Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Analytical Finance II Report
Course Code: MMA708

AN APPROACH TO BOOTSTRAP THE SOFR-SWAP CURVE USING PYTHON

Romann IVANOFF
roman.ivanoff@edu.devinci.fr

Rémi LIEBIG
remi.liebig@edu.devinci.fr

Vincent NAZZARENO
vincent.nazzareno@edu.devinci.fr

Examiner: Jan Römann
Mälardalen University, Västerås

24th October 2024

Background and motivations

The project aims to construct a SOFR swap curve, using data from the Federal Reserve Economic Data (FRED) platform. The Secured Overnight Financing Rate (SOFR) has become a key benchmark in financial markets, especially following the transition away from LIBOR. Accurately modeling the SOFR curve is crucial for various financial instruments, including interest rate derivatives, and is of high importance for risk management, valuation, and pricing purposes.

In this context, bootstrapping provides a reliable and accurate method for constructing a swap curve. The use of Python for this process enhances flexibility, reproducibility and data visualization. By constructing this curve, the project contributes to the broader goal of enhancing understanding and modeling of interest rate benchmarks in the post-LIBOR financial landscape, with SOFR becoming a dominant reference rate.

Method

The bootstrapping method for constructing the SOFR swap curve will involve the following steps:

1. **Data collection**: Historical SOFR rates will be retrieved from FRED using Python.
2. **Curve construction**: Using a bootstrapping approach described in Section 6.1.4 of the reference material [1], the swap curve will be iteratively constructed.
3. **Modeling in Python**: The bootstrapping algorithm will be implemented in Python to automate the curve-building process. This will include the construction of zero-coupon rates and forward rates as needed.
4. **Validation**: The resulting SOFR curve will be validated against known market data for accuracy.

Outcomes

The primary outcome of this work will be a Python-based framework for constructing and analyzing the SOFR swap curve. This framework will:

- Provide a detailed and reliable swap curve based on SOFR data.
- Serve as a foundation for further extensions, such as multi-curve frameworks or the integration of other interest rate benchmarks.

Limitations

Several factors may limit the scope of this work:

- Availability and quality of historical SOFR data from FRED.
- Computational limitations when dealing with large datasets in Python.
- The accuracy of the bootstrapped curve is dependent on the interpolation methods and market data consistency.

Methodology

In this section, we describe the methodology used to bootstrap the SOFR swap curve. The bootstrapping procedure involves constructing a continuous yield curve by sequentially solving for rates in different maturity segments, specifically using short-term cash deposits, medium-term forward rate agreements (FRAs), and long-term interest rate swaps. This section is divided into three parts corresponding to these time segments.

Short Term: Cash Deposits

We start by calculating the discount factor and from it, the zero rate. Using the overnight rate (O/N), the discount factor is given by:

$$D_{\mathrm{O/N}} = \frac{1}{1 + q_{\mathrm{O/N}} \cdot \frac{d_{\mathrm{O/N}}}{360}}$$

From here, we can calculate the zero rate as:

$$Z_{\mathrm{O/N}} = -100 \cdot \frac{\ln(D_{\mathrm{O/N}})}{\frac{d_{\mathrm{O/N}}}{365}}$$

Zero rates are commonly given as continuously compounded rates using the Act/365 day count convention. If today is a Friday, then $d_{\mathrm{O/N}} = 3$ instead of $d_{\mathrm{O/N}} = 1$ as shown above.

Next, we calculate the T/N discount factor using the following equation:

$$D_{\mathrm{T/N}} = \frac{D_{\mathrm{O/N}}}{1 + q_{\mathrm{T/N}} \cdot \frac{d_{\mathrm{T/N}}}{360}}$$

From here, the zero rate can be calculated as:

$$Z_{\mathrm{T/N}} = -100 \cdot \frac{\ln(D_{\mathrm{O/N}})}{\frac{d_{\mathrm{T/N}}}{365}}$$

For longer maturities in the short-term segment (such as 1-week, 1-month, 2-month, and 3-month deposits), the discount factor D_i is calculated as:

$$D_i = \frac{D_{\mathrm{T/N}}}{1 + q_i \cdot \frac{d_i}{365}}, \quad i = \{\mathrm{t_{1W}}, \mathrm{t_{1m}}, \mathrm{t_{2m}}, \mathrm{t_{3m}}\}$$

The zero rate for these maturities is given by:

$$Z_i = -100 \cdot \frac{\ln(D_i)}{\frac{d_i}{365}}, \quad i = \{\mathrm{t_{1W}}, \mathrm{t_{1m}}, \mathrm{t_{2m}}, \mathrm{t_{3m}}\}$$

As seen above, we use $D_{\mathrm{T/N}}$ as the numerator. However, the choice of the numerator depends on the spot lag, which is defined as the number of business days between the trade date and the value date.

Finally, the possible choices are:

1. Spot lag = 0: The numerator is 1.
2. Spot lag = 1: The numerator is $D_{\mathrm{O/N}}$.
3. Spot lag > 1: The numerator is $D_{\mathrm{T/N}}$.

Medium Term: Forward Rate Agreements - FRA

Next, we are ready to handle the FRA rates. Since these are forward contracts (with netted principal cash payments), the corresponding quoted rates are forward rates. Therefore, we need a so-called stub rate. The stub rate shall have its maturity on the same date as the start date of the first FRA contract. This stub has to be calculated only if we have IMM FRA contracts. We can calculate that rate using (linear) interpolation on the discount factors:

$$D_{\mathrm{stub}}(t) = D(t_0) + \frac{D(T) - D(t_0)}{T - t_0}(t - t_0)$$

The stub rate is then given by:

$$z_{\mathrm{stub}} = -\frac{100 \cdot \ln(D_{\mathrm{stub}})}{d_{\mathrm{stub}}/365}$$

where d_{stub} is the number of days between today and the stub maturity. Now, we can handle the FRA rates as given below:

$$D_{\text{i}}^{\text{FRA}} = \frac{D_{i-1}^{\text{FRA}}}{1 + q_{\text{FRA}} \cdot \frac{d_{\text{FRA}}}{360}}, \quad i = \{\text{t}_{3\text{m}}, \text{t}_{6\text{m}}, \text{t}_{9\text{m}}\}$$

where $D_0^{\text{FRA}} = D_{\text{stub}}$. From here we get the zero rate as:

$$Z_{\text{FRA}}(T) = -100 \cdot \frac{\ln(D_i^{\text{FRA}})}{\frac{t_{\text{i}} - t_0}{365}}, \quad i = \{\text{t}_{3\text{m}}, \text{t}_{6\text{m}}, \text{t}_{9\text{m}}\}$$

If the FRA contracts are quoted in price, the first FRA discount factor is calculated as:

$$D_{\text{FRA}}(t) = \frac{D_{\text{stub}}(t)}{1 + \left(\frac{100 - P_{\text{FRA}}}{100}\right) \cdot \frac{91}{360}}$$

where P_{FRA} is the quoted price of the FRA contract and 91 is the number of days between the two IMM dates. The implied par rates are calculated as:

$$r_{\text{implied par}}(t_{\text{FRA}}) = 100 \cdot \left(\frac{D_{TIN}}{D_{\text{FRA}}} - 1\right) \cdot \frac{1}{t_{\text{FRA}} - t_{TIN}}$$

Long Term: Swaps

After computig zero-rates with medium term variables (FRAs) we can continue with the rest of the curve, which is given by swap rates. Since they start at 3Y, we first calculate approximate swap rates for 1Y and 2Y by (linear) interpolation.

We now recall how to calculate the par swap rates r_T^{par}:

$$r_T^{\text{par}} = \frac{D_{TIN} - D_T}{\sum_{i=1}^{T} Y_i \cdot D_i} = \frac{D_{TIN} - D_T}{\sum_{i=1}^{T-1} Y_i \cdot D_i + Y_T \cdot D_T}$$

We then have:

$$D_{TIN} = \frac{r_T^{\text{par}} \sum_{i=1}^{T-1} Y_i \cdot D_i}{1 + Y_T \cdot r_T^{\text{par}}}$$

where Y_t is the year fraction at time t, given by:

$$Y_t = \frac{360 \cdot (y_t - y_{t-1}) + 30 \cdot (m_t - m_{t-1}) + d_t - d_{t-1}}{360}$$

where y_{t-1} is the previous year where a rate exists, m_t the month for the rate, and d_t the days. For the years where swap rates are not quoted in the market (11Y, 13Y, 14Y, 16Y, etc.), we use (linear) interpolation to find the zero rate when calculating the discount factors. Suppose all tenor spreads are zero, then the swap rates can be considered as maturing once a year. This gives $Y_t = 1$ for all t. We then have:

$$D_T = \frac{D_{TIN} - r_T^{\text{par}} \sum_{i=1}^{T-1} Y_i \cdot D_i}{1 + Y_T \cdot r_T^{\text{par}}}$$

Newton-Raphson: Derive new maturities

To get high accuracy in the calculated values where we have to use extrapolation, we also use a Newton-Raphson method. This is applied for the discount factors. The Newton-Raphson method calculates the discount factors so that we can reprice the swaps and find their quotes, i.e., the interest rates for the fixed legs.

If we have semi-annual data, the above formula is replaced by

$$D_T = \frac{D_{TIN} - r_T^{\text{par}} \sum_{i=1}^{T-0.5} Y_i \cdot D_i}{1 + \frac{r_T^{\text{par}}}{2}}$$

From here, we get the zero rate as usual:

$$Z_T = -100 \cdot \frac{\ln(D_T)}{d_T/365}$$

If the zero-coupon rates are expressed as annual bond equivalent yields, we have:

$$D_T = \frac{1}{\left(1 + \frac{Z_T}{100}\right)^{\frac{d_T}{365}}}$$

Solving the previous equation, the zero-coupon rate is:

$$Z_T = \left(\frac{1}{D_T}\right)^{\frac{365}{d_T}} - 1$$

If you want continuously compounded zero rates, the discount factor will be calculated as:

$$D_T = \exp\left(\frac{Z_T}{100} \cdot \frac{-d_T}{365}\right)$$

From the latter equation, the zero-coupon rate becomes a function of the discount factor, as follows:

$$Z_T = -\left(\ln(D_T) \cdot \frac{365}{d_T}\right) \cdot 100$$

If you prefer to represent zero-coupon rates as simple annualized rates, the discount factor should be written as:

$$D_T = \frac{1}{1 + \frac{Z_T}{100}}$$

Results and Analysis

Bootstrapped Zero-Coupon Curve

The bootstrapped zero-coupon curve is derived from the quoted market rates for different maturities. In the figure below, we observe the following components:

- **Zero Rate (Blue)**: The bootstrapped zero-coupon rates obtained from discount factors derived from market data.
- **Quote Rate (Orange)**: The quoted market rates used to bootstrap the zero-coupon rates.
- **Difference from Book (Green)**: The difference between the bootstrapped zero-coupon rates and a reference curve from the book we based this project on.

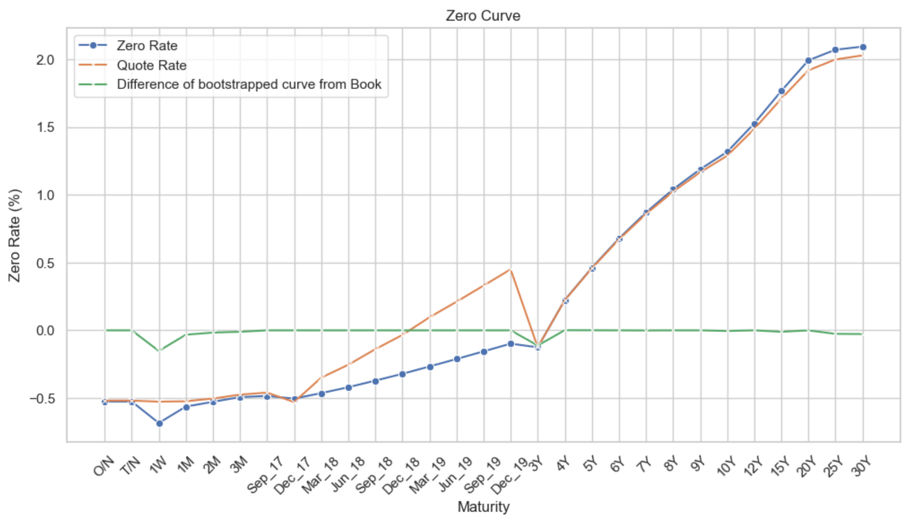

Figure 1: Bootstrapped Zero-Coupon Curve

Key Insights:

- The bootstrapped curve (blue) closely follows the quoted rates (orange), demonstrating a good calibration of the curve to market data.
- The difference from the book curve (green) remains minor, indicating that the bootstrapped curve aligns with the book bootstrapped curve.

Cubic Spline Interpolation of Zero Rates

Cubic spline interpolation is applied to smooth the bootstrapped zero-coupon rates across various maturities, creating a continuous and refined curve. The plot below shows:

- **Original Zero Rates (Red Dots)**: The refined zero-coupon rates derived from the bootstrapping process.
- **Cubic Spline Interpolation (Blue Line)**: A continuous curve generated using cubic spline interpolation, providing a smooth representation of the zero-coupon rates across maturities.

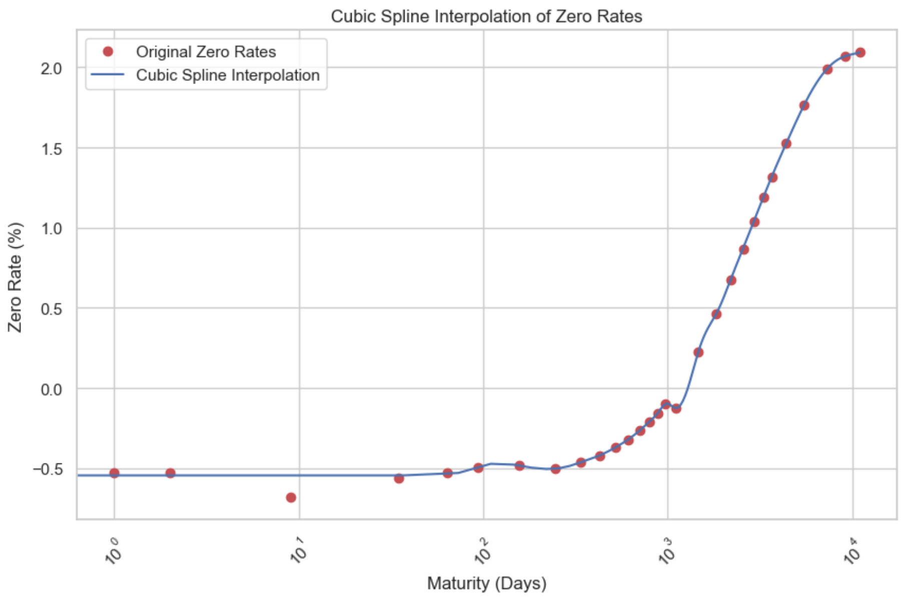

Figure 2: Cubic Spline Interpolation of Zero-Coupon Rates

Key Insights:

- The interpolation smooths out the zero-coupon rates, particularly in areas where data points are sparse (e.g., between 10 to 30 years).
- The cubic spline curve closely follows the original zero rates while ensuring smooth transitions between maturities.

Analysis of Results

The bootstrapped zero-coupon rates are derived directly from the quoted market rates, which reflect the cost of borrowing at various maturities. These rates are essential for pricing interest rate derivatives and fixed-income instruments, as they provide a reliable reference for market conditions across different time periods.

The quote rates represent the market rates at which swaps or other financial instruments are traded. The primary objective of the bootstrapping process is to generate a smooth zero-coupon curve based on these market rates. This curve allows for accurate pricing and valuation of financial instruments.

The difference between the bootstrapped curve and the reference theoretical curve (from the book) is minimal. This indicates that the bootstrapping process was performed accurately, as the bootstrapped zero-coupon rates align closely with the theoretical expectations.

The cubic spline interpolation further improves the smoothness of the curve, ensuring that the zero-coupon rates are continuously represented across all maturities. This continuous representation is particularly beneficial for pricing instruments and performing sensitivity analyses, where a smooth and uninterrupted rate curve is required.

Limitations

A More General Bootstrap

In bank risk systems, the bootstrapping process is more complex due to multiple currencies and instruments. Each curve is bootstrapped using selected nodes, such as:

$$\{1d, 2d, 3d, 1w, 1m, 2m, 3m, 6m, 9m, 1y, 2y, 3y, 4y, 5y, 7y, 10y, 12y, 15y, 20y, 25y, 30y\}$$

These nodes simplify the comparison across portfolios. The zero-coupon yield curves are used to compute the risk by shifting these nodes. Specific interpolation methods between nodes allow repricing of the instruments used to bootstrap the curve. Different methods can lead to different curves. Traders hedge using market instruments, which may lead to differing curves in risk and trading systems.

Interpolation Methods

To smooth the link between maturities term (short/medium/long) we can use different interpolation methods, each having different formulas.

- Piecewise constant interpolation.
- Linear interpolation
- Polynomial interpolation
- Spline interpolation

In our work we used linear interpolation for quick calculation and cubic spline for smoothing of the whole curve. A better approach would be to do spline interpolation on the different links.

References

[1] J. R. M. Röman, *Analytical Finance: The Mathematics of Interest Rate Derivatives, Markets, Risk and Valuation*. Springer, 2017, vol. 2, ISBN: 978-3-319-52583-9. DOI: 10.1007/978-3-319-52584-6. [Online]. Available: https://doi.org/10.1007/978-3-319-52584-6.

1. Appendix

The following functions are used throughout the bootstrapping process for the short, medium and long term maturities.

Appendix A: Python Code for short-term maturities

This section provides the Python code used to calculate the discount factors and zero-coupon rates for short-term curves.

```
def calculate_discount_factor(rate, period_days, base_discount_factor=1):
    """
    Function to calculate the discount factor.
    Parameters:
    - rate: The quoted rate (in decimals).
    - period_days: The number of days for the term.
    - base_discount_factor: The initial discount factor (default is 1).

    Returns:
    - The calculated discount factor.
    """
    return base_discount_factor / (1 + rate * (period_days / 360))

# Function to calculate zero-coupon rates
def calculate_zero_rate(discount_factor, period_days):
    """
    Function to calculate the zero-coupon rate.
    Parameters:
    - discount_factor: The discount factor for the given term.
    - period_days: The number of days for the term.

    Returns:
    - The zero-coupon rate (in percentages).
    """
    return -100 * (log(discount_factor) / (period_days / 365))

def short_term_ZC_curve(terms, quoted_rates, start_date_short_str, maturity_short_str):
    """
    Function to compute the short-term zero-coupon curve and discount factors.
    Parameters:
    - terms: List of terms in days.
    - quoted_rates: List of quoted rates (in decimals).
    - start_date_short_str: List of start dates as strings.
    - maturity_short_str: List of maturity dates as strings.

    Returns:
    - D_short: Discount factors for short-term instruments.
    - ZC_short: Zero-coupon rates for short-term instruments.
    """
    ZC_short = []
    D_short = []

    # Calculate business days between start and maturity dates
    bus_day = [np.busday_count(start_date_short_str[i],
               maturity_short_str[i]) for i in range(len(start_date_short_str))]

    D_ON = calculate_discount_factor(quoted_rates[0], terms[0])
    Z_ON = calculate_zero_rate(D_ON, terms[0])

    D_TN = D_ON / (1 + (quoted_rates[1] * terms[1]) / 360)
    Z_TN = calculate_zero_rate(D_ON, terms[1])

    ZC_short.extend([Z_ON, Z_TN])
    D_short.extend([D_ON, D_TN])

    for i in range(2, len(start_date_short_str)):
        if bus_day[i] == 0:
            D_i = calculate_discount_factor(quoted_rates[i], terms[i])
        if bus_day[i] == 1:
            D_i = D_ON / (1 + quoted_rates[i] * (terms[i] / 360))
        else:
            D_i = D_TN / (1 + quoted_rates[i] * (terms[i] / 360))

        Z_i = calculate_zero_rate(D_i, terms[i])
        D_short.append(D_i)
        ZC_short.append(Z_i)

    return D_short, ZC_short
```

Appendix B: Python Code for medium-term maturities

This section provides the Python code used to calculate the discount factors and zero-coupon rates for medium-term curves.

```
def mid_term_ZC_curve(terms, quoted_rates_short, quoted_rates_mid, t_short, maturity_short, maturi
    """
    Function to compute the mid-term zero-coupon curve and discount factors.
    Parameters:
    - terms: List of terms in days.
    - quoted_rates_short: List of short-term quoted rates (in decimals).
    - quoted_rates_mid: List of mid-term quoted rates (in decimals).
    - t_short: List of short-term maturities in days.
    - maturity_short: List of short-term maturity dates.
    - maturity_short_str: List of short-term maturity dates as strings.
    - start_date_short: List of short-term start dates.
    - start_date_short_str: List of short-term start dates as strings.
    - start_date_mid: List of mid-term start dates.

    Returns:
    - discount_factors: Discount factors for mid-term instruments.
    - zero_rates: Zero-coupon rates for mid-term instruments.
    """
    discount_factors = []
    zero_rates = []

    short_DF = short_term_ZC_curve(t_short, quoted_rates_short, start_date_short_str, maturity_sho

    t0 = maturity_short[-2]
    T = maturity_short[-1]
    t_stub = start_date_mid[0]
    start_date0 = start_date_short[0]

    D_t0 = short_DF[0][-2]
    D_T = short_DF[0][-1]
    D_stub = D_t0 + (D_T - D_t0) * ((t_stub - t0).days / (T - t0).days)
    z_stub = calculate_zero_rate(D_stub, (t_stub - start_date0).days)

    D_i1 = D_stub / (1 + quoted_rates_mid[0] * (terms[0] / 360))
    Z_i1 = calculate_zero_rate(D_i1, (start_date_mid[1] - start_date0).days)

    discount_factors.append(D_i1)
    zero_rates.append(Z_i1)

    for i in range(1, len(terms)):
        start_i = start_date_mid[i + 1]
        D_im1 = discount_factors[-1]

        D_i = D_im1 / (1 + quoted_rates_mid[i] * (terms[i] / 360))
        Z_i = calculate_zero_rate(D_i, (start_i - start_date0).days)

        discount_factors.append(D_i)
        zero_rates.append(Z_i)

    return discount_factors, zero_rates
```

The code above is used to calculate the discount factors and zero-coupon rates for short-term and mid-term instruments during the bootstrapping process.

Appendix C: Python Code for long-term maturities

This section provides the Python code used to calculate the discount factors and zero-coupon rates for long-term curves.

```
import math

def long_term_bootstrap(swap_rates_dict, D_t_dict, Z_t_dict):
    """
    Function to bootstrap the zero-coupon curve and discount factors for
    long-term maturities
    using swap rates. This function calculates the discount factors and
    zero-coupon rates
    for maturities from 3 to 30 years using the provided swap rates.
    It builds on the pre-existing discount factors and zero rates for shorter
    maturities.

    Parameters:
    - swap_rates_dict: A dictionary mapping maturity terms (in years) to
      corresponding swap rates.
    - D_t_dict: A dictionary storing discount factors for previously
      bootstrapped maturities.
    - Z_t_dict: A dictionary storing zero-coupon rates for previously
      bootstrapped maturities.

    Returns:
    - Updates D_t_dict and Z_t_dict with bootstrapped discount factors and
      zero-coupon rates
      for maturities between 3 and 30 years.
    """

    for T in range(3, 31):
        r_T_par = swap_rates_dict.get(T, interpolate_swap_rate(T))

        for t in range(3, T):
            if t not in D_t_dict:
                Z_t = interpolate_zero_rate(t, Z_t_dict)
                D_t_dict[t] = math.exp(-Z_t * t / 100)

        sum_D = sum(D_t_dict[t] for t in range(1, T))
        D_T = (1 - r_T_par * sum_D) / (1 + r_T_par)
        Z_T = (-math.log(D_T) / T) * 100

        D_t_dict[T] = D_T
        Z_t_dict[T] = Z_T

        print(f"Term: {T}Y, Swap Rate: {r_T_par*100:.4f}%,
            Discount Factor: {D_T:.8f}, Zero Rate: {Z_T:.8f}%")
```

The above code iteratively computes discount factors and zero rates for maturities ranging from 3 to 30 years using swap rates. It also interpolates missing swap rates where necessary.

Appendix D: Newton-Raphson Method for Refining Discount Factors

The following Python code applies the Newton-Raphson method to refine discount factors based on swap rates. This method is used to iteratively improve the accuracy of the discount factors calculated during the bootstrapping process.

```
import numpy as np
import pandas as pd
from copy import deepcopy
```

```
max_iterations = 1000
tolerance = 1e-9

def newton_raphson_for_discount_factors(df):
    """
    Function to apply the Newton-Raphson method to refine discount factors
    based on swap rates.
    Parameters:
    - df: The DataFrame containing the initial bootstrapped discount factors
      and swap rates.

    Returns:
    - df_copy: A DataFrame with refined discount factors and zero-coupon rates.
    """
    df_copy = deepcopy(df)  # Work on a copy of the original DataFrame

    for index, row in df_copy.iterrows():
        # For tenors >= 3Y, apply Newton-Raphson
        if index >= 16:  # Starting from 3Y
            T = index - 15  # 3Y corresponds to index 16
            quote_rate = row['Quote rate'] / 100  # Convert to decimal
            discount_factor = row['Discount Factor']

            # Define the function and its derivative
            def f(D_T):
                sum_discount = sum(df_copy.loc[i, 'Discount Factor']
                                   for i in range(1, index))
                return (df_copy.loc[1, 'Discount Factor'] - D_T
                        - quote_rate * sum_discount) / (1 + quote_rate)

            def f_prime(D_T):
                return -1 / (1 + quote_rate)

            # Apply Newton-Raphson
            iteration = 0
            while iteration < max_iterations:
                f_value = f(discount_factor)
                f_prime_value = f_prime(discount_factor)

                # Update the discount factor
                new_discount_factor = discount_factor - f_value / f_prime_value

                # Check for convergence
                if abs(new_discount_factor - discount_factor) < tolerance:
                    break

                discount_factor = new_discount_factor
                iteration += 1

            # Update the discount factor in the DataFrame
            df_copy.at[index, 'Discount Factor'] = discount_factor
            df_copy.at[index, 'Zero Rate (%)'] = (-100 * np.log(discount_factor) / df_copy.at[inde

    return df_copy

# Assuming ‘bootstrapped_data‘ is the initial bootstrapped DataFrame before
```

```
  applying Newton-Raphson
refined_df = newton_raphson_for_discount_factors(bootstrapped_data.reset_index()).set_index('Tenor

# Now compare the refined discount factors with the original ones
bootstrapped_data['Discount Factor Refined'] = refined_df['Discount Factor']
bootstrapped_data['Zero Rate (%) Refined'] = refined_df['Zero Rate (%)']
bootstrapped_data['Difference ZC Refined'] = bootstrapped_data['Zero Rate (%) Refined']
                                              - bootstrapped_data['Zero Rate (%)']
bootstrapped_data['Difference DF Refined'] = bootstrapped_data['Discount Factor Refined']
                                              - bootstrapped_data['Discount Factor']
```

The code iteratively applies the Newton-Raphson method to adjust the discount factors for tenors greater than or equal to 3 years. The final results include refined discount factors and zero-coupon rates, which are compared with the initial values.

Appendix E: Cubic Spline Interpolation of Zero Rates

The following Python code performs cubic spline interpolation to smooth the refined zero rates over various maturities. This method provides a continuous curve for zero-coupon rates derived from the refined discount factors.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

# Perform cubic spline interpolation
cs = CubicSpline(bootstrapped_data.reset_index().index,
                 bootstrapped_data['Zero Rate (%) Refined'], bc_type='natural')
new_maturities = np.linspace(0, 28, 100)  # 100 space points for interpolation
interpolated_zero_rates = cs(new_maturities)

# Plot the original zero rates and the interpolated curve
plt.figure(figsize=(10, 6))
plt.plot(bootstrapped_data.index, bootstrapped_data['Zero Rate (%) Refined'], 'ro',
         label='Original Zero Rates')
plt.plot(new_maturities, interpolated_zero_rates, 'b-', label='Cubic
         Spline Interpolation')
plt.xlabel('Maturity (Years)')
plt.xticks(rotation=45)
plt.ylabel('Zero Rate (%)')
plt.title('Cubic Spline Interpolation of Zero Rates')
plt.legend()
plt.grid(True)
plt.show()
```

The cubic spline interpolation is used here to generate a smooth curve of zero rates across a continuous range of maturities. The red points indicate the original refined zero rates, while the blue line represents the interpolated curve. The interpolation provides a more refined visualization of the zero-coupon rates.

1.1. Appendix F: All Curve Plotted

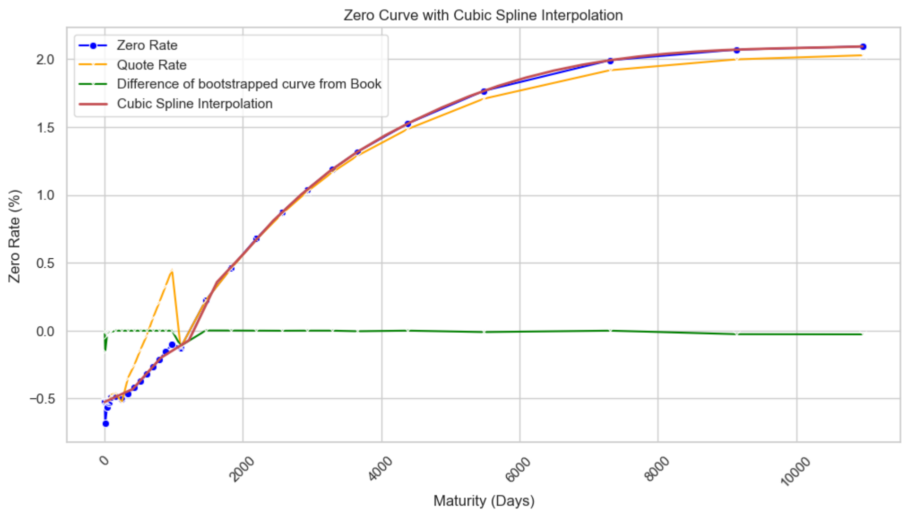

Figure 3: All Curves: Bootstrapped, Market, and Theoretical Curves