Matthew Lawson

MATH 4500: Numerical Analysis

Professor Lin Mu

Numerical Solutions to Financial Annuities

Annuities play a central role in both theoretical and applied finance, particularly in modeling retirement savings, loans, and insurance payouts. In mathematical terms, an annuity is a stream of fixed payments made at regular intervals, whose value either accumulates or is drawn down over time, with interest. While many textbook cases assume fixed rates and straightforward computations, real-world annuities often lead to nonlinear equations that cannot be solved analytically. This paper explores three such annuity-based financial problems that require numerical root-finding techniques to determine the appropriate interest rates.

To solve these problems, Python is used to implement several numerical methods, including the Secant, Bisection, and Newton's methods. Through convergence tables, residual plots, and function graphs, the performance of each method is evaluated. The overarching goal is to find monthly or annual interest rates that satisfy given annuity conditions and assess the long-term financial outcomes.

Problem 1:

Problem 1 introduces the idea of the simple annuity as a "fund of money to which payments are made at regular intervals". In this "simple" annuity, the money accumulated in the fund is compounded at an interest rate *r*, at a defined schedule. At the end of each defined payment interval, a payment, a_i, is made to the fund. This payment is added at the end of period *i.* In practice, the timing of payments and compounding may not match. For the case of the problems discussed in this paper, I assume that compounding always occurs at the same frequency as payments (weekly, monthly, annually, etc.). The following recursive definition is provided for the value, *V*, of a simple annuity at any time *i:*

$$V_1 = a_1 \qquad V_i = V_{i-1}(1 + r) + a_i, \ (i = 2, 3, \ldots)$$

The problem then asks to prove the following generalized formula for the value of an annuity at any time *n:*

$$\sum_{i=1}^{n} a_i x^{(n-i)}$$

To address this proof two methods were identified, a term expansion approach and a direct induction proof. By expanding the first four terms as defined by the recursive definition:

$$V_1 = a_1$$

$$V_2 = V_1 x + a_2 = a_1 x + a_2$$

$$V_3 = V_2x + a_3 = (a_1x + a_2)x + a_3 = a_1x^2 + a_2x + a_3$$

$$V_4 = V_3x + a_4 = (a_1x^2 + a_2x + a_3)x + a_4 = a_1x^3 + a_2x^2 + a_3x + a_4$$

The above pattern gives the following annuity value, V_n, after n payments:

$$V_n = a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n = \sum_{i=1}^{n} a_ix^{n-i}$$

The induction proof follows similarly:

We are given the following recurrence relation for the value of annuity, V_i, with payments $a_1, a_2, \ldots, a_n$, and interest compounded at a fixed rate r per interval:

$$V_1 = a_1, \quad \text{and} \quad V_i = V_{i-1} \cdot x + a_i, \quad \text{for } i = 2, 3, \ldots, n$$

where $x = 1 + r$. We seek to show the value of the annuity after n payments is:

$$V_n = \sum_{i=1}^{n} a_ix^{n-i}, \quad \text{for all } n \in \mathbb{N}$$

We proceed by induction on n. Let $V_1 = a_1$, and $V_i = V_{i-1}x + a_i$ for $i \geq 2$. Let

$$S = \{n \in \mathbb{N} : V_i = \sum_{i=1}^{n} a_ix^{n-i}\}$$

Base Case: First, consider $n = 1$. By definition, $V_1 = a_1$
The formula gives:

$$\sum_{i=1}^{1} a_ix^{1-i} = a_1x^0 = a_1$$

So $1 \in S$, the base case is true.

Inductive Step: Suppose $n \in S$. Meaning that:

$$V_n = \sum_{i=1}^{n} a_ix^{n-i}$$

Then,

$$V_{n+1} = V_n \cdot x + a_{n+1}$$

Substituting into the formula for V_n:

$$V_{n+1} = \left(\sum_{i=1}^{n} a_ix^{n-i}\right) \cdot x + a_{n+1}$$

$$= \sum_{i=1}^{n} a_ix^{(n-i)+1} + a_{n+1} = \sum_{i=1}^{n} a_ix^{(n+1)-i} + a_{n+1}x^0 = \sum_{i=1}^{n+1} a_ix^{(n+1)-i}$$

Thus $n + 1 \in S$. By PMI (Principle of Mathematical Induction), $S = \mathbb{N}$.

The second part of Problem 1, I modeled a 5-year annuity with tiered monthly payments. The goal was to determine the monthly interest rate that would grow the total fund to $24,738 after 60 months. The monthly payments increased annually: $200 in year one, $275 in year two, $312 in year three, $380 in year four, and $400 in the final year.

The future value of the annuity is modeled as:

$$F(r) = \sum_{i=1}^{60} a_i (1 + r)^{60-i} - 24,738$$

where a_i is the payment amount in a month . This formulation effectively defines a root finding problem for the monthly interest rate r. The resulting function gives a high order polynomial that does not have an algebraic solution, numerical methods must be used to approximate the root.

The secant method was selected for use approximating the root for this problem. Below is the Python implementation for the secant method:

```python
import numpy as np
import matplotlib.pyplot as plt

payments = np.repeat([200, 275, 312, 380, 400], 12)
target_value = 24738

# f(r)
def f(r):
    x = 1 + r
    exponents = np.arange(len(payments)-1, -1, -1)
    return np.sum(payments * x**exponents) - target_value

# Secant method (altered to track iteration history for convergence table)
def secant_method(f, r0, r1, tol=1e-8, max_iter=100):
    history = [(0, r0, f(r0)), (1, r1, f(r1))]
    f0, f1 = history[0][2], history[1][2]
    for k in range(2, max_iter+1):
        r2 = r1 - f1 * (r1 - r0) / (f1 - f0)
        f2 = f(r2)
        history.append((k, r2, f2))
        if abs(r2 - r1) < tol:
            break
        r0, f0, r1, f1 = r1, f1, r2, f2
    return history[-1][1], history
```

Notice that the function f(r) is defined using numpy to sum the individual payments corresponding to their effective future compound factor. This factor is simply the interest rate compounded the number of times required for that specific payment defined by its location in the payments array. The secant method itself is a simply implementation of the secant method formula. Line 18 contains the actual secant method formula itself. An additional history variable was used to store iteration variables for use generating the convergence table.

The method converged in 7 iterations, yielding , or approximately 1.0188% per month. The convergence was superlinear, which is evident in the convergence table in Figure **1**. Function plots and convergence tables confirmed the accuracy of this result. These are included as Figures **2** and **1**, respectively.

Problem 2:

Problem 2 expands on the idea of the simple annuity found in Problem 1. Here, two strategies over 44 years are provided that introduce the idea of payment scheduling and its effect on annuity valuation. Strategy A involves $1,000 payments to a simple annuity for the first six years of the investment period. Strategy B makes no payments for the first six years, but invests $1,000 every year for the following 38. The problem identifies that these two strategies are worth equal amounts after 44 years, and asks for the interest rate for that statement to be true, along with the value of each of the accounts at that time.

Here the value of those two series of payments can be modeled as follows:

$$V_A = 1000\sum_{k=1}^{6}(1+r)^{44-k} = 1000\sum_{j=38}^{43}(1+r)^{j} = 1000\,\frac{(1+r)^{44}-(1+r)^{38}}{r}$$

$$V_B = 1000\sum_{k=7}^{44}(1+r)^{44-k} = 1000\sum_{m=0}^{37}(1+r)^{m} = 1000\,\frac{(1+r)^{38}-1}{r}$$

Since the payments are constant at $1,000, the value of the annuities can be modeled as finite geometric series corresponding to the number of times each individual payment is compounded in the investment period (44 years). By setting the strategies equal to one another and using the the finite geometric series formula, these can be further reduced to:

$$\frac{(1+r)^{44}-(1+r)^{38}}{r} = \frac{(1+r)^{38}-1}{r}$$

$$(1+r)^{44} - 2\,(1+r)^{38} + 1 = 0$$

$$x^{44} - 2x^{38} + 1 = 0$$

Here the variable x = 1+ r to give the compound factor. This gives a high order polynomial which can be used with root finding methods to approximate the required *r* for the statement to be true.

The Bisection and Newton's methods were selected to approximate the required interest rate for this problem. The implementations were fairly basic and Below are the implementations with slight adjustments to account for the convergence table outputs (full code provided at end of report):

```python
# f2(r) and derivative funcs
def f2(r):
    x = 1 + r
    return x**44 - 2*x**38 + 1

def f2prime(r):
    x = 1 + r
    return 44 * x**43 - 76 * x**37

# Newton's Method
def newton_method(func, dfunc, r0, tol=1e-8, max_iter=100):
    history = [(0, r0, func(r0))]
    r = r0
    for k in range(1, max_iter+1):
        f_val = func(r)
        df_val = dfunc(r)
        r_next = r - f_val / df_val
        history.append((k, r_next, func(r_next)))
        if abs(r_next - r) < tol:
            break
        r = r_next
    return r_next, history

```

```python
def bisection_method(f, a, b, tol=1e-8, max_iter=100):
    # Check if f(a) and f(b) have opposite signs
    if f(a) * f(b) >= 0:
        raise ValueError("f(a) and f(b) must have opposite signs")

    history = []
    iterations = 0
    print("\nBisection Method Convergece:")
    print("Iter | a          | b          | c          | f(c)")
    print("-" * 60)

    while (b - a) / 2 > tol and iterations < max_iter:
        c = (a + b) / 2
        fc = f(c)
        history.append((iterations, c, fc))
        print(f"{iterations:4d} | {a:.8f} | {b:.8f} | {c:.8f} | {fc:.6e}")

        if fc == 0 or (b - a) / 2 < tol:
            break
        elif f(a) * fc < 0:
            b = c
        else:
            a = c
        iterations += 1
    return c, history

# Run Bisection
r_bis, hist_bis = bisection_method(f2, 0.05, 0.20)
print(f"\nFinal root:")
print(f"Bisection ≈ {r_bis:.10f}")

```

✓ 0.0s

Both methods found an approximate annual interest rate of 12.125% which corresponded with a final total value for each annuity of $630, 035.51. As is evident by the convergence table outputs in Figures **3** and **4,** the bisection method converges linearly as expected with a total number of iterations of 22. The Newton's Method converged roughly quadratically for the final few iterations as it approached the root, and took 11 total iterations. The relatively large number of iterations for both methods can be attributed to poor initial guesses. This is evident particularly in the Newton's method convergence table as the method was particularly slow to converge for the first few iterations. In the future, a combination of both Bisection and Newton's method would work best, providing a more accurate initial guess for Newton's to start at.

Problem 3:

Problem 3 introduces the concept of a payout annuity, in which the fund starts with a lump sum of money and withdrawals are made at the end of each period. The lump sum (including the withdrawal) accrues interest over the period, and the payout is made at the end of the period. The problem asks for a derivation for the formula for this type of annuity. I proceed as follows:

Similarly to Problem 1, we can construct the following recurrence relation defined by a payout annuity:

$$V_i = V_{i-1} \cdot x - a_i \qquad V_0 = \text{initial value} \qquad \text{for } i = 1, 2, 3, \ldots$$

Again, generating the first few terms:

$$V_1 = V_0 \cdot x - a_1$$

$$V_2 = V_1 \cdot x - a_2 = (V_0 x - a_1)x - a_2 = V_0 x^2 - a_1 x - a_2$$

$$V_3 = V_2 \cdot x - a_3 = (V_0 x^2 - a_1 x - a_2)x - a_3 = V_0 x^3 - a_1 x^2 - a_2 x - a_3$$

The above pattern gives the following formula for the value of a payout annuity, V_n, after n payments:

$$V_n = V_0 x^n - a_1 x^{n-1} - a_2 x^{n-2} - \ldots - a_n$$

$$= V_0 x^n - \sum_{i=1}^{n} a_i x^{n-i}$$

Notice, this is almost the same formula found in Problem 1 only that here the annuity begins with its maximum value and should decrease over time. This assumes the withdrawals exceed the generated interest.

Here, the final formula reduces to effectively the total lump sum compounded for all periods, less the individual withdrawals compounded for only the number of periods they are part of the lump sum.

The second part of Problem 3 proposes a scenario which resembles the typical approach to retirement savings. For a total of 40 years, $1,000 monthly payments are made that also compound monthly. At the end of the 40 years, $5,000 is withdrawn monthly for the following 20 years. In order to solve this problem, it is important to notice that the value of the annuity at the end of year 40, must equal the initial value of the payout annuity at this point in time as well. To construct this payout annuity form, I discount back each $5,000 withdrawal to year 40. This gives a future (although discounted back) value of the payout annuity at year 40 defined as:

$$V_{\text{pv}} = 5000 \sum_{\ell=1}^{240} (1+r)^{-\ell} = 5000 \sum_{\ell=1}^{240} v^{\ell} = 5000 \, \frac{1-(1+r)^{-240}}{r}$$

Using the same process defined in problems 1 and 2 for the simple annuity, simplifying the two annuity values and setting them equal to each other gives the following function:

$$1000 \, \frac{(1+r)^{480}-1}{r} \; = \; 5000 \, \frac{1-(1+r)^{-240}}{r}.$$

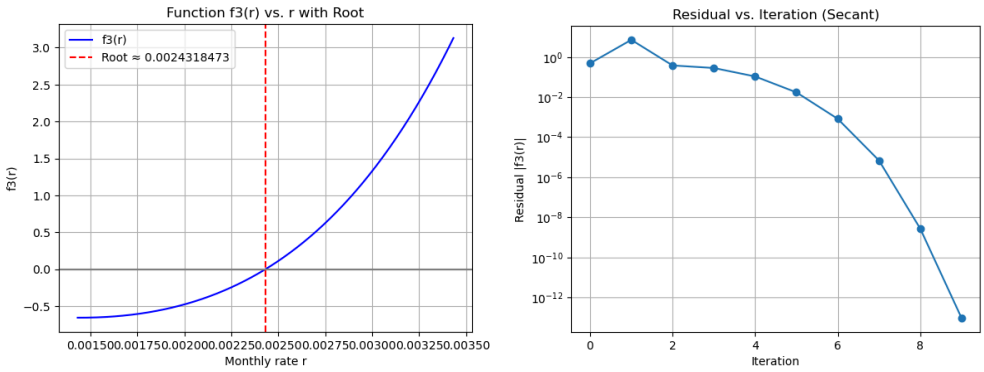
$$x^{720} - 6\,x^{240} + 5 = 0$$

This function can now be used to solve the root finding problem for the interest rate *r*.

The Secant method was identified for this problem as it performed slightly better based on the number of iterations than both Bisection and Newton's for the previous high order polynomials. Although Newton's converges quadratically which is faster than Secant, it performed worse likely due to the behavior of high order polynomials. Given that a wide initial range was selected to search for the root, Secant was selected to balance this efficiency tradeoff. The same secant method implementation was used as in Problem 1.

The Secant method produced a root at r = 0.0024318473 which equates to a roughly 0.24% monthly interest rate. As requested, this figure was annualized for a total effective annual

interest rate for this scenario of 2.957566%. Included below are two simple graphs of the used function and the plot of the residuals



I used three numerical root finding methods in this project: Secant, Newton's, and Bisection. While Newton's method offers the fastest theoretical convergence close to the root (quadratic), it was somewhat sensitive to initial guesses in this instance. Bisection was the most robust, but took a significantly larger number of total iterations to converge to a root. Secant offered a solid balance with superlinear convergence and no reliance on derivatives. From an applied finance perspective, I mathematically defined the value of various annuities. The value of these annuities varied based on the total and timing of the payments as well as the interest rate and compounding frequency. This project strengthened my numerical analysis skill set and enhanced my ability to mathematically formulate real-world problems. Moving forward, these skills provide a strong foundational knowledge that can be applied across a variety of domains and industries.

Figures

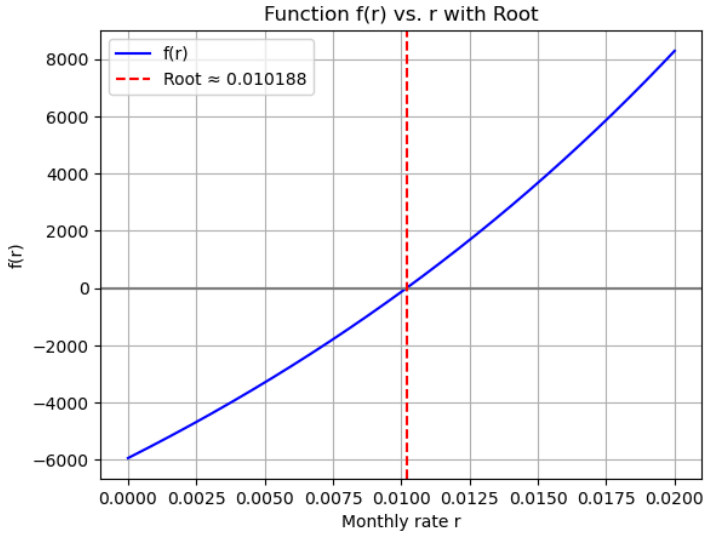
These are figures specifically referenced in the paper to account for space. All other code/proofs are attached.

1.

```
Secant Method Convergence Table:
Iter |            r |           f(r)
-----------------------------------
   0 | 0.0000000000 |  -5.934000e+03
   1 | 0.0200000000 |   8.284542e+03
   2 | 0.0083468475 |  -1.241962e+03
   3 | 0.0098660589 |  -2.232726e+02
   4 | 0.0101990340 |   7.763471e+00
   5 | 0.0101878451 |  -4.648093e-02
   6 | 0.0101879116 |  -9.604064e-06
   7 | 0.0101879117 |   2.182787e-11

Estimated monthly interest rate: 0.01018791 (1.0188%)
```

2.



3. Newton's Conv. Table

```
Newton's Method Convergence:
Iter |            r |          f2(r)
-------------------------------------
   0 | 0.1000000000 |  -7.544611e+00
   1 | 0.2138586608 |   1.894185e+03
   2 | 0.1913715374 |   6.678339e+02
   3 | 0.1707864758 |   2.312600e+02
   4 | 0.1526544305 |   7.733759e+01
   5 | 0.1378266344 |   2.397736e+01
   6 | 0.1274653775 |   6.164547e+00
   7 | 0.1224033363 |   9.494943e-01
   8 | 0.1212972205 |   3.720680e-02
   9 | 0.1212502293 |   6.450975e-05
  10 | 0.1212501475 |   1.950298e-10
  11 | 0.1212501475 |  -1.421085e-13

Final root:
Newtons ≈ 0.1212501475

Final root (annual rate): r = 0.1212501475
Total value after 44 years: $630,035.51
```

4. Bisection Conv Table

```
Bisection Method Convergece:
Iter | a          | b          | c          | f(c)
------------------------------------------------------------
   0 | 0.05000000 | 0.20000000 | 0.12500000 | 3.397420e+00
   1 | 0.05000000 | 0.12500000 | 0.08750000 | -7.379083e+00
   2 | 0.08750000 | 0.12500000 | 0.10625000 | -6.755510e+00
   3 | 0.10625000 | 0.12500000 | 0.11562500 | -3.602156e+00
   4 | 0.11562500 | 0.12500000 | 0.12031250 | -7.146158e-01
   5 | 0.12031250 | 0.12500000 | 0.12265625 | 1.168589e+00
   6 | 0.12031250 | 0.12265625 | 0.12148438 | 1.864198e-01
   7 | 0.12031250 | 0.12148438 | 0.12089844 | -2.739254e-01
   8 | 0.12089844 | 0.12148438 | 0.12119141 | -4.624826e-02
   9 | 0.12119141 | 0.12148438 | 0.12133789 | 6.945703e-02
  10 | 0.12119141 | 0.12133789 | 0.12126465 | 1.144781e-02
  11 | 0.12119141 | 0.12126465 | 0.12122803 | -1.743929e-02
  12 | 0.12122803 | 0.12126465 | 0.12124634 | -3.005514e-03
  13 | 0.12124634 | 0.12126465 | 0.12125549 | 4.218704e-03
  14 | 0.12124634 | 0.12125549 | 0.12125092 | 6.059840e-04
  15 | 0.12124634 | 0.12125092 | 0.12124863 | -1.199918e-03
  16 | 0.12124863 | 0.12125092 | 0.12124977 | -2.970051e-04
  17 | 0.12124977 | 0.12125092 | 0.12125034 | 1.544799e-04
  18 | 0.12124977 | 0.12125034 | 0.12125006 | -7.126500e-05
  19 | 0.12125006 | 0.12125034 | 0.12125020 | 4.160685e-05
  20 | 0.12125006 | 0.12125020 | 0.12125013 | -1.482923e-05
  21 | 0.12125013 | 0.12125020 | 0.12125016 | 1.338877e-05
  22 | 0.12125013 | 0.12125016 | 0.12125015 | -7.202356e-07

Final root:
Bisection ≈ 0.1212501466
```