South African Government bond's

Develop a program that calculates the South African Government bond's dirty price, clean price and accrued interest from its yield for a given settlement date.

Crystaline Nomasizo Dliwayo

Goverment Bond Calculator

June 4, 2024

Contents

1 Introduction

In the ever-evolving landscape of global finance, South African Government bonds have stood the test of time as a cornerstone of economic stability and growth. These financial instruments, also known as sovereign bonds, play a pivotal role in shaping the fiscal policies of nations and serve as a barometer for their economic health.This paper aims to delve into the intricate world of South African Government bonds, exploring the algorithm to calculate the Accrued Interest, All in Price (AIP), and Clean Price (CP). These values are defined in the JSE's Bond Pricing Formula.

The market volatility has become the norm, government bonds often emerge as a safe haven for investors. However, like any investment, they are not without risk. The purpose of this paper is to write a Java program that will calculate the goverment bond alogorithms for the Accrued interest, AIP, CP additionally applying different test cases scenarios to derive the values of Accrued interest, AIP, CP and lastly applying the JUnit test to verify if the results pass or fail.

This study seeks to:

- Develop a Java program that will implement the government bond algorithms for calculating Accrued Interest, AIP, and CP using the following programming languages Java.
- This program will be designed to handle various test case scenarios to derive the values of Accrued Interest, AIP, and CP
- The JUnit tests will be developed to verify whether the calculated values of Accrued Interest, AIP, and CP pass or fail based on predefined criteria. This rigorous testing process will ensure the reliability and robustness of the developed Java program.

2 South African Government bonds Alogorithms

This methodology report outlines the JSE bond exchange alogorithms of calculating various parameters related to South African Government bonds. The parameters include Accrued Interest, All-in Price (AIP), Clean Price (CP), and Dirty Price.The following imports were applied when developing the Govermentment bond calculator java.time.LocalDate and java.time.temporal.ChronoUnit. This was done suing the intelliJ environment instead of eclipse due to errors when applying the Junit test.The program is structured into two main classes: GovermentBond_A and BondDetails.

2.1 Class: BondDetails

This class is used to store the details of a bond. It includes the following attributes:

- **maturityDate**: The date when the bond matures and the face value is paid back to the bondholder.
- **CouponRate**: The annual interest rate paid on the bond's face value.
- **R**: The face value of the bond (redemption amount)
- **NCD**: The date of the next coupon payment.
- **LCD**: The date of the last coupon payment.
- **booksCloseDate1** and **booksCloseDate2**: The dates when the bond's books are closed for the calculation of the next coupon payment.
- **settlementDate**: The date when the bond transaction is settled, i.e., the buyer pays for the bond.
- **Yield**: The bond's market yield.

The BondDetails class also includes two methods:

- **calculateCUMEX()**: This method calculates CUMEX (Coupon Ex-Dividend), which indicates whether the bond's settlement date is before the books close dates. If it is, CUMEX is 1; otherwise, it's 0.
- **calculateN()**: This method calculates N (number of periods), which is the number of semi-annual periods between the Next Coupon Date and the maturity date.

2.2 Class: GovermentBond_A

This class contains the main method and several static methods for calculating and printing bond details:

- **calculateAccruedInterest(BondDetails bond):** : calculateAccruedInterest(BondDetails bond): This method calculates the accrued interest using the formula:

  $\text{Accrued Interest} = \text{Days Accrued} \times \text{Coupon Rate}\frac{}{\text{Days in Year}}$

$\text{Accrued Interest} = \text{daysAccrued } \times \text{bond.couponRate}_{\overline{\text{DAYS\_IN\_YEAR}}}$

where Days Accrued is the number of days between the Last Coupon Date and the Settlement Date.

- **calculateAIP(BondDetails bond)** This method calculates the All-in Price (AIP):

  AIP is calculated using the below formular:

  When the discount factor equals 1 then:

  $$\text{AIP} = \text{BPF} \times \left(\text{CPNatNCD} + \text{CPN} \times \left(\frac{\text{discountFactor} \times (1 - \text{discountFactor}^N)}{1 - \text{discountFactor}}\right) + \text{bond.R} \times \text{discountFactor}^N\right)$$

  When the discount factor equals 1 then:

  $\text{AIP} = \text{CPNatNCD} + \text{CPN } \times N + \text{bond.R}$

- **calculateCleanPrice(double aip, double accruedInterest)** This method calculates the Clean Price by subtracting the Accrued Interest from the AIP:

  CleanPrice = AIP - AccruedInterest

- **calculateDirtyPrice(double cleanPrice, double accruedInterest)** This method calculates the Dirty Price by adding the Accrued Interest to the Clean Price:

  DirtyPrice = CleanPrice + AccruedInterest

- **calculateAndPrint(BondDetails bond)** This method calls the above methods to calculate the Accrued Interest, AIP, Clean Price, and Dirty Price for a given bond, and then prints these values.

3 Unit Testing for Government Bond Calculation in Java

The code is structured into a single class: GovermentBond_ATest. This class contains two test methods: testR186_bondCalculation and testR2032_bondCalculationB.

3.1 Class: GovermentBond_A

This class contains unit tests for the GovermentBond_A class. Each test method creates a BondDetails object with specific parameters same from the original code, calls the calculation methods from the GovermentBond_A class, and asserts that the calculated values match the expected results.

3.2 Class: Key Tests - testR186_bondCalculation

This test method creates a BondDetails object for the R186 bond with specific parameters same from the original code. It then calculates the accrued interest, All-In Price (AIP), and clean price for the bond using the methods from the GovermentBond_A class. The calculated values are printed to the console and compared with the expected values using the assertEquals method from the JUnit framework.

3.3 Class: Key Tests - testR2032_bondCalculationB

This test method creates a BondDetails object for the R2032 bond with specific parameters. Similar to the previous test, it calculates the accrued interest, AIP, and clean price for the bond, prints the calculated values, and asserts that they match with the expected values for the test to either pass or fail.

3.4 Utility Method

The round method is a utility method used to round a double value to a specified number of decimal places. This method is used to round the calculated values before printing and comparing them with the expected values.

4 Code usage and other bond test cases scenarios

The parameters for the R186 and R2032 bonds are defined within the code. When the code is executed in IntelliJ, it outputs the Accrued Interest, All-In Price (AIP), Clean Price (CP), and Dirty Price. If a user wishes to perform similar calculations for another government bond, they would need to add the new bond's details under the main method, and the results for the new bond will be printed.

I conducted several test case scenarios by adding the same bond parameters under the main method but changed the coupon rate to 0. This was done to observe the impact on the Accrued Interest, AIP, and CP when there are no periodic interest payments. Interestingly, if the accrued interest is always zero, it results in the All-In Price (AIP) being identical to the Clean Price (CP).

One test case involved setting the Next Coupon Date (NCD) equal to the Maturity Date, which resulted in N being 0 and the AIP, CP, and Dirty Price yielding negative results. This scenario is unrealistic in normal markets as it implies that purchasing the bond not only costs nothing but actually provides money to the buyer.

Another test case involved setting the Settlement Date after the Books Close Date. In this scenario, the buyer pays the accrued interest but does not receive the upcoming coupon payment. Instead, the coupon payment is made to the seller who was the holder of record on the books close date.

5 Conclusion

The implementation of government bond algorithms using Java, specifically for the JSE bond exchange, has proven to be highly effective. These algorithms allow for the precise calculation of various bond parameters such as Accrued Interest, All-In Price (AIP), Clean Price (CP), and Dirty Price. This is crucial in financial markets where accurate and timely information is key to making informed investment decisions.

The use of Java provides a robust and flexible platform for these calculations. Its object-oriented nature allows for the creation of bond objects with defined parameters, making the code reusable and easy to manage. Furthermore, Java's extensive library support facilitates complex mathematical calculations required in bond valuation.

The integration of JUnit tests within this process cannot be overstated. These tests ensure the accuracy and reliability of the bond algorithms. By simulating different scenarios, such as varying coupon rates or settlement dates, JUnit tests provide a comprehensive evaluation of the algorithm's performance under diverse market conditions. This is particularly useful in identifying potential issues or anomalies that could impact the algorithm's functionality.

Moreover, JUnit tests offer a way to validate the impact of changes in bond parameters on the calculated results. This is essential in a dynamic financial market where bond parameters can fluctuate. By ensuring that the algorithms respond correctly to these changes, JUnit tests help maintain the integrity of the bond valuation process.

6 Code Applied

This will be submitted under the GitHub.