Yield Curve Construction

Author: Laura Anedda

In this project, the focus is on the construction of the zero-coupon yield curve by employing a combination of interpolation techniques and bootstrapping. The analysis leverages market data, including swap rates and Euribor rates, to derive insights into the term structure of interest rates and its macroeconomic implications.

Content

Understanding the Zero-Coupon Yield Curve

A yield curve, also known as the term structure of interest rates, illustrates the relationship between interest rates and the time to maturity of debt securities. It is a fundamental tool in financial markets, offering insights into asset pricing and market participants' expectations about future economic conditions.

A **zero-coupon yield curve** specifically represents the yields to maturity of hypothetical zero-coupon bonds across various maturities. Unlike coupon-bearing bonds, zero-coupon bonds pay no periodic interest, making this curve a pure reflection of the time value of money. Each point on the curve corresponds to a discount factor for a given maturity, indicating the present value of a unit of currency at a future date.

Since zero-coupon bonds are not available for all maturities, the zero-coupon yield curve is typically estimated using data from coupon-bearing bonds and other fixed-income instruments, such as interest rate swaps or treasury bills. This estimation involves interpolation and bootstrapping techniques to create a continuous framework essential for pricing fixed-income instruments and managing interest rate risk.

Applications of the Zero-Coupon Yield Curve

The zero-coupon yield curve serves three primary purposes:

- **Asset Pricing and Discounting Cash Flows:** The zero-coupon yield curve is a critical tool for calculating the present value of future cash flows, facilitating accurate pricing of bonds, derivatives, and other financial instruments. By isolating the time value of money, it provides an unbiased discounting framework that excludes the impact of periodic coupon payments. This characteristic makes it particularly useful for theoretical valuations and for developing consistent pricing models across maturities.
- **Understanding Market Expectations**: Constructed from observable market prices of financial instruments, the zero-coupon yield curve reflects collective investor sentiment about future economic conditions, including expectations for interest rates, inflation, and associated risks. Market participants incorporate their expectations into the pricing of these instruments, and by analyzing the curve’s shape and dynamics, these implicit signals can be interpreted effectively. This analytical capability positions the curve as a critical resource for guiding monetary policy, formulating investment strategies, and understanding market sentiment.
- **Risk management:** Financial institutions use the curve to assess and manage interest rate risk. By providing a continuous discounting framework, it enables financial institutions to measure the sensitivity of portfolios to changes in interest rates (e.g., duration and convexity). Furthermore,it plays a pivotal role in stress testing and scenario analysis, allowing institutions to evaluate the potential impact of interest rate fluctuations on their financial stability and performance.

Getting Started- Loading the data and python packages

The construction of a zero-coupon curve requires rates corresponding to each maturity. However, in practice, zero-coupon rates are typically quoted only for short-term maturities, usually up to one year. The choice of reference instruments for these maturities depends on the specific market and context of analysis.

In the Eurozone, the **Euro Interbank Offered Rate (Euribor)**, serves as a reliable benchmark for short-term maturities. Euribor reflects the average interest rate at which major European banks lend to one another in euros, typically covering terms from one week to twelve months. These rates are widely used in financial markets due to their transparency and frequent updates, making them an ideal proxy for zero-coupon rates in the short-term segment.

To estimate zero-coupon rates beyond one year, various financial instruments are employed. Among these, interest rate swaps (IRS) are particularly prominent due to their liquidity, flexibility, and transparent pricing. By applying appropriate interpolation and bootstrapping methodologies, interest rate swaps enable the estimation of zero-coupon rates across an extended maturity spectrum, facilitating the construction of a comprehensive yield curve.

Interest Rate Swaps

An **Interest rate swaps (IRS)** involves an exchange between:

- Fixed-rate payments (r).
- Variable rate payments $(i(t_{i-1}, t_i))$ based on agreed benchmarks such as Euribor.

In a standard or "plain vanilla" swap, the contract specifies the notional principal, payment frequency (e.g., quarterly or semi-annual), tenor, maturity, fixed rate, and floating rate. One party pays a fixed rate, while the other pays a floating rate, both based on the same notional amount,with the floating leg recalibrated periodically according to prevailing market rates. At the stipulation date t_0, to ensure that there are no arbitrage opportunities, the net present value of the swap must be zero:

$$PV^{\text{IRS}} = 0$$

This condition implies that the present value of the fixed leg equals the present value of the variable leg:

$$PV^{\text{Fix}} = PV^{\text{Var}}$$

This equality can be expressed as:

$$\sum_{k=1}^{n} r \cdot (1 + zc_k)^{t_0 - t_k} = \sum_{k=1}^{n} i(t_{k-1}, t_k) \cdot (1 + zc_k)^{t_0 - t_k} \tag{2.1}$$

where zc_k represents the zero-coupon rate for maturity t_k.

The par rate r is the fixed rate in an interest rate swap that balances the present value of the fixed leg with that of the floating leg. It can be determined by solving the following equation:

$$r = \frac{\sum_{k=1}^{n} (1 + zc_k)^{t_0 - t_k}}{\sum_{k=1}^{n} i(t_0, t_{k-1}, t_k) \cdot (1 + zc_k)^{t_0 - t_k}} \tag{2.2}$$

Interest rate swaps offer reliable data for yield curve construction, due to their high liquidity and transparent, frequently updated pricing. Their extensive maturity range, from a few months to over 30 years, enables the creation of a comprehensive curve that incorporates both short- and long-term maturities. Reflecting market consensus on future interest rate movements, swaps provide critical insights into expected interest rate trends and are essential for deriving zero-coupon rates at longer maturities.

2.1. Loading the python packages

In [8]:
```python
#!pip install pandas_market_calendars
#!pip install nelson-siegel-svensson
import numpy as np
import pandas as pd
from pandas.tseries.offsets import CustomBusinessDay
from pandas_market_calendars import get_calendar
import pandas_market_calendars as mcal
from nelson_siegel_svensson import NelsonSiegelSvenssonCurve
from scipy.interpolate import CubicSpline, interp1d
from scipy.optimize import curve_fit
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
```

2.2. Loading and Preparing the Data

Constructing a robust yield curve requires careful preparation and integration of various market rates. In this section, we consolidate and preprocess data from two key sources —Euribor and Swap rates— ensuring that all maturities are represented uniformly and consistently. The resulting dataset will form the foundation for accurate yield curve construction and subsequent analyses.

1. **Loading the Data**: We start by importing two distinct sets of market rates:
   - **Euribor Rates**: Short-term interest rates that reflect the cost of borrowing in the Eurozone's interbank market.
   - **Swap Rates**: Medium- to long-term interest rates derived from interest rate swaps, providing insights into market expectations over longer horizons.

   By incorporating both datasets, we capture a comprehensive range of maturities essential for constructing a yield curve that spans from short- to long-term segments.

2. **Standardizing Maturity Periods**:\

Market data often expresses maturities in multiple formats (e.g., months as `3M` or `6M`, and years as `1Y` or `2Y`). To ensure analytical consistency:

- **Months to Years**: Maturities given in months are converted by dividing by 12 (e.g., `3M` → 0.25 years).
- **Years to Numerical Values**: Maturities denoted in years are directly converted to their numerical equivalent (e.g., `1Y` → 1 year).
- **Unlabeled Units**: Any maturity lacking a unit is assumed to be in years and is cast as a floating-point number.

Applying this standardized conversion function to both the Euribor and Swap dataframes ensures that all maturities are represented uniformly in years. Following this step, both datasets are sorted by their numerical maturity values to maintain a logical progression along the time axis.

4. **Combining Euribor and Swap Data**:
   With both datasets now consistently formatted and sorted, we concatenate them into a unified `market_rates` dataframe. This single dataset seamlessly integrates short-term Euribor data with longer-term Swap rates, offering a comprehensive view of the interest rate environment. Any duplicate maturities arising from the concatenation are removed, preserving only the first occurrence and ensuring data integrity.

5. **Classifying the Source of Each Rate**:\

To distinguish between different segments of the yield curve, we add a `Source` column to the combined `market_rates` dataframe:

- **Euribor**: Assigned to shorter maturities, capturing the early portion of the curve.
- **Swap**: Assigned to the medium- and long-term maturities, reflecting market expectations further into the future.\

This classification is designed to facilitate the bootstrapping process by distinguishing between short-term Euribor and medium/long-term swap rates, which can be handled differently in subsequent yield curve calculations.

In [10]:
```python
euribor_data = pd.read_csv("Data/Euribor_rates.csv", sep = ';')
swap_data = pd.read_csv("Data/Swaps_rates.csv", sep = ';')

euribor_data.set_index('Maturity', inplace=True)
swap_data.set_index('Maturity', inplace=True)

def maturity_to_years(maturity):
    """
    Converts maturity periods to numerical values in years.
    """
    if maturity[-1] == 'M':
        return int(maturity[:-1]) / 12  # Months to years
    elif maturity[-1] == 'Y':
        return int(maturity[:-1])       # Years
    else:
        return float(maturity)          # Assume already in years if no unit

# Apply maturity conversion and sort for both Euribor and Swap data
euribor_data.index = euribor_data.index.map(maturity_to_years)
euribor_data = euribor_data.sort_index()
swap_data.index = swap_data.index.map(maturity_to_years)
swap_data = swap_data.sort_index()
```

In [11]:
```python
# Concatenate Euribor and Swap data, removing duplicates
market_rates = pd.concat([euribor_data, swap_data], axis=0)
market_rates = market_rates[~market_rates.index.duplicated(keep='first')]

#Add 'Source' column to classify the origin of each interpolated rate based on maturity.
#The `Source` is set to "Euribor" for short-term maturities and "Swap" for medium/long-term maturities.
market_rates['Source'] = ''                                 # Initialize an empty column
market_rates.loc[euribor_data.index, 'Source'] = 'Euribor'  # Assign 'Euribor' for Euribor indices
market_rates.loc[swap_data.index, 'Source'] = 'Swap'        # Assign 'Swap' for Swap indices
#for 1st 3 row (short-term rates)
market_rates.iloc[:4, market_rates.columns.get_loc('Source')] = 'Euribor'
#for the rest (medium to long-term rates)
market_rates.iloc[4:, market_rates.columns.get_loc('Source')] = 'Swap'

market_rates.head(20)
```

Out[11]:

	Rate	Source
Maturity		
0.083333	3.092	Euribor
0.250000	3.057	Euribor
0.500000	2.923	Euribor
1.000000	2.630	Euribor
2.000000	2.449	Swap
3.000000	2.419	Swap
4.000000	2.415	Swap
5.000000	2.415	Swap
6.000000	2.421	Swap
7.000000	2.431	Swap
8.000000	2.440	Swap
9.000000	2.456	Swap
10.000000	2.434	Swap
11.000000	2.445	Swap
12.000000	2.486	Swap
15.000000	2.462	Swap
20.000000	2.416	Swap
25.000000	2.265	Swap
30.000000	2.185	Swap

Yield Curve Construction: Parametric and Interpolation Models

Yield curve construction is a fundamental process in financial markets, which provides a continuous representation of interest rates across various maturities. However, the data available in the market is often discrete, requiring mathematical techniques to create a smooth and continuous yield curve. Indeed, in financial markets, swap rates are quoted for select maturities. Typically, **we find quoted swap rates for**:

- Annually from **1 to 10 years**,
- Selected **maturities at 12, 15, 25, and 50 years**,
- Additional **long-term maturities at 30, 40, and 50 years**.

These quoted rates, while essential, provide only a fragmented view of the yield curve, insufficient for comprehensive applications.
Bootstrapping, the process of deriving zero-coupon rates from observed market data, **relies on these discrete data points**. However, **additional methods are required to bridge the gaps between quoted maturities, ensuring a smooth and continuous curve**. This step is essential for accurately discounting cash flows across various time horizons and for applications such as pricing, risk management, and portfolio optimization.

To ensure a complete representation of the yield curve, two distinct approaches are commonly employed: **interpolation methods** and **parametric models**.

- **Interpolation** is a mathematical technique essential for **estimating unknown values that lie between known data points**. In yield curve construction, interpolation provides a practical solution by **filling in gaps between quoted maturities**, ensuring **a smooth and continuous representation of interest rates**. It is particularly effective for constructing local segments of the curve, where the focus is on achieving precision and maintaining alignment with observed data.
- **Parametric models**, in contrast, take a holistic approach by **defining the entire yield curve using a mathematical formula with a set of interpretable parameters**. This approach not only **smooths the gaps** between data points but also **allows for extrapolation beyond observed maturities**, offering insights into the broader shape and dynamics of the term structure. Parametric models are particularly valuable for applications requiring a macroeconomic perspective or for projecting long-term trends.

These two approaches represent **distinct strategies for addressing the same challenge**: constructing a continuous yield curve from fragmented data.
Interpolation methods are ideal for scenarios where exact data fitting and local precision are paramount, while **parametric models** provide greater flexibility for modeling structural trends and extending the curve beyond the range of available market quotes. The choice between these options depends on the specific analytical objectives, whether the priority is accurate alignment with existing data or a more comprehensive and extrapolative representation of the yield curve.

Interpolation Methods for Yield Curve Construction

Various interpolation methods are used, including linear, quadratic, and cubic splines. Each technique offers a trade-off between simplicity, computational efficiency, and the smoothness of the resulting curve.

Linear Interpolation

Linear interpolation assumes the change between two known points to be linear, meaning it follows a straight line. It calculates the interpolated value as a weighted average based on the relative distance of the target point between two adjacent known points. Considering the swap rate i_{SW_i} at maturity t_i as unknown and positioned between t_{i-1} and t_{i+1} (the maturities of known swap rates), the relationship between these rates is given by:

$$i_{SW_i} = i_{SW_{i-1}} + \frac{i_{SW_{i+1}} - i_{SW_{i-1}}}{t_{i+1} - t_{i-1}} \cdot (t_i - t_{i-1}) \tag{3.1}$$

While linear interpolation is computationally efficient and easy to implement, it may not adequately capture the nuanced behavior of the yield curve, particularly over longer horizons.

Quadratic Spline Interpolation

Quadratic spline interpolation introduces curvature by fitting a second-degree polynomial between each pair of consecutive data points, allowing for a smoother transition that better reflects the dynamics of the yield curve. This method ensures continuity in both the function value and its first derivative, avoiding abrupt changes in slope.

For an interest rate i_{SW_i} at time t_i , the quadratic spline interpolation can be expressed as:

$$i_{SW_i} = i_{SW_{i-1}} + \frac{i_{SW_{i+1}} - i_{SW_{i-1}}}{t_{i+1} - t_{i-1}} \cdot (t_i - t_{i-1}) + \alpha(t_i - t_{i-1})^2 \tag{3.2}$$

The term $\alpha(t_i - t_{i-1})^2$ adds curvature into the spline, enhancing its flexibility compared to a linear fit.

Quadratic splines offer a balance between simplicity and accuracy, providing smooth transitions with moderate computational demands, using only three data points per segment. This makes them ideal for applications requiring efficient calculations without sacrificing reliability.

Cubic Spline Interpolation

In **cubic spline interpolation**, a cubic polynomial is constructed for each interval between consecutive data points, ensuring a smooth and continuous curve.
Given $n+1$ points, (t_0, i_{SW_0}),(t_1, i_{SW_1}),...,(t_n, i_{SW_n}), we define a cubic polynomial $S_i(t)$ on each interval $[t_i, t_{i+1}]$, with each $S_i(t)$ taking the form:

$$S_i(t) = a_i + b_i(t - t_i) + c_i(t - t_i)^2 + d_i(t - t_i)^3$$

Where a_i,b_i,c_i and d_i are coefficients that ensure smoothness and continuity of the spline across intervals. The process ensures that the curve passes through all the data points while maintaining smooth transitions between intervals.

The primary advantage of cubic splines lies in their ability to generate a very smooth curve. This is achieved by ensuring the continuity of the second derivative at all interior points, thereby eliminating abrupt changes or "kinks" that may occur in linear or quadratic interpolation. This characteristic makes cubic splines particularly effective for modeling natural variations, such as interest rate movements over time. The resulting continuity in both the first and second derivatives produces a visually appealing curve that accurately captures the underlying data trends.

For an interest rate i_{SW_i} at time t_i , the cubic spline interpolation can be generalized as follows:

$$i_{SW_i} = i_{SW_{i-1}} + \frac{i_{SW_{i+1}} - i_{SW_{i-1}}}{t_{i+1} - t_{i-1}} \cdot (t_i - t_{i-1}) + \beta(t_i - t_{i-1})^2 + \gamma(t_i - t_{i-1})^3 \tag{3.3}$$

where β and γ are additional coefficients derived from solving a system of equations. These equations enforce:

- Continuity of the first derivative across intervals, ensuring smooth transitions.
- Continuity of the second derivative, maintaining the curve's natural flow.

These coefficients depend on boundary conditions and the values of the data points, making cubic splines an optimal choice for interpolating yield curves in financial markets where smooth and accurate representation of interest rate variations is critical.

Parametric models for Yield Curve Construction: Nelson-Siegel-Svensson (NSS) model

Interpolation methods are effective for creating smooth transitions between known rates but often lack the flexibility needed to represent complex term structures or extrapolate beyond observed maturities. To address these limitations, parametric models offer an alternative by fitting the yield curve with a predefined functional form. Unlike interpolation, which strictly adheres to the given data points, parametric models optimize a set of

parameters to capture the broader shape and dynamics of the yield curve, producing a continuous representation of the term structure.
Among the most prominent parametric models is the **Nelson-Siegel framework** and its extended version, the **Nelson-Siegel-Svensson (NSS) model**.

The NSS model expresses the yield curve as a combination of level, slope, and curvature components, providing a smooth and flexible representation of the term structure:

$$y(t) = \beta_0 + \beta_1 \frac{1 - e^{-t/\tau_1}}{t/\tau_1} + \beta_2 \left(\frac{1 - e^{-t/\tau_1}}{t/\tau_1} - e^{-t/\tau_1} \right) + \beta_3 \left(\frac{1 - e^{-t/\tau_2}}{t/\tau_2} - e^{-t/\tau_2} \right)$$

Here:

- β_0 represents the **long-term level of interest rate**.
- β_1 determines the **initial slope of the curve**.
- β_2 captures the **medium-term curvature**.
- β_3 adds an **additional curvature component**, allowing for more complex shapes.
- τ_1,τ_2 **decay factors** dictating the exponential decline of each term.

By introducing the additional parameter β_3, the NSS model is able to accommodate more complex shapes in the yield curve, such as multiple humps or troughs, often observed in real-world market data. This flexibility makes NSS a preferred choice in financial markets for pricing, risk management, and policy analysis.

The model's **parameters** are initialized to capture the economic dynamics of interest rates effectively: β_0 represents the long-term interest rate level, often initialized as the average of observed long-term rates, capturing the stable level rates tend to approach over time. β_1 determines the initial slope of the curve, derived as the difference between the short-term rate and the long-term level, ensuring the model reflects the curve's initial steepness. β_2 and β_3 control medium- and long-term curvature, starting with small values to introduce flexibility while maintaining stability during optimization. τ_1 and τ_2 are decay factors dictating the exponential decline of each term, set to align with typical medium- and long-term maturities.
These parameters are inputs for a non-linear optimization process that adjusts them to minimize deviations from observed rates, resulting in a smooth, economically meaningful yield curve.

The difference between observed and estimated yields is quantified using the sum of squared errors (SSE), defined as:

$$SSE = \sum_{i=1}^{N} (r(t_i) - y(t_i))^2$$

where $r(t_i)$ represents the observed yield for a given maturity, and $y(t_i)$ is the yield predicted by the model. The SSE serves as the objective function to be minimized.

The optimization algorithm, typically a gradient-based method like L-BFGS-B, adjusts all parameters simultaneously to minimize the total error. At each iteration, parameters are updated, yields recalculated, and the error reevaluated. This process repeats until the error stabilizes, ensuring convergence to an optimal parameter set. By optimizing across all maturities simultaneously, the NSS model provides a cohesive and accurate representation of the yield curve.

The parameters of the model are intuitively linked to economic factors, enabling practitioners to interpret shifts in the curve as changes in long-term rates, short-term trends, or medium-term market dynamics. The NSS model thus strikes a balance between mathematical rigor and practical applicability, positioning it as a cornerstone of modern yield curve modeling.

Practical Considerations: Business and Settlement Days

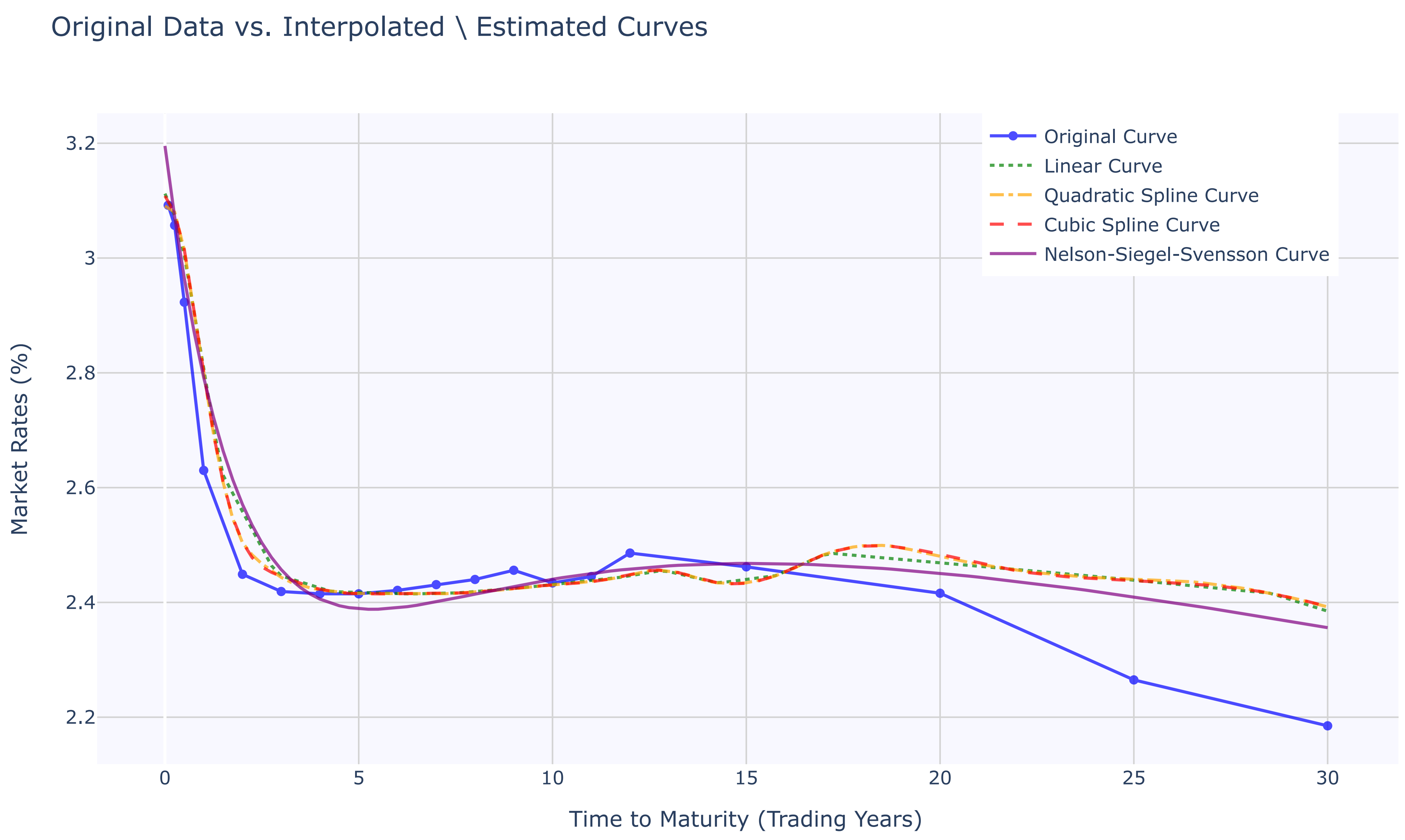
When constructing a yield curve and interpolating financial data, it is crucial to consider **business days** and **settlement days**.

- **Business days:** These refer to days when financial markets operate, excluding weekends and holidays. They are essential for accurate duration calculations and interest rate computations. Ignoring business days can introduce errors, particularly for short-term instruments or periods spanning non-market days.
- **Settlement days:** These define when a transaction is settled and ownership is transferred, impacting the cash flow timing and discounting. Accurate handling of settlement conventions ensures proper alignment of maturities and cash flows.

Ignoring these factors can distort yield curve construction, leading to inaccuracies in interpolated rates and subsequent applications like pricing and risk management.

In [14]:
```python
class YieldCurveFitter:
    def __init__(self, market_rates, base_date='2024-11-05', evaluation_date='2024-11-05'):
        """
        Initialize the yield curve fitter with market rates, base date, and evaluation date.

        Args:
        - market_rates (pd.DataFrame): A DataFrame containing 'Rate' and 'Source' columns,
                                       indexed by maturities in years.
        - base_date (str or pd.Timestamp): The start date for maturity calculations.
        - evaluation_date (str or pd.Timestamp): The date used to evaluate remaining maturities.

        Purpose:
        This class supports both interpolation and parametric methods to fit yield curves. It includes:
        - Interpolation Methods: Linear, Quadratic Spline, and Cubic Spline for generating smooth yield curves.
        - Parametric Method: Nelson-Siegel-Svensson for estimating yield curves using functional forms.
        - Integration of a trading calendar and settlement conventions for accurate maturity handling.
        """
        if 'Rate' not in market_rates.columns:
            raise ValueError("market_rates must contain a 'Rate' column.")
        if 'Source' not in market_rates.columns:
            raise ValueError("market_rates must contain a 'Source' column.")

        # Input data for interpolation
        self.market_rates = market_rates.dropna(subset=['Rate'])   # Ensure no NaN rates
        # Date references for calculations
        self.base_date = pd.Timestamp(base_date)
        self.evaluation_date = pd.Timestamp(evaluation_date)
        # Placeholder for estimated data results
        self.fitted_data = None

    def calculate_remaining_maturities(self):
        """
        Calculate remaining maturities in trading years based on the given calendar and settlement rules.

        Steps:
        1. **Generate Business Dates**:
            - For each maturity in years, convert it to days by multiplying by 365.
            - Add these days to the base_date to obtain the corresponding business dates.
            - This handles fractional years by approximating them to the nearest day.

        2. **Apply T+2 Settlement Rule**:
            - For each business date, add two business days to account for the settlement period.
            - This uses a custom business day offset based on the Euronext Amsterdam (XAMS) trading calendar.

        3. **Compute Remaining Maturities**:
            - Calculate the number of trading days in a standard year based on the XAMS calendar.
            - For each settlement date, compute the difference in days from the evaluation_date.
            - Divide the day difference by the trading_days_per_year to obtain the remaining maturities in trading years.

        Returns:
            - maturities: List of remaining maturities expressed in trading years.
        """
        # Load Euronext Amsterdam calendar (XAMS)
        calendar = mcal.get_calendar('XAMS')
        business_day_offset = CustomBusinessDay(calendar=calendar)  # Custom business days using XAMS calendar
        business_dates = [self.base_date + pd.Timedelta(days=int(d * 365))  # Use days to handle fractional years
                          for d in self.market_rates.index]

        # Apply T+2 settlement rule by adding two business days to each business date
        settlement_dates = [ b_date + 2 * business_day_offset for b_date in business_dates]
        # Determine trading days per year using the XAMS calendar
        trading_days_per_year = len(calendar.valid_days(self.base_date, self.base_date + pd.DateOffset(years=1)))

        # Calculate remaining maturities in trading years
        maturities = [(d - self.evaluation_date).days / trading_days_per_year for d in settlement_dates]
        return maturities

    def fit_yields(self, method, num_points=121):
        """
        Fit yields using the selected method.

        Args:
        - method: Interpolation ('Linear', 'Quadratic Spline', 'Cubic Spline') and Parametric ('Nelson-Siegel-Svensson') met
        - num_points: Number of points at which to evaluate the fitted curve.

        Returns:
        - DataFrame with interpolated or estimated rates.
        """
        x = self.calculate_remaining_maturities() # Independent variable (remaining maturities in trading years)
        y = self.market_rates['Rate'].values      # Dependent variable (market rates)
        x_vals = np.linspace(0, 30, num_points)   # Evaluate the curve up to 30 trading years, adjustable

        # Select interpolation method and compute rates
        if method in ['Linear', 'Quadratic Spline', 'Cubic Spline']:
            interpolator = {
                'Linear': lambda: interp1d(x, y, kind='linear', fill_value='extrapolate'),
                'Quadratic Spline': lambda: interp1d(x, y, kind='quadratic', fill_value='extrapolate'),
                'Cubic Spline': lambda: CubicSpline(x, y, bc_type='natural')
            }[method]()  # Get the appropriate interpolator
            fitted_rates = interpolator(x_vals)  # Apply interpolation
            result_col_name = f'Interpolated_Yield ({method})'
        # Select parametric method and compute rates
        elif method == 'Nelson-Siegel-Svensson':
            # Nelson-Siegel-Svensson parametric model
            def nss_model(t, beta0, beta1, beta2, beta3, tau1, tau2):
                return NelsonSiegelSvenssonCurve(beta0, beta1, beta2, beta3, tau1, tau2)(t)

            # Initial parameter guesses for NSS model
            beta0 = np.mean(y[-5:])     # Long-term average
            beta1 = y[0] - beta0        # Short-term impact
            beta2, beta3 = 0.01, 0.005  # Initial guesses for curvature terms
            tau1, tau2 = 2.0, 10.0      # Initial guesses for decay factors
            p0 = [beta0, beta1, beta2, beta3, tau1, tau2]  # Initial parameter values
            # Fit NSS model to the data and compute estimated rates
            params, _ = curve_fit(nss_model, x, y, p0=p0)
            fitted_rates = nss_model(x_vals, *params)
            result_col_name = f'Estimated_Yield ({method})'

        else:
            raise ValueError("Invalid method. Choose 'Linear', 'Quadratic Spline', 'Cubic Spline', or 'Nelson-Siegel-Svenss

        # Create a DataFrame for the fitted yields
        self.fitted_data = pd.DataFrame({result_col_name: fitted_rates}, index=x_vals)

        # Classify the 'Source' of rates as 'Euribor' for short-term maturities and 'Swap' for medium/long-term
        self.fitted_data['Source'] = np.where(self.fitted_data.index <= max(euribor_data.index), 'Euribor', 'Swap')
        self.fitted_data.index.name = 'Remaining_Maturity(Years)'

        return self.fitted_data

    def plot_comparison(self):
        """
        Plot the original market rates against the fitted curves for visual comparison.
        It includes linear, quadratic, cubic interpolation, and Nelson-Siegel-Svensson.
        """
        fig = go.Figure()

        # Plot original rates data
        fig.add_trace(go.Scatter(
            x=self.market_rates.index, y=self.market_rates['Rate'],
            mode='lines+markers', line=dict(color='blue'),
            name='Original Curve',
            marker=dict(symbol='circle', size=6),
            opacity=0.7
        ))

        methods = ['Linear', 'Quadratic Spline', 'Cubic Spline', 'Nelson-Siegel-Svensson']
        colors = ['green', 'orange', 'red', 'purple']
        dash_styles = ['dot', 'dashdot', 'dash', 'solid']

        # Plot each interpolation/estimation method
        for method, color, dash_style in zip(methods, colors, dash_styles):
            fitted_data = self.fit_yields(method=method)
            col_name = [c for c in fitted_data.columns if 'Yield' in c][0]

            # Plot the curve
            fig.add_trace(go.Scatter(
                x=fitted_data.index,
                y=fitted_data[col_name],
                mode='lines',
                name=f'{method} Curve',
                line=dict(dash=dash_style, color=color),
                opacity=0.7
            ))

        fig.update_layout(
            title='Original Data vs. Interpolated \ Estimated Curves',
            xaxis_title='Time to Maturity (Trading Years)',
            yaxis_title='Market Rates (%)',
            width=1000,
            height=600,
            showlegend=True,
            plot_bgcolor='rgb(248, 248, 255)', paper_bgcolor='white',
            legend=dict(x=0.68, y=1),
            xaxis=dict(showgrid=True, gridwidth=0.4, gridcolor='LightGray', tickmode='auto'),
            yaxis=dict(showgrid=True, gridwidth=0.4, gridcolor='LightGray'),
        )

        fig.show()


# Fit the yields
fitter = YieldCurveFitter(market_rates=market_rates)
linear_data = fitter.fit_yields(method='Linear')
```

```
quadratic_data = fitter.fit_yields(method='Quadratic Spline')
cubic_data = fitter.fit_yields(method='Cubic Spline')
nss_data = fitter.fit_yields(method='Nelson-Siegel-Svensson')

fitter.plot_comparison()
```



Bootstrapping: A Key Technique for Yield Curve Construction

The **bootstrapping procedure** is a systematic approach used to derive **zero-coupon rates** for maturities beyond one year. It does so by starting from instruments with periodic coupon payments (such as interest rate swaps) and progressively stripping away coupon effects until the underlying zero-coupon yields are obtained. This approach ensures that the resulting yield curve consistently reflects market expectations for each discrete maturity.

In the case of an **interest rate swap**,the notional principal is not physically exchanged, but the **fixed leg can be mathematically modeled as a fixed-rate bond**. This is because both involve a series of fixed cash flows discounted at appropriate rates, enabling consistent valuation and comparison.

The bootstrapping process involves constructing a series of **hypothetical coupon bonds**, each:

- **Priced at par** (i.e., with a present value equal to the notional).
- **Pays a coupon equal to the swap rate**.
- Assuming a **notional principal of 100**.

For each maturity, we imagine issuing such a bond. By equating its present value (PV) to its par value, we can solve for the zero-coupon rate that discounts its cash flows correctly. In other words, each hypothetical bond reveals a pure spot rate at that maturity, free from the averaging effects of the coupon structure.

Consider the **first maturity**, at which there is only one payment date. On this date, the coupon plus the principal are paid together ($100 \cdot (1 + i_{sw_1})$). To value this one-period bond at par (100), we must discount that cash flow by the **first zero-coupon rate** i_{zc_1}:

$$\frac{100 \cdot (1 + i_{sw_1})}{(1 + i_{zc_1})} = 100 \tag{4.1}$$

Here:

- i_{sw_1} : The first swap rate (coupon rate for the bond).
- i_{zc_1} : The first zero-coupon rate (to be calculated).
- 100 : The assumed notional principal.

The **left-hand side** represents the **present value of the bond's cash flow** (coupon + principal) discounted at the first zero-coupon rate. Since the bond is priced at par, this value **must equal its notional principal of 100**.

Rearranging and solving for i_{zc_1} it yields:

$$i_{zc_1} = i_{sw_1}$$

This result is intuitive: **for the first period, the zero-coupon rate is equal to the first swap rate.**

Following the determination of the first zero-coupon rate i_{zc_1}, **a recursive method** is applied to derive the subsequent zero-coupon rates.

To derive the **pure spot rate** for a **two-year maturity**, **we construct a hypothetical bond with a coupon fixed at the two-year swap rate** i_{sw_2}. The first cash flow is discounted using i_{zc_1}, reflecting the market's one-year expectations, while the second cash flow is discounted using the unknown two-year zero-coupon rate, i_{zc_2}.
The relationship is expressed as:

$$100 = \frac{100 \cdot i_{sw_2}}{1 + i_{zc_1}} + \frac{100 \cdot (1 + i_{sw_2})}{(1 + i_{zc_2})^2} \tag{4.2}$$

Here, the **two-year swap rate**, i_{sw_2}, represents the average expectations of rates over both the first and second years. It **does not isolate the rate for just the second year**, which is why further calculations are necessary to extract the pure spot rate for the two-year maturity.
Solving the Equation (4.2) for i_{zc_2} we extract the second year's pure spot rate –independent of the first year's rate or coupon effects–, thereby refining the yield curve so that each zero-coupon rate aligns purely with market expectations for its specific maturity.

This logic extends iteratively to longer maturities. For a three-year maturity, the equation would be:

$$100 = \frac{100 \cdot i_{sw_3}}{1 + i_{zc_1}} + \frac{100 \cdot i_{sw_3}}{(1 + i_{zc_2})^2} + \frac{100 \cdot (1 + i_{sw_3})}{(1 + i_{zc_3})^3} \tag{4.3}$$

More generally, for an n-year maturity:

$$100 = \frac{100 \cdot i_{sw_n}}{1 + i_{zc_1}} + \frac{100 \cdot i_{sw_n}}{(1 + i_{zc_2})^2} + \cdots + \frac{100 \cdot (1 + i_{sw_n})}{(1 + i_{zc_n})^n} \tag{4.4}$$

This process finally results in the following **general formula** for computing the zero-coupon rate i_{zc_i} at the i-th maturity:

$$i_{zc_i} = \left(\frac{1 + i_{sw_i}}{1 - \sum_{j=1}^{i-1} \frac{i_{sw_j}}{(1+i_{zc_j})^j}} \right)^{\frac{1}{i}} - 1, \quad i = 1, 2, \ldots, n \tag{4.5}$$

Once interpolation of the observed market rates is completed, the **bootstrapping procedure enables the construction of the entire zero-coupon yield curve**. By leveraging the interpolated values alongside this systematic approach, zero-coupon rates for all maturities can be derived. This **ensures a smooth and continuous yield curve that accurately captures market expectations for each specific period**, providing a reliable foundation for pricing and risk management of fixed-income securities.

In [17]:
```python
class ZeroCouponBootstrap:
    def __init__(self):
        """
        The ZeroCouponBootstrap class computes zero-coupon curves from previously fitted yield curves.

        """
        # Store input data for all interpolation and parametric methods
        self.data_dict = {
            'Linear': linear_data,
            'Quadratic Spline': quadratic_data,
            'Cubic Spline': cubic_data,
            'Nelson-Siegel-Svensson': nss_data
        }
        # Store the computed bootstrapped zero-coupon curves for each method used
        self.results = {}
        self.calculate_curves()

    def calculate_curves(self):
        """
        Compute the zero-coupon yields for each method in data_dict.
        This method iterates over each provided method's DataFrame.

        For each maturity, if the 'Source' is:
        - 'Euribor', the rate is assumed to represent a zero-coupon yield directly.
```

```
          and is appended to the zero_coupon_bonds list without modification.
        - 'Swap', a bootstrapping formula is applied to derive zero-coupon yields from swap rates,
          considering previously calculated zero-coupon yields.
        """
        for method, data in self.data_dict.items():
            zero_coupon_bonds = []
            euribor_count = 0

            for idx, row in data.iterrows():
                T = row.name   # Maturity in years
                rate = row[f'Interpolated_Yield ({method})'
                            if method != 'Nelson-Siegel-Svensson'
                            else f'Estimated_Yield ({method})']
                source = row['Source']

                if source == 'Euribor':
                    # Directly add Euribor rates to zero-coupon bond list
                    zero_coupon_bonds.append(rate)
                    euribor_count += 1
                else:
                    # Calculate the zero-coupon rate for Swap sources
                    _sum = sum(
                        rate / (1 + zero_coupon_bonds[j])**(j)
                        for j in range(euribor_count, len(zero_coupon_bonds))
                    )
                    # NOTE: The current formula for calculating the z.c.r. from Swap rates is adjusted for quarterly data.
                    # Since the data is quarterly, the formula does not include the exponent **1/T.
                    zero_coupon_rate = ((1 + rate) / (1 - _sum) ) - 1
                    zero_coupon_bonds.append(round(zero_coupon_rate, 5))

            # Add the zero-coupon bond rates to the DataFrame
            data['Zero_Coupon_Bond'] = zero_coupon_bonds
            cols = [col for col in data.columns if col != 'Source'] + ['Source']
            self.data_dict[method] = data[cols]
            self.results[method] = data

    def plot_comparison(self):
        """
        Plot the bootstrapped zero-coupon bond curves against their corresponding fitted yield curves for each method.

        For each method, two lines are plotted:
            - Bootstrapped Curve (Zero_Coupon_Bond)
            - Fitted Curve (Interpolated or Estimated Yield)
        """
        methods = list(self.results.keys())
        rows, cols = 2, 2
        fig = make_subplots(rows=rows, cols=cols, subplot_titles=methods,
                            vertical_spacing=0.1, horizontal_spacing=0.08)

        for idx, method in enumerate(methods, start=1):
            row, col = divmod(idx - 1, cols)
            data = self.results[method]

            # Bootstrapped curve
            fig.add_trace(
                go.Scatter(
                    x=data.index, y=data[f'Zero_Coupon_Bond'],
                    mode='lines+markers', marker=dict(size=5), line=dict(width=1.5, color='blue'),
                    name='Bootstrapped Curve',
                    showlegend=(idx == 3)
                ),
                row=row + 1, col=col + 1
            )

            # Interpolated / Estimated curve
            fig.add_trace(
                go.Scatter(
                    x=data.index,
                    y=data[f'Interpolated_Yield ({method})' if method != 'Nelson-Siegel-Svensson'
                                else f'Estimated_Yield ({method})'],
                    mode='lines', line=dict(dash='dash', width=1.5, color='red'),
                    name='Fitted Curve',
                    showlegend=(idx == 3)
                ),
                row=row + 1, col=col + 1
            )

        fig.update_layout(
            title='Bootstrapped Zero-Coupon Bond Curves vs. Fitted Curves',
            height=800, width=1000,
            plot_bgcolor='rgb(248, 248, 255)',
            legend=dict(x=0.85,  y=1, orientation="v")
        )
        fig.update_xaxes(title_text='Time to Maturity (Years)', showgrid=True, gridwidth=0.4, gridcolor='LightGray')
        fig.update_yaxes(title_text='Zero Rates (%)', showgrid=True, gridwidth=0.4, gridcolor='LightGray')

        fig.show()
```

In [18]:
```
bootstrapper = ZeroCouponBootstrap()

for method, df in bootstrapper.data_dict.items():
    print("\n" + "-" * 90)
    if method == "Nelson-Siegel-Svensson":
        print(f"\033[1mESTIMATION METHOD: {method}\033[0m")
    else:
        print(f"\033[1mINTERPOLATION METHOD: {method}\033[0m")
    display(df)
```

```
------------------------------------------------------------------------------------------
INTERPOLATION METHOD: Linear
```

	Interpolated_Yield (Linear)	Zero_Coupon_Bond	Source
Remaining_Maturity(Years)			
0.00	3.112169	3.112169	Euribor
0.25	3.074203	3.074203	Euribor
0.50	3.005462	3.005462	Euribor
0.75	2.910191	2.910191	Euribor
1.00	2.807721	2.807721	Euribor
...	...	...	...
29.00	2.405911	2.422750	Swap
29.25	2.400619	2.417390	Swap
29.50	2.395326	2.412040	Swap
29.75	2.390034	2.406680	Swap
30.00	2.384742	2.401330	Swap

121 rows × 3 columns

```
------------------------------------------------------------------------------------------
INTERPOLATION METHOD: Quadratic Spline
```

	Interpolated_Yield (Quadratic Spline)	Zero_Coupon_Bond	Source
Remaining_Maturity(Years)			
0.00	3.090583	3.090583	Euribor
0.25	3.080143	3.080143	Euribor
0.50	3.013777	3.013777	Euribor
0.75	2.909694	2.909694	Euribor
1.00	2.801100	2.801100	Euribor
...	...	...	...
29.00	2.409171	2.426440	Swap
29.25	2.405263	2.422480	Swap
29.50	2.401132	2.418300	Swap
29.75	2.396777	2.413890	Swap
30.00	2.392199	2.409260	Swap

121 rows × 3 columns

```
------------------------------------------------------------------------------------------
INTERPOLATION METHOD: Cubic Spline
```

	Interpolated_Yield (Cubic Spline)	Zero_Coupon_Bond	Source
Remaining_Maturity(Years)			
0.00	3.108280	3.108280	Euribor
0.25	3.078013	3.078013	Euribor
0.50	3.014904	3.014904	Euribor
0.75	2.908957	2.908957	Euribor
1.00	2.797572	2.797572	Euribor
...	...	...	...
29.00	2.409483	2.426750	Swap
29.25	2.405620	2.422840	Swap
29.50	2.401476	2.418650	Swap
29.75	2.397070	2.414190	Swap
30.00	2.392422	2.409480	Swap

121 rows × 3 columns

```
------------------------------------------------------------------------------------------
ESTIMATION METHOD: Nelson-Siegel-Svensson
```

	Estimated_Yield (Nelson-Siegel-Svensson)	Zero_Coupon_Bond	Source
Remaining_Maturity(Years)			
0.00	3.195375	3.195375	Euribor
0.25	3.071867	3.071867	Euribor
0.50	2.964751	2.964751	Euribor
0.75	2.872067	2.872067	Euribor
1.00	2.792076	2.792076	Euribor
...	...	...	...
29.00	2.367459	2.383270	Swap
29.25	2.364630	2.380410	Swap
29.50	2.361782	2.377530	Swap
29.75	2.358916	2.374630	Swap
30.00	2.356032	2.371720	Swap

121 rows × 3 columns

Visualizing the Yield Curve

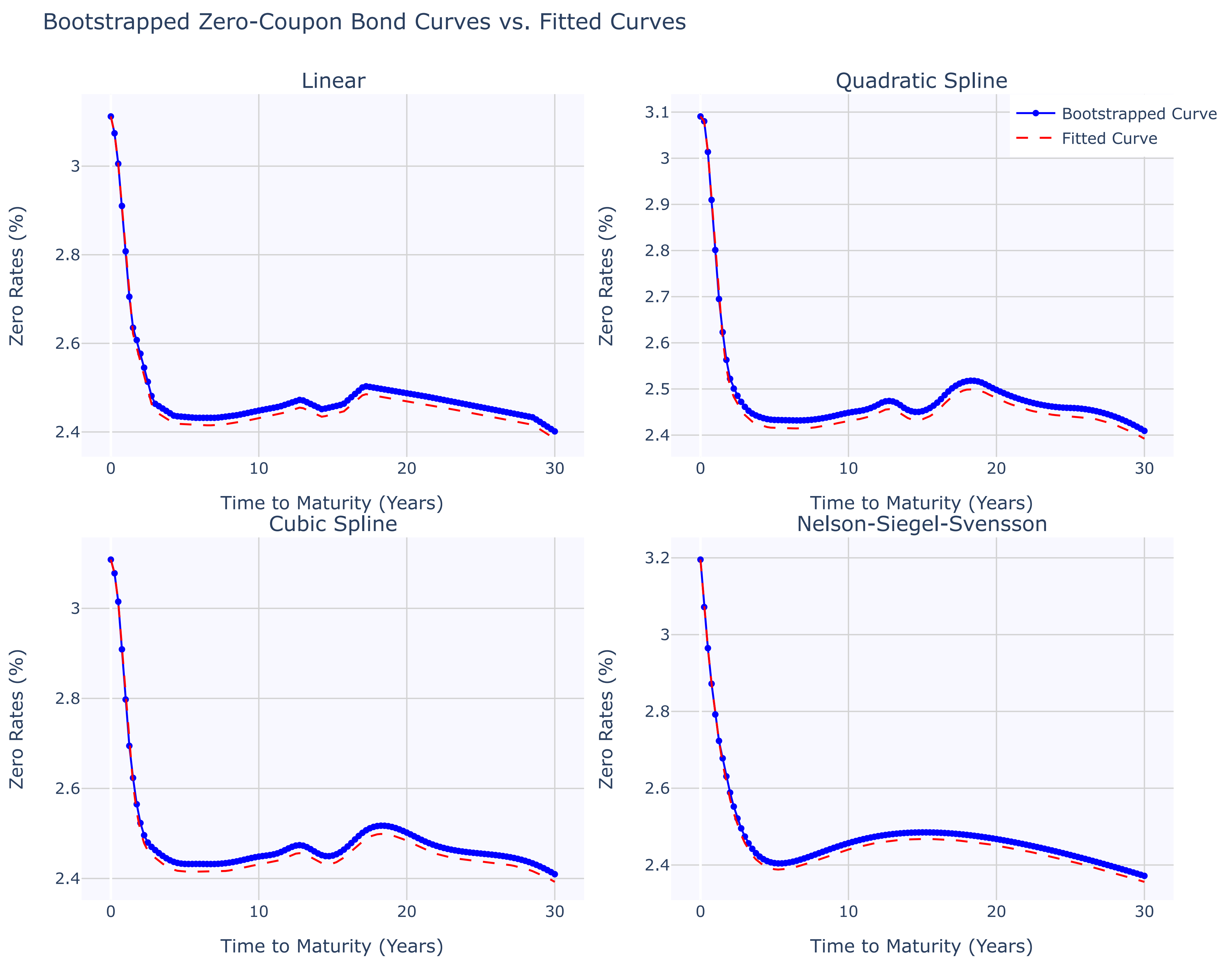
The chart below compares the bootstrapped zero-coupon bond curves (solid blue) with the interpolated curves (dashed red) using Linear, Quadratic Spline, and Cubic Spline methods, along with the parametric Nelson-Siegel-Svensson (NSS) model.
While all methods offer viable approximations, the analysis reveals distinct performance differences:

- **Linear**: The method exhibits significant deviations in high-curvature segments, particularly in the short-term maturities (0–5 years) and medium-term hump (10–20 years). Its piecewise linear structure fails to capture the subtleties of the yield curve, resulting in abrupt changes in slope.
- **Quadratic Spline**: This method offers an improvement in smoothness and alignment compared to linear interpolation. However, minor deviations are evident in medium-term maturities, where the yield curve features subtle inflection points or fluctuations.
- **Cubic Spline**: The cubic spline method delivers the highest level of accuracy and smoothness. It closely mirrors the bootstrapped curve across all maturities, minimizing deviations even in areas of sharp curvature or volatility. This makes it a reliable tool to represent market expectations.
- **The Nelson-Siegel-Svensson (NSS)** method, closely aligns with the bootstrapped zero-coupon curve across all maturities, delivering a smooth and precise representation. It effectively captures the initial sharp decline in short-term rates (0–5 years), the medium-term hump (10–20 years), and the gradual flattening in the long term (20–30 years).

The cubic spline method emerges as the most effective between the interpolation methods, offering robust accuracy and continuity, which are essential for applications such as pricing, risk management, and portfolio optimization. Linear and quadratic methods, while acceptable for preliminary analyses, fall short in environments where precision and smoothness are paramount. However, the Nelson-Siegel-Svensson (NSS) method, as a parametric model, demonstrates a better performance, particularly for capturing long-term yield curve dynamics. Its primary strengths lie in its interpretability and flexibility, which are crucial for interest rate modeling, curve extrapolation, and macroeconomic analysis. The NSS method offers a robust framework for understanding the underlying economic factors that drive the yield curve, which is crucial for accurate forecasting and analysis of macroeconomic trends.

Thus, despite the cubic spline's excellent interpolation capabilities, NSS model stands out as the preferred choice for comprehensive financial analysis. Its ability to provide both precision and valuable economic insights makes it the most suitable tool for modeling yield curves across both short- and long-term horizons.

In [20]:
```
bootstrapper.plot_comparison()
```



Macroeconomic Insights from the Yield Curve: Current Trends and Implication

6.1. Yield Curve Shapes: Normal, Inverted, and Flat

The zero-coupon rate curve is an important thermometer of the perception of the health of the economy. A key metric in this analysis is the **term spread**, defined as the difference between long-term and short-term interest rates. This spread acts as a dynamic indicator, reflecting market expectations about future economic conditions.

The yield curve shape is not static: as the general level of interest rates rises or falls, the yield curve correspondingly shifts up or down, adopting various slopes that correspond to different economic outlooks. This dynamism can make us witnesses of different shapes of the zero-coupon rate curve, which are:

1. **Upward-Sloping Yield Curve**
2. **Flat Yield Curve**

3. **Inverted Yield Curve**

Upward-Sloping Yield Curve

This shape of the curve can be observed when the **term spread is positive**, which means that longer-term interest rates are higher than short-term rates.
It usually signals optimism about future economic growth. During periods of economic expansion, central banks often set low short-term interest rates to stimulate borrowing and spending, encouraging economic activity, business investment, and consumer lending. As the economy grows, inflation expectations tend to rise, leading investors to demand higher returns for longer commitments to compensate for future price increases and potential risks. This demand pushes long-term interest rates higher, resulting in an upward-sloping yield curve that reflects positive economic sentiment and expectations of sustained growth.

Flat Yield Curve

This shape of the curve can be observed when the **term spread is close to zero**, meaning that short-term and long-term interest rates are approximately equal.
This often signals economic uncertainty or a transition period, suggesting that investors have mixed expectations about future economic growth and inflation rates. In such scenarios, investors may prefer short-term bonds over long-term ones due to the minimal yield premium for taking on additional interest rate risk associated with longer maturities.

Inverted Yield Curve

This shape of the curve can be observed when the **term spread is negative**, this happens when short-term interest rates exceed long-term rates. This is a noteworthy and uncommon event. Under normal circumstances, the yield curve is not inverted, as debt with longer maturities typically pays higher interest rates than shorter-term ones.
From an economic perspective, an inverted yield curve suggests that the near term is perceived as riskier than the long term. The increased demand for long-term bonds raises their prices and lowers their yields. Historically, an inverted yield curve has been one of the most reliable leading indicators of an impending recession. In fact, it generally reflects widespread market concerns about economic health.

6.2. Analysis and Insights on the Current Yield Curve

The computed yield curve clearly illustrates the factors influencing its shape. The short-term segment is predominantly shaped by the European Central Bank's (ECB) monetary policy, while the long-term segment reflects investor expectations about the economy's future health.

- **Short-Term:** The curve shows a sharp decline, with yields starting above 3.2% and quickly dropping below 2.6% within the first few years. This steep decline likely reflects the ECB's recent 25-basis-point rate cut (effective October 23, 2024), introduced to counter slowing economic growth or inflationary pressures in the Eurozone. Central bank rate cuts typically lower short-term yields as these closely follow monetary policy. Furthermore, such cuts signal reduced borrowing costs and potential stimulus, contributing to lower yields across the curve.

- **Medium-Term:** The curve stabilizes, hovering around 2.4-2.5% with a slight hump or uplift between the 10-20 year maturities. This flattening in the medium-term segment may suggest uncertainty about the medium-term economic outlook. Investors are likely uncertain whether recent ECB rate cuts will be effective in spurring economic growth or inflation over the next decade. The slight hump may indicate that some investors expect a moderate recovery in the medium term, potentially due to a lagged response from monetary stimulus. However, the relatively flat curve suggests cautious sentiment rather than strong confidence.

- **Long-Term:** Beyond the 20-year maturity, the curve slopes downward, with yields dropping below 2.3% and continuing to decline as maturity increases. This long-term inversion signals pessimism about future economic growth and inflation. In times of uncertainty, investors view long-term bonds as safe havens, and increased demand for these securities compresses their yields. This is common during periods of uncertainty, when investors prioritize security over returns, resulting in compressed long-term yields.

The inversion aligns with various market indicators, such as the high demand for bonds issued by European countries. A particularly illustrative metric is the cover ratio, which measures the the volume of bids received relative to the volume of bonds ultimately allocated. Recent auction data from European central banks for medium- to long-term maturities exhibit elevated cover ratios, typically ranging from 1.5 to 2.0. This strong demand may reflect that investors are desire to secure bonds at current yields, anticipating potential declines in future rates.

This heightened demand for long-term bonds also reflects market expectations regarding future inflation. Bonds are particularly sensitive to inflation, therefore, if investors anticipate a decline in inflation, long-term bonds become more attractive. This is because, in low-inflation or deflationary scenarios, central banks can maintain accommodative monetary policies, including persistently low interest rates to stimulate the economy, without the constraint of countering inflationary pressures. This environment enhances the appeal of long-duration securities.

Furthermore, the downward-sloping yield curve may indicate a "flight to safety" driven by recent geopolitical tensions, such as ongoing conflicts or regional instability. Under such conditions, long-term bonds are perceived as safer investments. This shift in investor preference compresses yields on longer maturities, as investors prioritize the security of capital preservation over the potential for higher returns, further reinforcing the inversion of the yield curve.

References

1. Brigo, D., & Mercurio, F. (2001). Interest Rate Models: Theory and Practice. Springer Finance. Springer.
2. ECB Press Release on Monetary Policy Decisions (17 October 2024).
3. Galiani, S., Polimeni, F., & Proietti, M. (2003). Credit derivatives e cartolarizzazione. Metodologie e analisi dei rischi. Il Sole 24 Ore.
4. Hagan, P., & West, G. (2006). Interpolation Methods for Curve Construction. Applied Mathematical Finance, 13(2).
5. Röman, J. R. M. (2017). Analytical Finance: Volume II: The Mathematics of Interest Rate Derivatives, Markets, Risk and Valuation (2017 Edition).