27.6	247287.2535
28.8	253651.2375

Table 1:

I evaluated the success of the method by first calculating the value of the function at the interest rate, using a table of the various interest rates and the corresponding values of the fund, and finally by inspection of a

graph.

By inspection we can see that the desired fund value falls within the range of our calculated yearly interest rate. This is a promising indicator for the success of the method. As a second check we evaluate the function at the calculated interest rate. This yields a value of 199994.5188. Which gives us a percent error of .002. Finally, we inspect the root of the graph of the interest rate function which reveals that the root does in fact fall where we calculated.

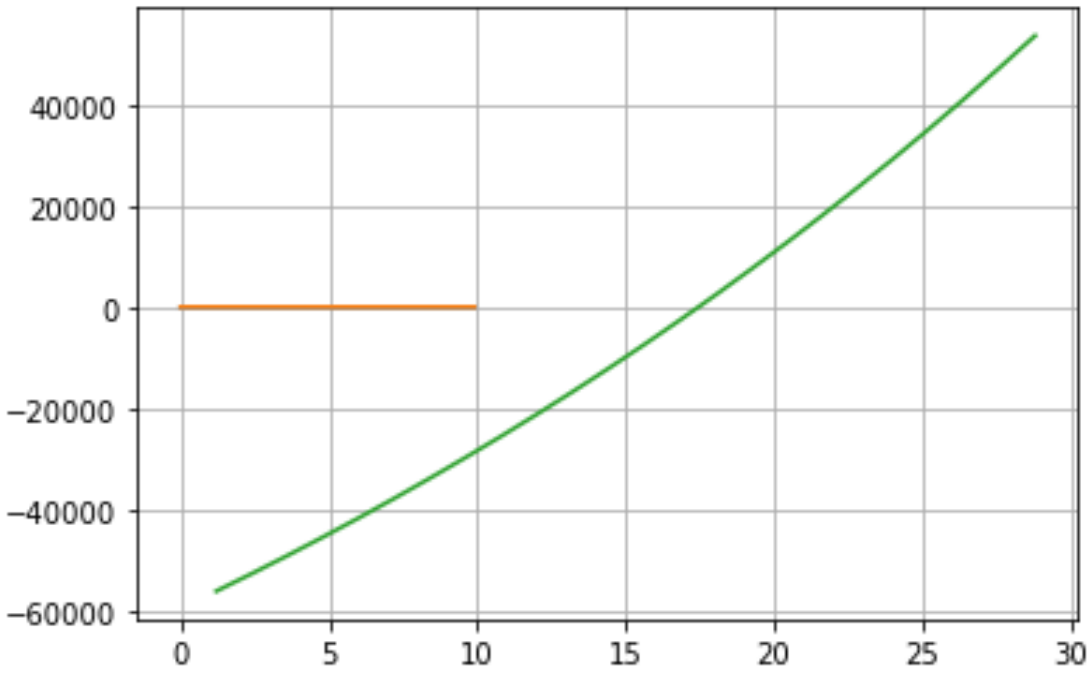

Figure 1: Roots of the interest rate function

Now I will offer a comparison of two other root finding methods. The first being Newton's method and the second being the bisection method. Newton's method is another form of fixed point iteration which relies on the calculation of the derivative rather than an evaluation of the function at two points. The main drawback is the calculation of the derivative of a function may not always be feasible and evaluating two separate functions for each iterations is computationally expensive. However given the initial condition of .001 Newton's method was able to calculate the root of the function in 4 iterations compared to the 7 the secant method used. Next we evaluate the bisection method. This method converges linearly and is thus much slower. When evaluating this function using the bisection method it took 34 iterations compared to 7 when using the secant method. The benefits of using bisection is that it will always converge given the root has been captured by the selected bounds.

Finally, I will discuss the limits of the secant method. One limit is when the slope of the secant line is horizontal. The function will not iterate because there is either no root of the secant line or there is an infinite number of roots of the secant line. Another limit is if the secant line does not cross the x axis. Because the secant line does not capture the root the method will not iterate.

4 Conclusion

Demonstrated by my analysis COL Lindquist will be able to purchase his RV in a time frame of 32 months given an interest rate of 17.46%. This is conveniently almost exactly the 5 year return rate of the Vanguard 500 index fund. I recommend that COL Lindquist continues with his plan and considers increasing his monthly contribution as a buffer to protect against market dips. Increasing the month contribution by 200 dollars will allow for a 2% dip in fund performance. This is admittedly not a lot but is a safety margin I feel worthwhile to consider.

5 Appendix A: Secant method code

```
    import math
import matplotlib.pyplot as plt


def f(x):
    return math.cos(x)


def Secant(p0,p1,tol,maxIter):
    """


    Parameters
    ----------
    p0 : float
        Value to intialize secant method.
    p1 : float
        Second value to intialize secant method.
    tol : float
        The tolerance of the program when determining the root
        of the function vs the root of the secant line.
    maxIter : integer
        Maximum number of iterations the program will run before
        it gives up.

    Returns
    -------
    p : float
        This is the calculated value of the root of the function.
    i : integer
        the amount of times it took to get there.

    """

    i = 2
    q0=f(p0)
    q1=f(p1)

    while i < maxIter:
        p = p1-q1*((p1-p0)/(q1-q0))
        if abs(p-p1)<tol:
            return p,i
        i=i+1
        p0=p1
        q0=q1
        p1=p
        q1=f(p)

k,i =Secant(0,math.pi,.00001,100)
```

```
print("the root for cos(x) between 0 and Pi:", k, "and the iterations it took to get there:", i)

def g(x):
    return 1-math.cos(x)

def Secant(p0,p1,tol,maxIter):

    i = 2
    q0=g(p0)
    q1=g(p1)

    while i < maxIter:
        p = p1-q1*((p1-p0)/(q1-q0))
        if abs(p-p1)<tol:
            return p,i
        i=i+1
        p0=p1
        q0=q1
        p1=p
        q1=g(p)

k,i =Secant(0,math.pi,.00001,100)

print("the root for 1-cos(x) between 0 and Pi:", k, "and the iterations it took to get there:", i)

P=70000
x=32
d= 2400

def h(j):
    return -200000+((P*(1+(j))**x))+(d/(j))*(((1+(j))**x)-1)

r=[]
s=[]
n=0.001
N0=.025
while n < N0:
    o=h(n)
    r.append(100*(12*n))
    s.append(o)
    n=n+.001

def Secant(p0,p1,tol,maxIter):

    i = 2
    q0=h(p0)
    q1=h(p1)

    while i < maxIter:
        p = p1-q1*((p1-p0)/(q1-q0))
        if abs(p-p1)<tol:
            return p,i
```

```
        i=i+1
        p0=p1
        q0=q1
        p1=p
        q1=h(p)

k,i =Secant(.01,.2,.00001,100)

print("the interest rate required for COL Lindquist to purchase his RV:", k, "and the iterations it took

def w(x):
    return x**2-10

def Secant(p0,p1,tol,maxIter):

    i = 2
    q0=g(p0)
    q1=g(p1)

    while i < maxIter:
        p = p1-q1*((p1-p0)/(q1-q0))
        if abs(p-p1)<tol:
            return p,i
        i=i+1
        p0=p1
        q0=q1
        p1=p
        q1=w(p)

k,i =Secant(5,10,.00001,100)

#print("the root for x^2-1 between 0 and 10:", k, "and the iterations it took to get there:", i)

plt.plot(r,s)
plt.grid()
```

6 Appendix B: Newtons Method

```
    import math
P=70000
x=32
d= 2200
def h(j):
    return -200000+((P*(1+(j))**x))+(d/(j))*(((1+(j))**x)-1)

def g(j):
    return 2240000*(1 + j)**31 + (70400*(1 + j)**31)/j - (2200*(-1 + (1 + j)**32))/j**2

def newton(p0,tol,maxIter):

    """
    Created on Tue Aug 31 07:37:29 2021
    The function newton(p0,tol,maxIter) is designed to find the root
    of a function using newtons method.

    Inputs: p0 is our intial guess for te root
            tol is the desired tolerance between successive approx.
            maxIter is the max number of iterations to try

    Outputs: p - the best guess for the root
             iter - how many iteration s it took to get there

    @author: Ezra
    """

    iter=1
    while iter < maxIter:
        p = p0 - h(p0)/g(p0)
        if abs(p-p0) < tol:
            return p, iter
        iter = iter+1
        p0=p

p,iter = newton(0.001,.000001,100)
print("Root is:",p,"Number of iterations =", iter)
```

7 Appendix C: Bisection Method

```
    import math
import matplotlib.pyplot as plt

def bisection(left, right, tol, numIter):

    i = 1
    fa= h(left)
    err = []

    while i <= numIter:
        p= left+(right-left)/2
        fp= h(p)
        if (fp == 0 or abs(fp) <tol):
            break
        i=i+1
        if fa*fp > 0:
            left = p
            fa=fp
        else:
            right = p

    return p,fp,err,i

P=70000
x=32
d= 2200

def h(j):
    return -200000+((P*(1+(j))**x))+(d/(j))*(((1+(j))**x)-1)
# def f(x):
#     return(3*x - math.exp(x))

p,fp,err,i = bisection(0.01,.2,0.00001,50)
print(p, i)
```

8 References

CDT Oscar Morales. Company F2. Assistance given through verbal instruction. Corrected an instance of implicit multiplication when I coded my interest rate formula. West Point, NY. 14 Sep 21.

Compound Interest. Wikipedia. *https* : *//en.wikipedia.org/wiki/Compound*$_i$*nterest* Included $\frac{i}{n}$ term. Before I read the article I was confused as to why my interest rate was so low. I then realized that I was calculating the monthly interest rate and had to multiply by 12. Accessed 14 Sep 2021.