A High-Performance Computing of Internal Rate of Return using a Centroid based Newton-Raphson Iterative Algorithm

for the Bachelor of Science Honours Degree in
Financial Mathematics and Industrial Statistics
MFM-3151

By
E.G.M.Lakruwan
SC/2020/11795

Supervisor:
Dr. Harshanie Lakshika Jayetileke

Department of Mathematics
University of Ruhuna
Matara.

2024

ACKNOWLEDGMENT

First of all, I would like to thank all of those who helped me in completing this report. Firstly, I would like to express my sincere gratitude to my supervisor, Dr. Harshanie Lakshika Jayetileke, for guiding me throughout this project and supporting me in finding and fixing my mistakes. Also, I'd like to thank all the other lecturers and the academic staff who helped me every step of the way to make this work a success. They offered their advice and help without hesitation whenever I needed it. Their expertise and advice were invaluable, and I am truly grateful to have such great mentors.

I would also like to thank my team members for all of their help and hard work. We were able to learn a lot together and help each other out. Also, all the friends who helped me in various ways, I am truly grateful that I had all their help which helped the succession of this report. I could not have completed it without you.
Thank you.

ABSTRACT

Financial analysts use Internal Rate of Return (IRR) to estimate how profitable an investment will be. This involves finding the discount rate that makes the present value of all future cash flows equal to the initial investment, in other words, where the net present value of the cash flow equal to zero. Knowing the IRR tells us when the project will "break even", where we can make profit.

This study focuses on finding highly accurate IRR values using a specific technique called the centroid method. Several methods exist for finding such values, but some may not work well for uneven cash flows or require good initial guesses. Most common method being the Newton Raphson method and it's modified and improver version being Midpoint-based Newton Raphson method. The modified midpoint-based method has high accuracy and high convergence rate, but it can be problametic in unequal cashflows, since it does not consider the distribution of the cashflow into account. The proposed centroid method tackles these issues, converging quickly with minimal error, even for uneven cash flows.

Through simulations, this study demonstrates that the centroid method outperforms the mid-point-based Newton-Raphson algorithm, making it a valuable tool for more accurate IRR calculations and better investment decisions.

Contents

List of Figures

1 Introduction

1.1 Background of the Study

In the realm of capital budgeting, the Internal Rate of Return (IRR) reigns supreme as a key metric for evaluating the potential profitability of investments. It essentially represents the discount rate that makes the net present value (NPV) of a project equal to zero. In simpler terms, IRR signifies the interest rate at which a project's cash inflows and outflows balance each other, effectively acting as a break-even point.

Determining the IRR directly, however, presents a formidable challenge due to its inherent nonlinear nature. This hurdle necessitates the use of iterative root-finding algorithms like secant, bisection, false position, and the ever-popular Newton-Raphson method to approximate the elusive IRR value. Among these contenders, the Newton-Raphson method stands out for its remarkable efficiency and rapid convergence, propelling it to the forefront of IRR estimation techniques. The Newton-Raphson algorithm is an iterative method employed to find the roots of nonlinear equations. Its underlying principle revolves around the concept of linearization. By approximating the function at the current guess using a tangent line, the algorithm generates a refined guess closer to the actual root.

1.2 Why is this Important

Even though it is faster and more efficient, the Newton-Raphson algorithm's effectiveness depends on the selection of an appropriate initial guess. A poorly chosen initial guess can lead to divergence, where the algorithm fails to converge to the root.

To address this issue, a midpoint-based Newton-Raphson technique has been proposed [10]. This method utilizes the midpoint of the cash flows as the initial guess, increasing the chances of convergence. Despite its advantages, the midpoint-based technique suffers from its static nature. It remains fixed regardless of the distribution of cash flows, potentially leading to inaccuracies, especially in cases of uneven cash flow patterns. This method can further be improved to increase the reliability when doing practical calculations which are more likely to be uneven cash flows.

1.3 Objectives

This study addresses the critical issue of initial guess selection by proposing a novel approach: the centroid-based Newton-Raphson algorithm. This proposed technique incorporates the weighted average of cash flows, known as the centroid, into the initial guess calculation. By dynamically adjusting the starting point based on the specific cash flow distribution of each project, the centroid-based algorithm aims to:

- Enhance the likelihood of convergence, ensuring the algorithm accurately finds the true IRR.
- Improve the overall accuracy of IRR estimation, leading to more reliable investment decisions.

- Reduce the number of iterations required to convergence.

Through simulations, this study compares the performance of the centroid-based algorithm against the traditional midpoint-based Newton-Raphson method, evaluating their accuracy and the number of iterations required to find the IRR. The findings aim to demonstrate the advantages of the centroid-based approach and its potential to improve the reliability and efficiency of IRR calculations in capital budgeting.

2 Literature Review

The rapid advancements in technology have significantly impacted on the financial sector, creating a plethora of investment opportunities. To make informed investment decisions and maximize profits, financial experts rely on various metrics to assess the viability of potential investments. One such popular tool is the Internal Rate of Return (IRR), which estimates the attractiveness of an investment by calculating the minimum rate at which the investment's net present value becomes zero [2]. However, calculating the IRR directly is not always straightforward since there is no analytical solution for that. While researchers have developed various techniques, including simplified closed-form approaches, these methods often lack accuracy.

One study [3] came up with a free n-point iterative technique of optimal order of convergence using rational interpolation. However, this technique required an initial guess of the true root, which could lead to non-convergence when the guessed value is too far from the actual value. A modified regula-falsi method (MRFM) was proposed [4] which was applied to photovoltaic applications and showed it has a faster convergence compared to other methods. However, MRFM can still cause non-convergence since it also required not one, but two initial guess inputs.

In the paper [5], an exponential interpolation and corrected secant formulas were introduced which achieved accurate results and can estimate the unknown. However, the formulas are complex for solving IRR. Given the complexity of solving function IRR, a fuzzy approach was proposed [6]. However, the result of the study could not be examined as the authors did not present the speed and accuracy of the said method. Another technique in determining IRR is by applying bisection and secant methods [7], however, both these algorithms are slow in converging, if they converge at all.

Prominent root-finding algorithms are bisection, false position, and secant, and Newton-Raphson method [7]. Among these methods, Newton-Raphson is the most preferred technique because of its quick convergence and accuracy [8]. However, this method also requires an initial guess which could lead to non-convergence and division by zero [9], [1].

The study of [10] tried to address this matter by generating the midpoint of cash flows and setting it as the initial guess of IRR. This approach was more effective in improving the speed and accuracy of the Newton-Raphson algorithm. In this research, we plan to further increase the accuracy of midpoint method using centroid method. This will also allow us to decrease the time needed to computation by decreasing the iterations needed until convergence.

3 Materials and Methods

This study exhibits the performance of the centroid-based Newton-Raphson algorithm (CNRA), a modified version of the midpoint-based Newton-Raphson algorithm (MNRA) [18], in real scenarios.

3.1 Existing Newton Raphson Algorithm

The study uses the below mentioned algorithms to analyze and implement the results.

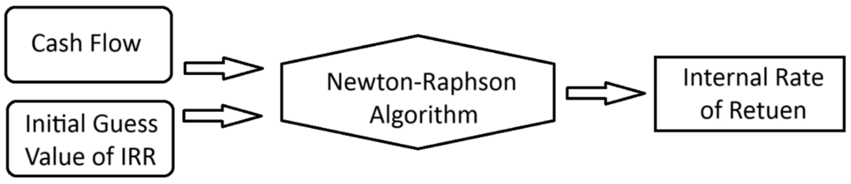


Figure 1: Generic Newton-Raphson algorithm for calculating IRR

A generic process of a Newton-Raphson algorithm in estimating IRR, as shown in Figure 1, would require an initial guessed IRR from the user and then employ the below equation (1) [13],

$$X_{i+1} = X_i - \frac{f(X_i)}{f'(X_i)} \tag{1}$$

MNRA simplified the process of calculating for the IRR, presented in Figure 2, by setting the initial guess IRR with the midpoint of cash flows instead of requiring input from the user [18].

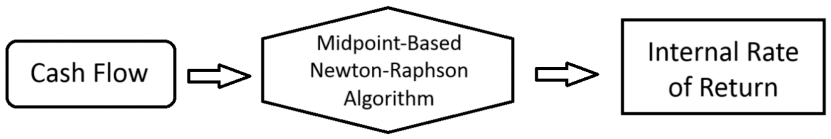


Figure 2: MNRA process for calculating IRR

To calculate for the initial IRR, MNRA employs (((n - 1) / 2) +1) of the midpoint of periods of cash flows which is then supplied to the equation (2).

$$IRR = \frac{\sum_{i=1}^{n} C_i^{\frac{1}{((n-1)/2+1)}}}{|C_i|} - 1 \qquad (2)$$

Where :
n = number of cash flows;
C_i = subsequent cash flows;
C_o = initial investment;

This method increased the accuracy of the original Newton-Raphson method and also had a faster convergence. Despite this, the midpoint becomes static when there are more varying values of cash flows.

3.2 Proposed Method

Using the below proposed centroid of cash flows instead of its midpoint, one can get more accurate output, even for unequal cash flows. The MNRA served as the reference point for CNRA. This algorithm takes a dynamic approach, as incorporates the weighted average of cash flows, known as the centroid, into the initial guess calculation.

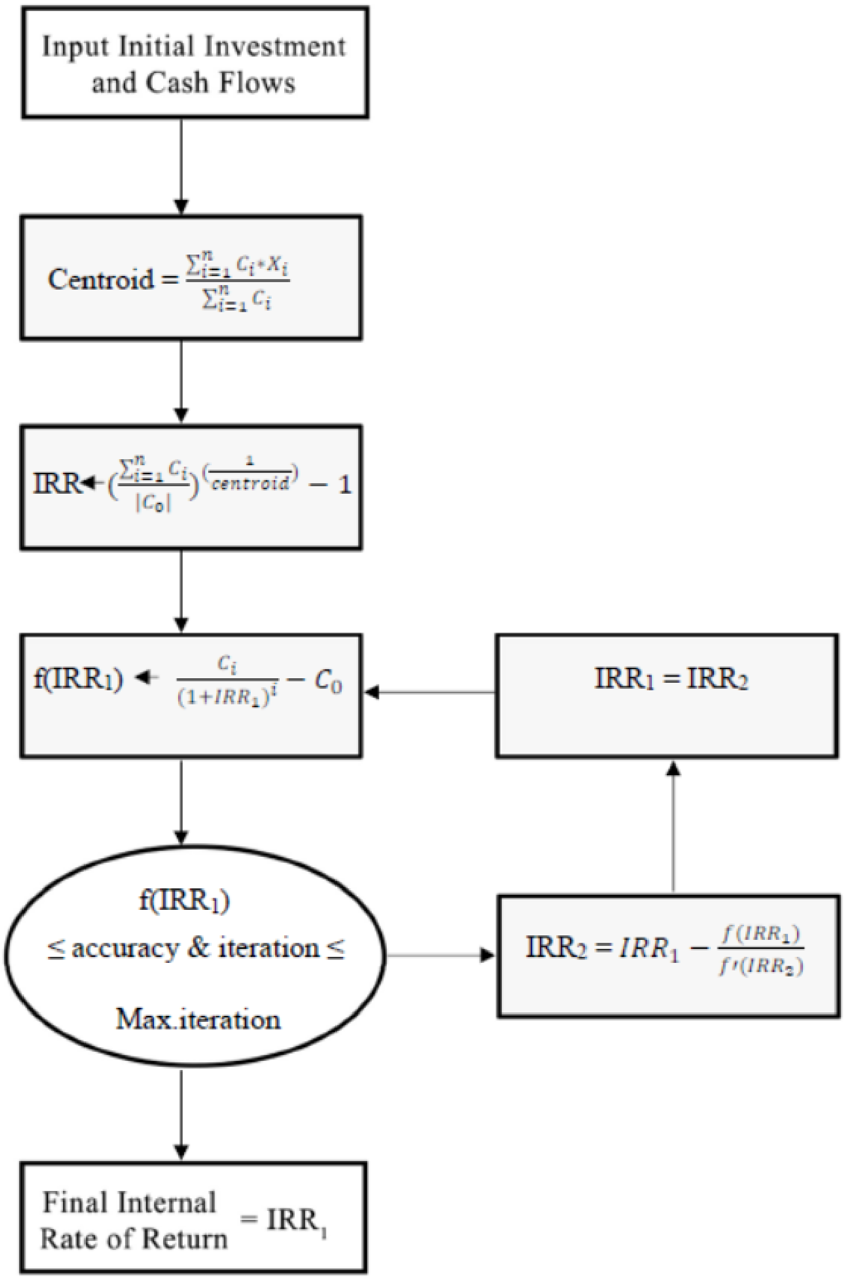

Input Initial Investment and Cash Flows
Centroid = $\frac{\sum_{i=1}^{n} C_i * X_i}{\sum_{i=1}^{n} C_i}$
IRR ← $\left(\frac{\sum_{i=1}^{n} C_i}{|C_0|}\right)^{\left(\frac{1}{centroid}\right)} - 1$
f(IRR1) ← $\frac{C_i}{(1+IRR_1)^i} - C_0$
IRR1 = IRR2
f(IRR1) ≤ accuracy & iteration ≤ Max.iteration
IRR2 = $IRR_1 - \frac{f(IRR_1)}{f'(IRR_2)}$
Final Internal Rate of Return = IRR1

Figure 3: Process of the Newton-Raphson algorithm with a centroid-based approach in calculating IRR [19].

Based on the process of the centroid approach presented in Figure 3, the centroid of time periods of cash flows is calculated with the equation (3).

$$Centroid = \frac{\sum_{i=1}^{n} C_i * X_i}{\sum_{i=1}^{n} C_i} \tag{3}$$

Where :
C_i = subsequent cash flows;
X_i = distance of C_i from time of first investment;

The centroid value is then supplied to the equation to calculate the initial IRR as displayed in the equation below (4).

$$IRR_1 = (\frac{\sum_{i=1}^{n} C_i}{|C_i|})^{(\frac{1}{centroid})} - 1 \tag{4}$$

Where :
C_0 = initial investment;

This study presents a simulation result of the CNRA and its comparison to MNRA, in terms of accuracy and number of iterations. The percentage of error in calculating the initial IRR is presented by computing the difference between the initial IRR and the final IRR over said final IRR.

$$Initial \quad IRR \quad Error = \frac{IRR_1 - IRR_f}{IRR_f} \tag{5}$$

All-in-all, the speed of the algorithm (CNRA) will be calculated by comparing the number of iterations until convergence with previous algorithms (MNRA). And the accuracy will be calculated by comparing the initial IRR error of the two methods.

4 Data

We have used two separate datasets to test the accuracy and speed of the methods. An equal cash flow, and a random(unequal) cashflow were used to test the performance of CNRA and MNRA in all two scenarios. The data was collected from a bank loan schedule. The two datasets are as follows.

Equal Cash flow:

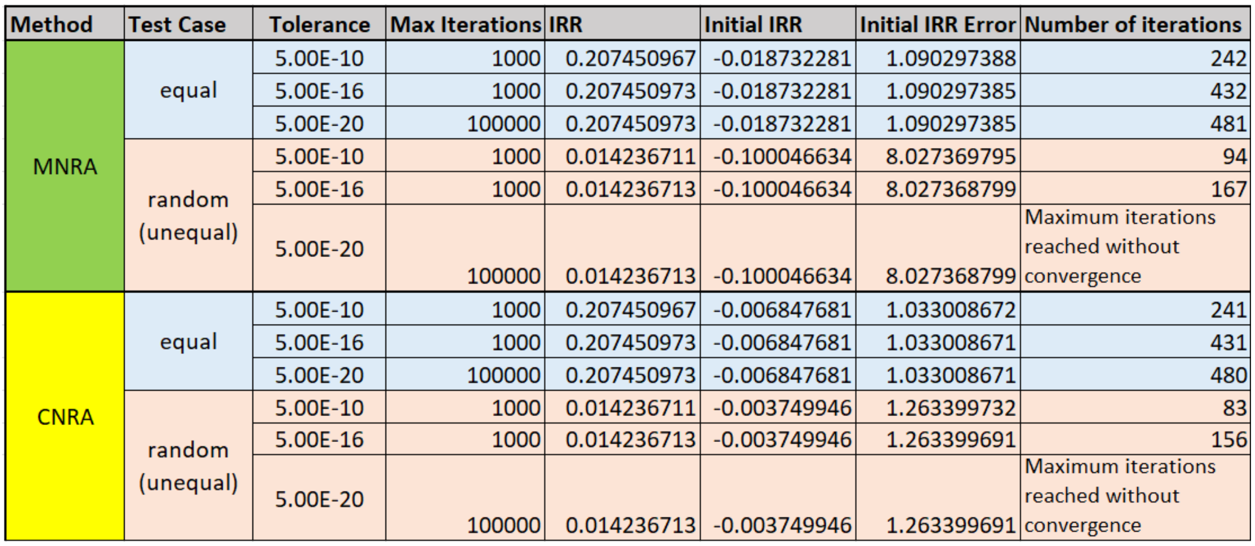
- loan of 1,000,000 LKR for 5 years, which will be paid back in equal payments of 25940.55 LKR monthly.

Random (unequal) Cash flow:

- An investment of 1,000,000 LKR and its payoffs for 5 years, which will be both positive and negative values to represent a business firm's monthly income for 5 years. The payments for each month were randomly generated using a proportion of the capital multiplied by a randomly generated number between 1 and -1. This represents a more diverse and practical cash flow, equivalent to one that we would see in practical situations.

5 Results

The below table compares the IRR output, initial IRR, initial IRR error, and number of iterations until convergence for the two methods for both equal and unequal cash flows in different tolerance levels.



Method	Test Case	Tolerance	Max Iterations	IRR	Initial IRR	Initial IRR Error	Number of iterations
MNRA	equal	5.00E-10	1000	0.207450967	-0.018732281	1.090297388	242
		5.00E-16	1000	0.207450973	-0.018732281	1.090297385	432
		5.00E-20	100000	0.207450973	-0.018732281	1.090297385	481
	random (unequal)	5.00E-10	1000	0.014236711	-0.100046634	8.027369795	94
		5.00E-16	1000	0.014236713	-0.100046634	8.027368799	167
		5.00E-20	100000	0.014236713	-0.100046634	8.027368799	Maximum iterations reached without convergence
CNRA	equal	5.00E-10	1000	0.207450967	-0.006847681	1.033008672	241
		5.00E-16	1000	0.207450973	-0.006847681	1.033008671	431
		5.00E-20	100000	0.207450973	-0.006847681	1.033008671	480
	random (unequal)	5.00E-10	1000	0.014236711	-0.003749946	1.263399732	83
		5.00E-16	1000	0.014236713	-0.003749946	1.263399691	156
		5.00E-20	100000	0.014236713	-0.003749946	1.263399691	Maximum iterations reached without convergence

Figure 4: Comparison Between CNRA and MNRA

Below figure shows how the generated IRR values in each iteration distribute from initial IRR value to the final IRR value in midpoint-based process.

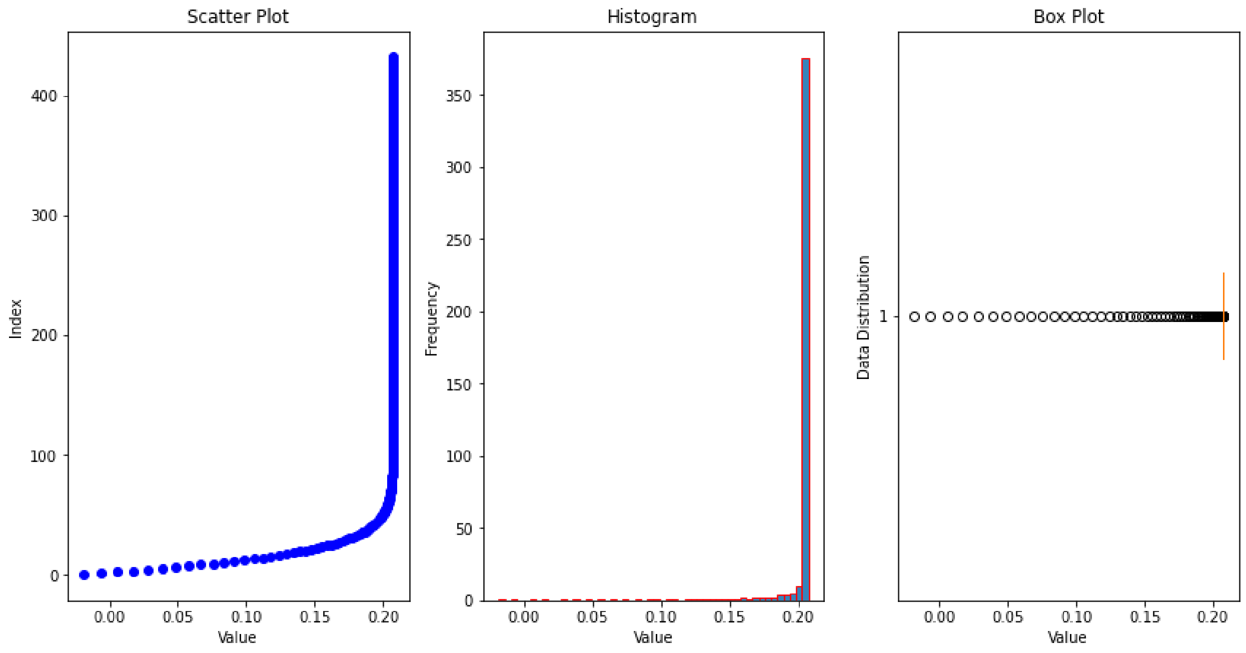


Figure 5: Visualization of iterations in MNRA process

The next figure shows how the generated IRR values in each iteration distribute from initial IRR value to the final IRR value in centroid-based process.

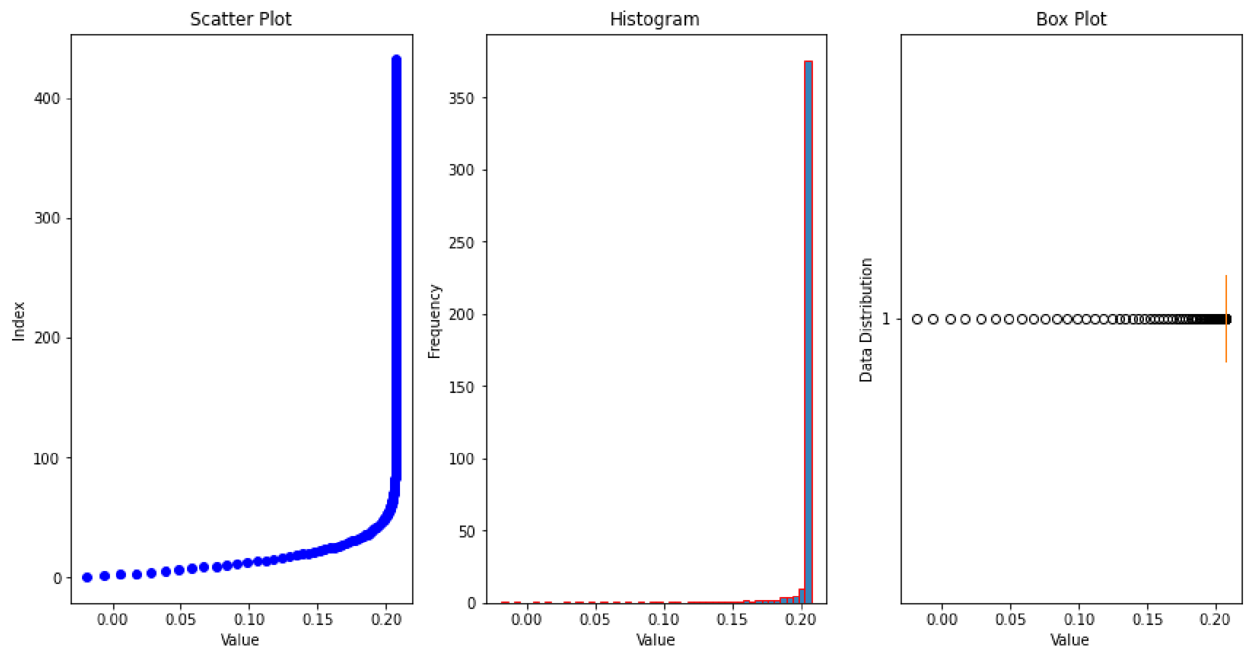


Figure 6: Visualization of iterations in MNRA process

6 Discussion and Conclusion

6.1 Analysis

In this section we study the simulation results of the CNRA and MNRA by comparing them in terms of speed and accuracy, where the speed is measured from number of iterations until convergence and the accuracy is measured using the distance between the automatically generated initial IRR value and the true IRR value.
As showd in the appendix, the python codes required to program the algorithms stated in methodology part were generated. Then, the data acquired were stored in csv files and loaded into the python programme to generate the IRR values through the mentioned processes.
In both CNRA and MNRA the algorithm automatically calculates the starting IRR required for the iterations using the centroid method and mid-point method, respectively. This greatly reduces the initial IRR error and the time (iterations) required for convergence. Both algorithms simply need the data (cash flow amounts) and it will generate the respective IRR values.
As demonstrated in Figure 4, the experiment is designed to compare the IRR values, generated initial IRR values, initial IRR errors, and number of iterations required for convergence in the two cases, CNRA and MNRA, for different tolerance values.

6.2 Discussion

Based on the trial results on Figure 4, it can be seen that the CNRA method provide relatively lesser errors then that of MNRA method. In the MNRA process for the equal cash flow, we can see that it has produced an initial IRR error of 1.09 or 109% while the CNRA process has only produced an initial IRR error of 1.033 or 103.3% for the same test case. Which is a 5.7% decrease in the initial IRR error for the proposed method, CNRA.
We can also see that as the tolerance value become smaller, the error produced by CNRA compared to MNRA decreases further.
This indicates that the CNRA has a slightly higher effectiveness and accuracy when calculating the initial IRR compared to MNRA process for equal cash flows.

When we consider the test case with unequal cash flows, we can see that the MNRA process has an initial IRR error of 8.027 or 802.7% which is very inaccurate, while the CNRA process only has an initial IRR error of 1.263 or 126.3% which is much closer to the true root, being 535.3more accurate than the previous method.
Here also we can see that as the tolerance value become smaller, the error decrease of CNRA is much more than the error decrease in MNRA process.
This implies that the suggested method, CNRA has a very higher effectiveness and accuracy when calculating the initial IRR compared to the MNRA process for unequal cash flows.
The two figures Figure 5, and Figure 6 visually represent how the initial IRR guess converges to the true IRR throughout the iterations in CNRA and MNRA processes with a tolerance value of 5x10-16.

As we can see both methods have almost similar distributions. But if one were to thoroughly examine Figure 5, and Figure 6, one will see that the starting points are a bit different. This is due to the two methods, MNRA and CNRA having different procedures to calculate the initial IRR. The CNRA starts a little bit closer to the true root of the function which is 0.207. Then in both cases the IRR values start to converge to the true IRR in an increasing manner. This indicates that though the initial guess calculation is different, both CNRA and MNRA uses the same iterative method, which being the Newton Raphson method.

If we compare the two methods in terms of convergence speed, we can use the number of iterations until convergence to represent the speed of the algorithm. As indicated in Figure 4, we can see that in both cases, the number of iterations needed to converge to the true IRR value increases generally as we reduce the tolerance value.

When considering equal cash flows, the number of iterations needed to converge doubles when the tolerance changes from 5x10-10 to 5x10-16. During the analysis process we found out that this happens between tolerance values of 5x10-15 and 5x10-16. The reason is after 6 iterations, the difference between two values of consecutive iterations being in 10-15s. Comparing the MNRA and CNRA processes, we can see that there's only a difference of 1 iteration between the two methods. The reason should be that there is only a small 5.7% difference in initial IRR error between the two methods.

If we consider the unequal (random) cash flows, the number of iterations also double as the same as in the previous case, when tolerance changes from 5x10-10 to 5x10-16. The reason is also the same as above. But here, after we change the tolerance value to 5x10-20, the algorithm does not converge to the true root even after 100,000 iterations. So we can assume that the difference between two values of consecutive iterations closer to the true root should be before there's an error of 10-20s.

Also, when comparing the two methods for unequal cash flows, we can clearly see that there's a significant difference in the number of iterations needed to convergence. In CNRA we can observe that there's a 11.7% reduction in the iteration needed to converge when tolerance value is 5x10-10, and 6.5% reduction in the iteration needed when tolerance value is 5x10-16, compared to MNRA.

6.3 Conclusion

The experimental results show that the CNRA has performed better than MNRA in practical cases. While both algorithms don't need an initial guess and generate the initial IRR value automatically, the centroid-based method has outperformed the midpoint-based method. Especially when it comes to unequal cash flows, which are more likely to be found in practical life, the CNRA method is far superior to MNRA method when calculating initial IRR. Then in the latter process, CNRA still holds the higher position in speed, comparing the number of iterations needed to converge. There also, when it comes to unequal cashflows it is obvious that CNRA process is far superior to MNRA process.

Altogether, in terms of both accuracy and speed, we can state that CNRA is more efficient in calculating IRR, even better when there're more varying values in the cash flow. Considering its accuracy and speed, the proposed algorithm, CNRA, is highly recommended for investment decision making in practical life.

Although this process has ensured convergence without the requirement of an initial guess value, it is recommended finding an analytical solution which does not require iterative process to find IRR in future research.

7 References

1. N. Pascual, A. Sison, and B. Gerardo, Calculating Internal Rate of Return (IRR) in Practice using Improved Newton-Raphson Algorithm, Philipp. Comput. J., vol. 13, no. 2, pp. 17–21, 2018.

2 E. B. Storey, (Keep score: Using internal rate of return to score investments: "and the winner is...,") J. Prop. Manag., vol. 6, p. 18+, 2016.
`https://go.gale.com/ps/i.do?p=AONE&u=anon~ae591ffc&id=GALE|A459804114&v=2.1&it=r&sid=googleScholar&asid=b15ec44b`

3 F. Zafar, N. Yasmin, S. Akram, and M. U. D. Junjua, A General Class of Derivative Free Optimal Root Finding Methods based on Rational Interpolation, Sci. World J., vol. 2015, p. Article ID 934260, 12 pages, 2015.
`https://doi.org/10.1155/2015/934260`

4 S. Chun and A. Kwasinski, Analysis of classical root-finding methods applied to digital maximum power point tracking for sustainable photovoltaic energy generation, IEEE Trans. Power Electron., vol. 26, no. 12, pp. 3730–3743, 2011.

5 J. Moten and C. Thron, Improvements on secant method for estimating Internal Rate of Return (IRR), Int. J. Appl. Math. Stat., vol. 42, no. 12, pp. 84–93, 2013.

6 V. B. Gisin and E. S. Volkova, Internal Rate of Return of Investment Projects with Fuzzy Interactive Payments, vol. 0, pp. 731–733, 2017.
`https://doi.org/10.1109/SCM.2017.7970705`

7 S. Z. M. Ruslan and M. M. Jaffar, Application of numerical method in calculating the internal rate of return of joint venture investment using diminishing musyarakah model, vol. 020007, p. 020007, 2017.
`https://doi.org/10.1063/1.4983862`

8 M. Salimi, T. Lotfi, S. Sharifi, and S. Siegmund, Optimal Newton–Secant like methods without memory for solving nonlinear equations with its dynamics, Int. J. Comput. Math., vol. 94, no. 9, pp. 1759–1777, 2017.

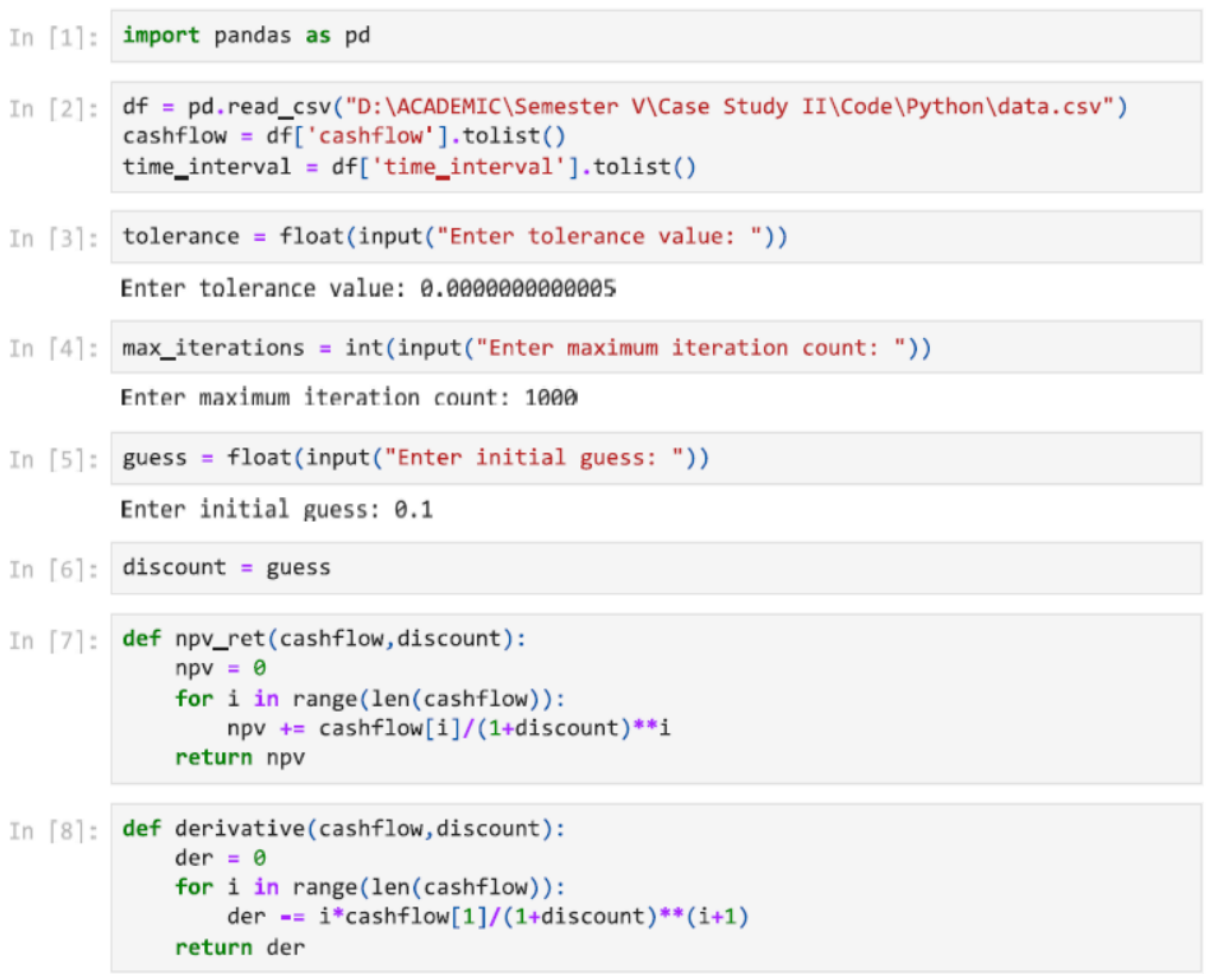
9 M. M. Mujahed and E. E. Elshareif, Internal Rate of Return (IRR): A New Proposed Approach, Benlamri R., Sparer M. Leadership, Innov. Entrep. as Driv. Forces Glob. Econ. Springer Proc. Bus. Econ. Springer, Cham, pp. 1–9, 2017.

10 N. S. Pascual, A. M. Sison, and R. P. Medina, Enhanced Newton - Raphson Algorithm in Estimating Internal Rate of Return (IRR), Int. J. Eng. Adv. Technol., vol. 8, no. 3S, pp. 389–392, 2019.

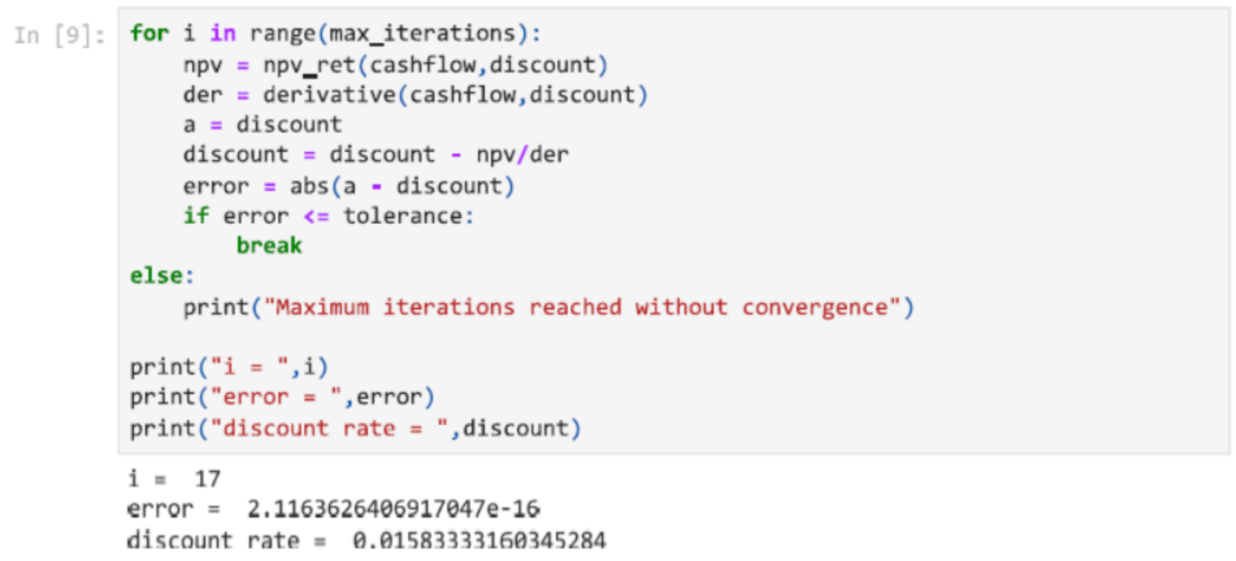
11 N. M. Nagares, A. M. Sison, and R. P. Medina, An Optimized Newton-Raphson Algorithm for Approximating Internal Rate of Return, Int. J. Adv. Trends Comput. Sci. Eng., vol. 8, no. 6, pp. 3265–3268, 2019.
`https://doi.org/10.30534/ijatcse/2019/95862019`

8 Appendix

Python code for traditional Newton Raphson method

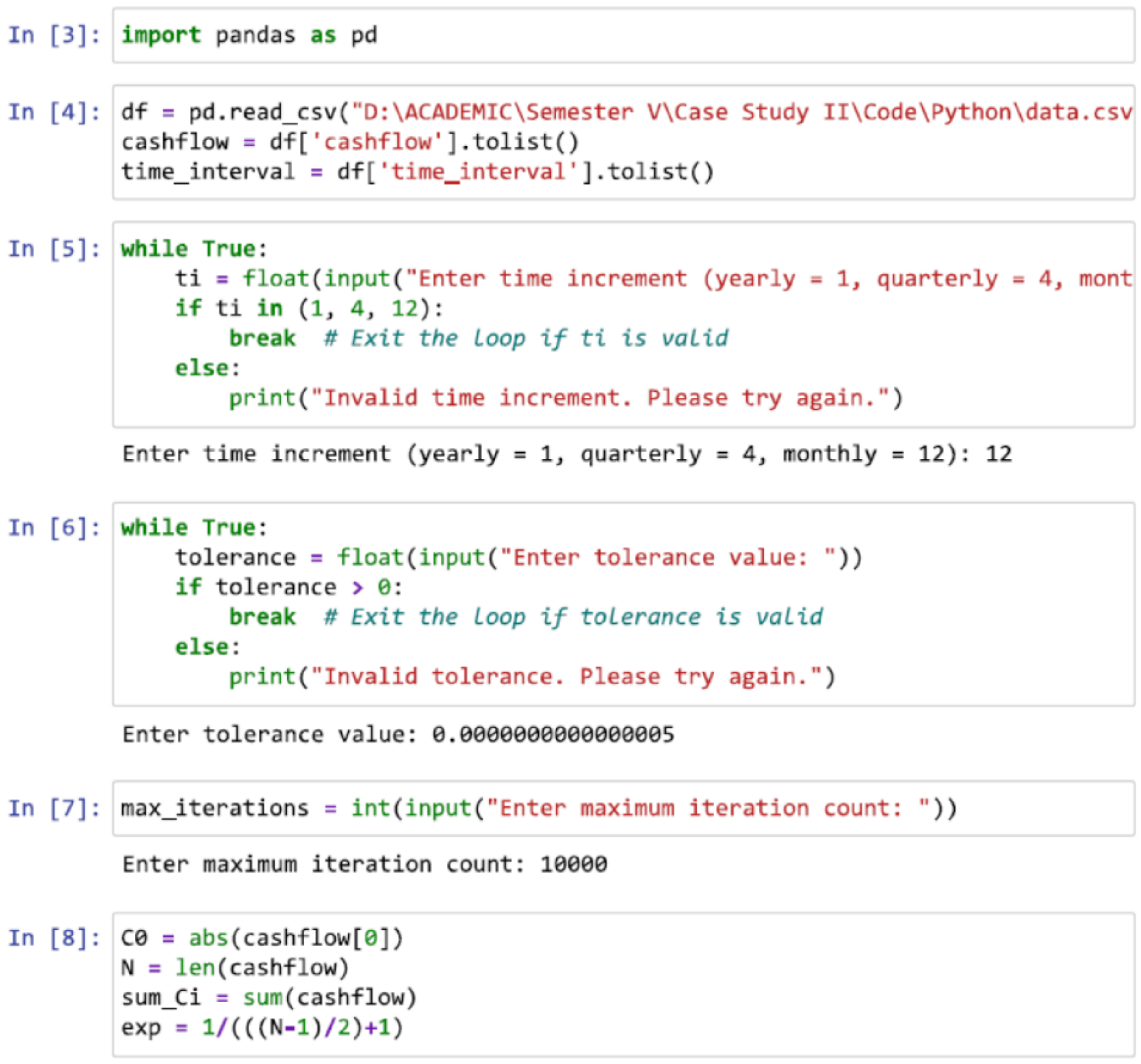


```
In [1]: import pandas as pd

In [2]: df = pd.read_csv("D:\ACADEMIC\Semester V\Case Study II\Code\Python\data.csv")
        cashflow = df['cashflow'].tolist()
        time_interval = df['time_interval'].tolist()

In [3]: tolerance = float(input("Enter tolerance value: "))

        Enter tolerance value: 0.0000000000005

In [4]: max_iterations = int(input("Enter maximum iteration count: "))

        Enter maximum iteration count: 1000

In [5]: guess = float(input("Enter initial guess: "))

        Enter initial guess: 0.1

In [6]: discount = guess

In [7]: def npv_ret(cashflow,discount):
            npv = 0
            for i in range(len(cashflow)):
                npv += cashflow[i]/(1+discount)**i
            return npv

In [8]: def derivative(cashflow,discount):
            der = 0
            for i in range(len(cashflow)):
                der -= i*cashflow[1]/(1+discount)**(i+1)
            return der
```



```
In [9]: for i in range(max_iterations):
            npv = npv_ret(cashflow,discount)
            der = derivative(cashflow,discount)
            a = discount
            discount = discount - npv/der
            error = abs(a - discount)
            if error <= tolerance:
                break
        else:
            print("Maximum iterations reached without convergence")

        print("i = ",i)
        print("error = ",error)
        print("discount rate = ",discount)

        i =  17
        error =  2.1163626406917047e-16
        discount rate =  0.015833333160345284
```

Python code for Midpoint-based Newton Raphson method



```
In [3]: import pandas as pd
```

```
In [4]: df = pd.read_csv("D:\ACADEMIC\Semester V\Case Study II\Code\Python\data.csv
        cashflow = df['cashflow'].tolist()
        time_interval = df['time_interval'].tolist()
```

```
In [5]: while True:
            ti = float(input("Enter time increment (yearly = 1, quarterly = 4, mont
            if ti in (1, 4, 12):
                break  # Exit the loop if ti is valid
            else:
                print("Invalid time increment. Please try again.")

        Enter time increment (yearly = 1, quarterly = 4, monthly = 12): 12
```

```
In [6]: while True:
            tolerance = float(input("Enter tolerance value: "))
            if tolerance > 0:
                break  # Exit the loop if tolerance is valid
            else:
                print("Invalid tolerance. Please try again.")

        Enter tolerance value: 0.0000000000000005
```

```
In [7]: max_iterations = int(input("Enter maximum iteration count: "))

        Enter maximum iteration count: 10000
```

```
In [8]: C0 = abs(cashflow[0])
        N = len(cashflow)
        sum_Ci = sum(cashflow)
        exp = 1/(((N-1)/2)+1)
```

```
In [9]:  discount = ((sum_Ci/C0)**exp)-1
         print(discount)
         IRR1=discount
         IRR=discount

         -0.018732280503210297
```

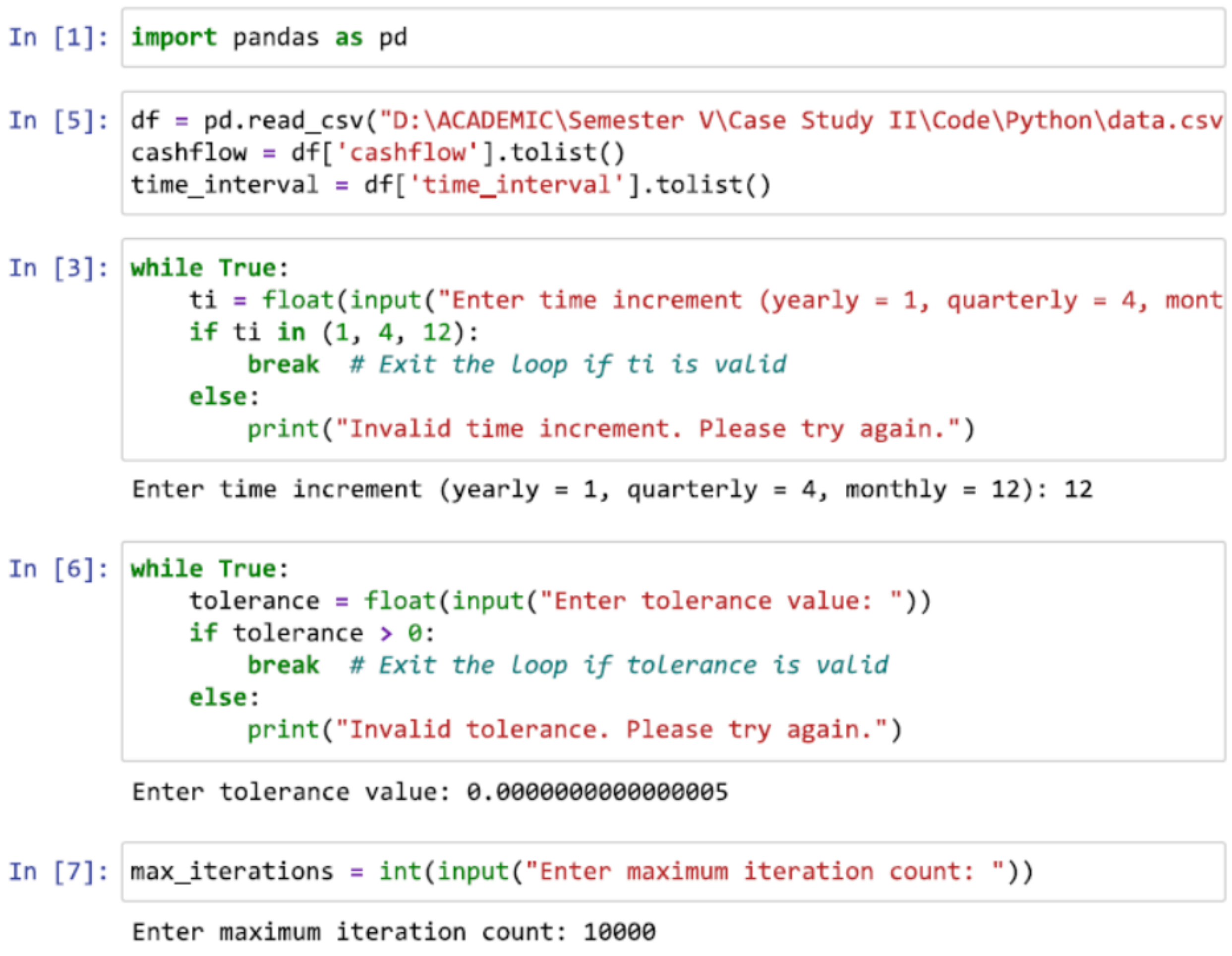
```
In [10]: def npv_ret(cashflow,IRR):
             npv = 0
             for i in range(len(cashflow)):
                 npv += cashflow[i]/(1+IRR)**(i/ti)
             return npv
```

```
In [11]: def derivative(cashflow,IRR):
             der = 0
             for i in range(len(cashflow)):
                 der -= i*cashflow[1]/(1+IRR)**((i+1)/ti)
             return der
```

```
In [12]: dis = []

         for i in range(max_iterations):
             npv = npv_ret(cashflow,IRR)
             der = derivative(cashflow,IRR)
             dis.append(IRR)
             a = IRR
             IRR = IRR - npv/der
             error = abs(a - IRR)
             if error <= tolerance:
                 break
         else:
             print("Maximum iterations reached without convergence")
```

```
In [13]: print("number of iterations =",i)
         print("Initial IRR =",IRR1)
         print("Final IRR =",IRR)
         print("Initial IRR error =",abs(IRR1-IRR)/IRR)
         print("Max.Deviation from the True IRR =",error)

         number of iterations = 432
         Initial IRR = -0.018732280503210297
         Final IRR = 0.20745097345548807
         Initial IRR error = 1.0902973854072062
         Max.Deviation from the True IRR = 4.718447854656915e-16
```

Python code for Centroid-based Newton Raphson method



```
In [1]: import pandas as pd

In [5]: df = pd.read_csv("D:\ACADEMIC\Semester V\Case Study II\Code\Python\data.csv
        cashflow = df['cashflow'].tolist()
        time_interval = df['time_interval'].tolist()

In [3]: while True:
            ti = float(input("Enter time increment (yearly = 1, quarterly = 4, mont
            if ti in (1, 4, 12):
                break  # Exit the loop if ti is valid
            else:
                print("Invalid time increment. Please try again.")

        Enter time increment (yearly = 1, quarterly = 4, monthly = 12): 12

In [6]: while True:
            tolerance = float(input("Enter tolerance value: "))
            if tolerance > 0:
                break  # Exit the loop if tolerance is valid
            else:
                print("Invalid tolerance. Please try again.")

        Enter tolerance value: 0.0000000000000005

In [7]: max_iterations = int(input("Enter maximum iteration count: "))

        Enter maximum iteration count: 10000
```

```
In [8]: # CALCULATE INITIAL DISCOUNT BASED ON CENTROID METHOD
        C0 = cashflow[0]

        product_sum = sum(Ci * Xi for Ci, Xi in zip(cashflow, time_interval))
        sum_Ci = sum(cashflow)

        centroid = product_sum / sum_Ci

        discount = (sum_Ci / abs(C0)) ** (1 / centroid) - 1
        print(discount)
        IRR1 = discount
        IRR = discount

        -0.006847680885083807
```

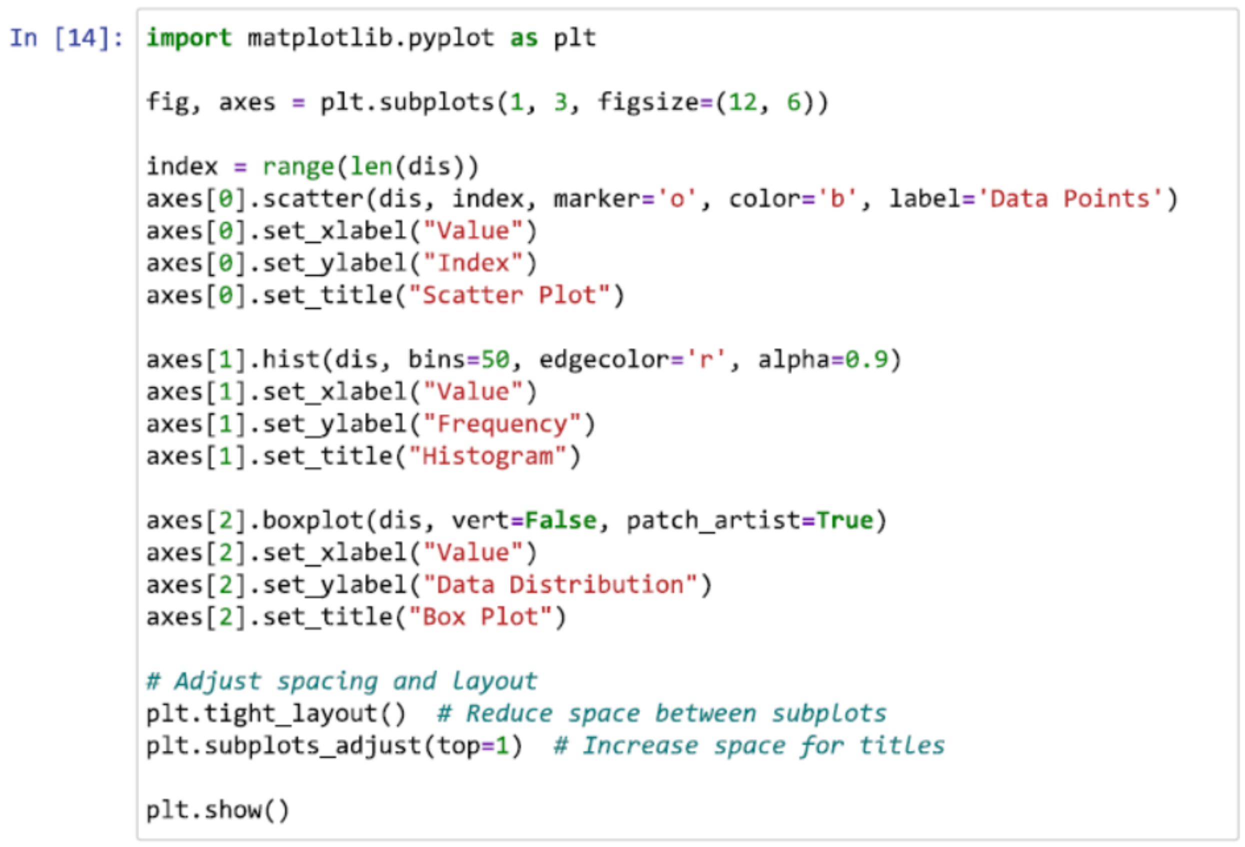
```
In [9]: # DEFINE f(X0)
        def npv_ret(cashflow,IRR):
            npv = 0
            for i in range(len(cashflow)):
                npv += cashflow[i]/(1+IRR)**(i/ti)
            return npv
```

```
In [10]: # DEFINE f'(X0)
         def derivative(cashflow,IRR):
             der = 0
             for i in range(len(cashflow)):
                 der -= i*cashflow[1]/(1+IRR)**((i+1)/ti)
             return der
```

```
In [11]: dis = []

         # NEWTON RAPHSON METHOD
         for i in range(max_iterations):
             npv = npv_ret(cashflow,IRR)
             der = derivative(cashflow,IRR)
             dis.append(IRR)
             a = IRR
             IRR = IRR - npv/der
             error = abs(a - IRR)
             if error <= tolerance:
                 break
         else:
             print("Maximum iterations reached without convergence")
```

```
In [12]: print("number of iterations =",i)
         print("Initial IRR =",IRR1)
         print("Final IRR =",IRR)
         print("Initial IRR error =",abs(IRR1-IRR)/IRR)
         print("Max.Deviation from the True IRR =",error)
```

```
number of iterations = 431
Initial IRR = -0.006847680885083807
Final IRR = 0.20745097345548807
Initial IRR error = 1.0330086707766308
Max.Deviation from the True IRR = 4.996003610813204e-16
```



```
In [14]: import matplotlib.pyplot as plt

         fig, axes = plt.subplots(1, 3, figsize=(12, 6))

         index = range(len(dis))
         axes[0].scatter(dis, index, marker='o', color='b', label='Data Points')
         axes[0].set_xlabel("Value")
         axes[0].set_ylabel("Index")
         axes[0].set_title("Scatter Plot")

         axes[1].hist(dis, bins=50, edgecolor='r', alpha=0.9)
         axes[1].set_xlabel("Value")
         axes[1].set_ylabel("Frequency")
         axes[1].set_title("Histogram")

         axes[2].boxplot(dis, vert=False, patch_artist=True)
         axes[2].set_xlabel("Value")
         axes[2].set_ylabel("Data Distribution")
         axes[2].set_title("Box Plot")

         # Adjust spacing and Layout
         plt.tight_layout()  # Reduce space between subplots
         plt.subplots_adjust(top=1)  # Increase space for titles

         plt.show()
```