Fixed Income Securities

Group Project – Option A

Authors:

Luís Ribeiro (nº 20231536)

Renato Morais (nº 20231135)

Fernando Tiago (nº 20231535)

Thiago Bellas (nº 20231131)

Contents

Introduction

This analysis delves into the comprehensive evaluation of an inflation-linked bond (ILB) and the development of a corresponding hedging strategy aimed at managing interest rate risk effectively. In examining the ILB, we focus on crucial aspects such as accrued interest calculation, simulation of inflation scenarios, and analysis of cash flows to understand variability influenced by inflation and interest rates. Additionally, we conduct bond price distribution analysis to assess interest rate risk using a specific methodology.

In parallel, we develop a hedging strategy involving sensitivity analysis of hedging instruments and portfolio metrics. Optimal hedge ratios are computed to mitigate risk effectively, and results include key metrics such as level, slope, and curvature durations, along with dollar durations for comprehensive risk assessment.

Furthermore, we evaluate the impact of changes in the yield curve, both with and without the hedging strategy. Visualizations and calculations demonstrate the effectiveness of the hedging strategy in mitigating losses associated with yield curve shifts. This analysis underscores the critical importance of robust risk management strategies in investment portfolios, particularly when dealing with complex financial instruments like ILBs. By integrating historical data, future projections, and risk assessments, investors can make informed decisions to optimize portfolio performance and minimize potential losses in dynamic market conditions.

Exercise 1 – Capital Indexed Inflation Linked Bond

In this first exercise, we are given a term sheet for a Capital Indexed Inflation Bond (ILB) with the following term sheet:

Notional amount	25000
Coupon Type	Fixed
Coupon rate	6.75%
Coupon frequency	Semi-annual
Currency	USD
Issue Date	31/7/2020
Maturity Date	21/7/2025
Trade Date	18/09/2020
Settlement Lag	T+1
Day Count	ACT/ACT
Inflation Reference Index	US Consumer Price Index
Inflation Reference Index Level at issue	237.14365
Inflation Reference Index Level at Settlement	251.14721

Since this is a Capital Index ILB, we are dealing with two types of inflation adjustment: notional adjusted to the inflation index ratio (Capital Index); and the cash-flows adjusted to the inflation index ratio (ILB).

In this case, we are given two CPI indices for settlement and issue dates. These will be useful for forecasting and index ratio adjustment, respectively.

For the forecasting, we are going to use a Geometric Brownian Motion (GBM) with a diffusion coefficient $\hat{\mu} = 0.05321$ and $\hat{\sigma} = 0.06358$.

For the yield curve, we are given a Nelson-Siegel-Svensson set of parameters:

β_0	β_1	β_2	β_3	τ_1	τ_2
5.9%	-1.6%	-0.5%	1%	5	0.5

Task A – Compute accrued interest

The accrued interest of a bond is the interest accumulated from the last coupon up until the settlement date. Before calculating it, we need to account for inflation using the Index Ratio, which in this case will be the settlement CPI divided by the issue date CPI, giving us 1.0590, indicating an increase of approximately 5.9% in the Consumer Price Index (CPI) between the issuance and settlement of the bond.

```
settlement_cpi = 251.14721
issue_cpi = 237.14365
index_ratio = settlement_cpi/issue_cpi
print("Index ratio: ", index_ratio)
```

```
Index ratio:  1.0590509591970942
```

The accrued interest value before the first coupon payment date was determined by considering the time elapsed from the issuance date to the settlement date. However, due to the semiannual coupon payment structure with payment dates on 21/1 and 21/7, the first coupon is prorated based on the time elapsed from the issuance date to the settlement date.

In this case, there was no previous coupon up until the settlement date, so we will compute the accrued interest from the issue date.

For better calculations, we have resorted to using QuantLib and defining the following functions which will also be used later:

```
import QuantLib as ql

def count_days(d1: ql.Date, d2: ql.Date):
    return ql.ActualActual(ql.ActualActual.ISDA).dayCount(d1, d2)-1

def count_years(d1: ql.Date, d2: ql.Date):
    return ql.ActualActual(ql.ActualActual.ISDA).yearFraction(d1, d2)

def settlement_lag_date(date: ql.Date, lag: int):
    return ql.TARGET().advance(date, lag, ql.Days)

def get_coupon_dates(issue_date:ql.Date, maturity_date: ql.Date, frequency: int):
    schedule = ql.Schedule(
        issue_date,
        maturity_date,
        ql.Period(frequency),
        ql.TARGET(),
        ql.Following,
        ql.Following,
        ql.DateGeneration.Backward,
        False
    )
    return [dt for dt in schedule]

def ql_date_to_pandas_timestamp(date: ql.Date):
    return pd.Timestamp(date.year(), date.month(), date.dayOfMonth())

def get_monthly_dates(d1:ql.Date, d2:ql.Date):
    return [
        d1 + ql.Period(i, ql.Months)
        for i in range((d2.year() - d1.year()) * 12 + d2.month() - d1.month() + 1 + 1)
    ]
```

Finally, we reach the result seen below:

```
u = count_days(issue_date, settlement_date)
w = count_days(coupon_dates[-2], coupon_dates[-1])

print("Days between first coupon/issue date and settlement:", u)
print("Days between regular coupon payments:", w)

accrued_interest = face_value * coupon_rate/coupon_frequency * u/w
real_accrued_interest = index_ratio * accrued_interest

print("Accrued Interest ($): ", accrued_interest)
print("Real Accrued Interest ($): ", real_accrued_interest)
```

✓ 0.0s

```
Days between first coupon/issue date and settlement: 51
Days between regular coupon payments: 180
Accrued Interest ($):  239.0625
Real Accrued Interest ($):  253.17936993305534
```

Note: the real accrued interest is simply the accrued interest adjusted for inflation.

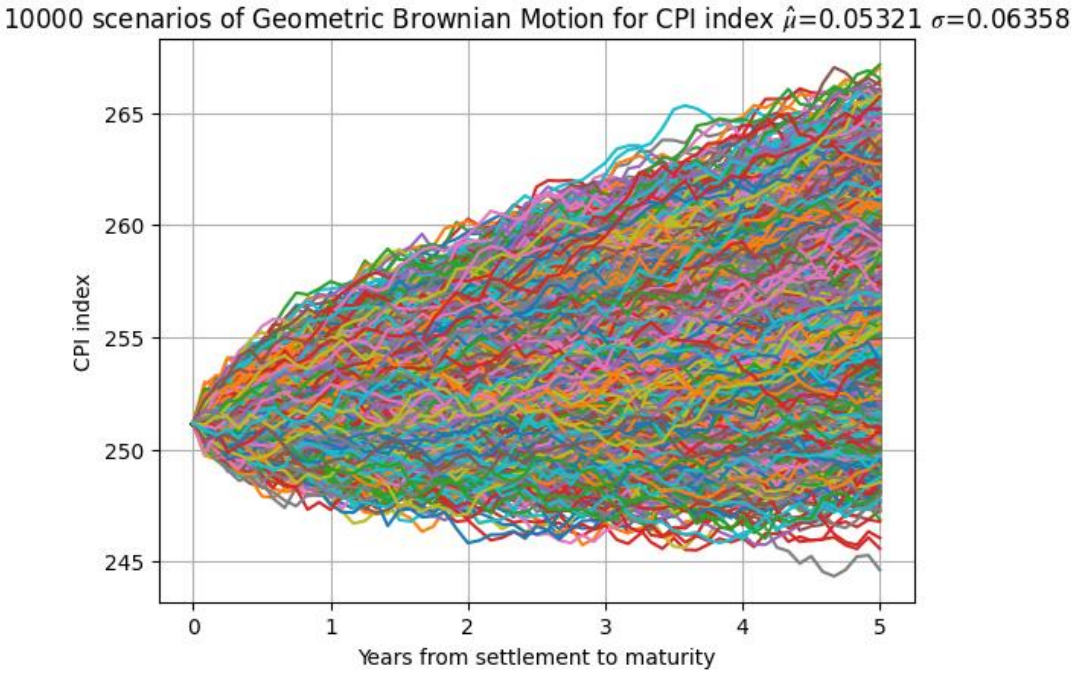
Task B – Simulate 10000 scenarios of the CPI index

Even though not the most correct technique, we have simulated the CPI index, using the settlement date CPI index and then forth with the given coefficients, for each month since the US CPI is updated monthly. The indices are then "interpolated" later.

Here is a snippet of code for simulating the 10000 scenarios of the CPI Index:

```
import numpy as np

years = round(count_years(settlement_date, maturity_date))
steps = 12 * years
dt = years / steps

constant_drift = 0.05321
diffusion_coefficient = 0.06358
n_scenarios = 10000

z = np.random.normal(0, np.sqrt(dt), size=(n_scenarios, steps))
cpi_scenarios = np.exp(
    (constant_drift*dt - 0.5 * diffusion_coefficient* dt **2) * dt
    + diffusion_coefficient * dt * z
)
cpi_scenarios = np.vstack([np.ones(n_scenarios), cpi_scenarios.T])
cpi_scenarios = settlement_cpi * cpi_scenarios.cumprod(axis=0)

t = np.linspace(0, years, steps + 1)
tt = np.full(shape=(n_scenarios, steps + 1), fill_value=t).T
```

Plotting them, we obtain the following figures for the various scenarios:

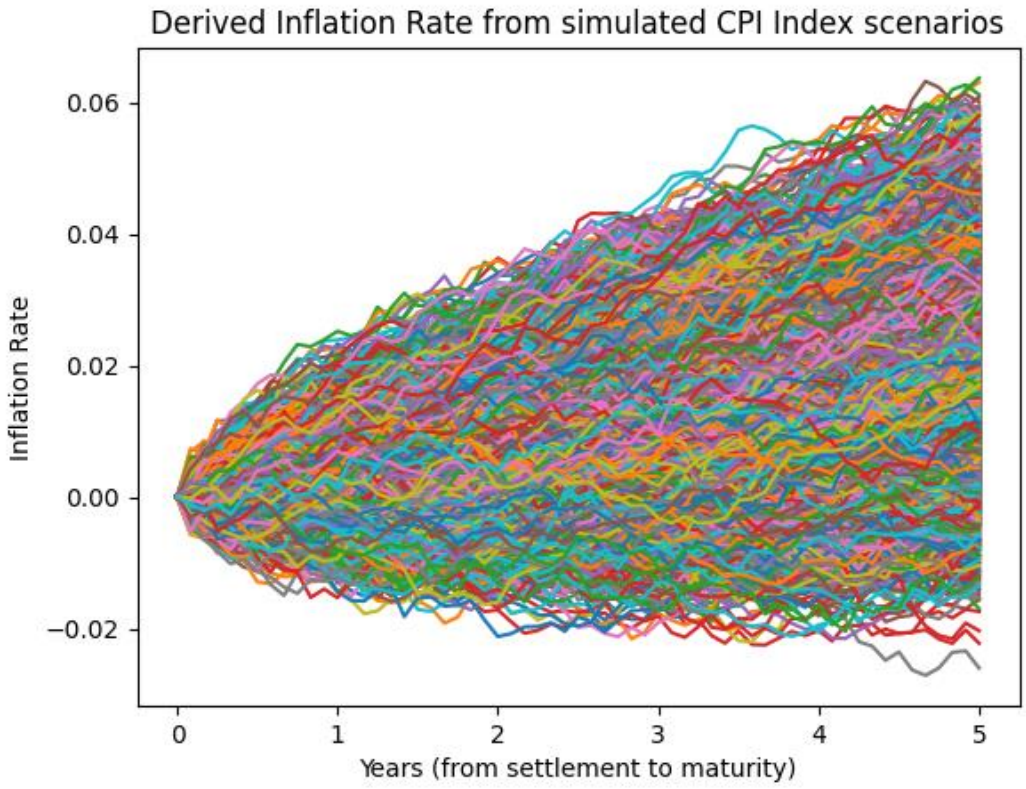


For each scenario, we can then compute the inflation rate which is given by:

$$\pi_t = \frac{CPI_t^R}{CPI_0^R} - 1$$

```
cpi_scenarios = cpi_scenarios.T #since each column is scenario

plt.figure();
plt.title("Derived Inflation Rate from simulated CPI Index scenarios");

inflation_scenarios = np.array([pd.Series(cpi_scenario)/cpi_scenario[0] - 1 for cpi_scenario in cpi_scenarios])

plt.plot(t, inflation_scenarios.T);
plt.xlabel("Years (from settlement to maturity)");
plt.ylabel("Inflation Rate");
```

Plotting them, we can see a clear uptrend as inflation tends to go up over time:



Task C – ILB cash-flows/fair value for each scenario

As previously stated, the first coupon will be short. So, we can start by determining how much of the coupon we will receive. We achieved it with the snippet below:

```
short_discount = count_days(issue_date, coupon_dates[0])/w
print(f"First coupon is short, so we'll only receive {round(short_discount*100, 2)}% of it.")
```

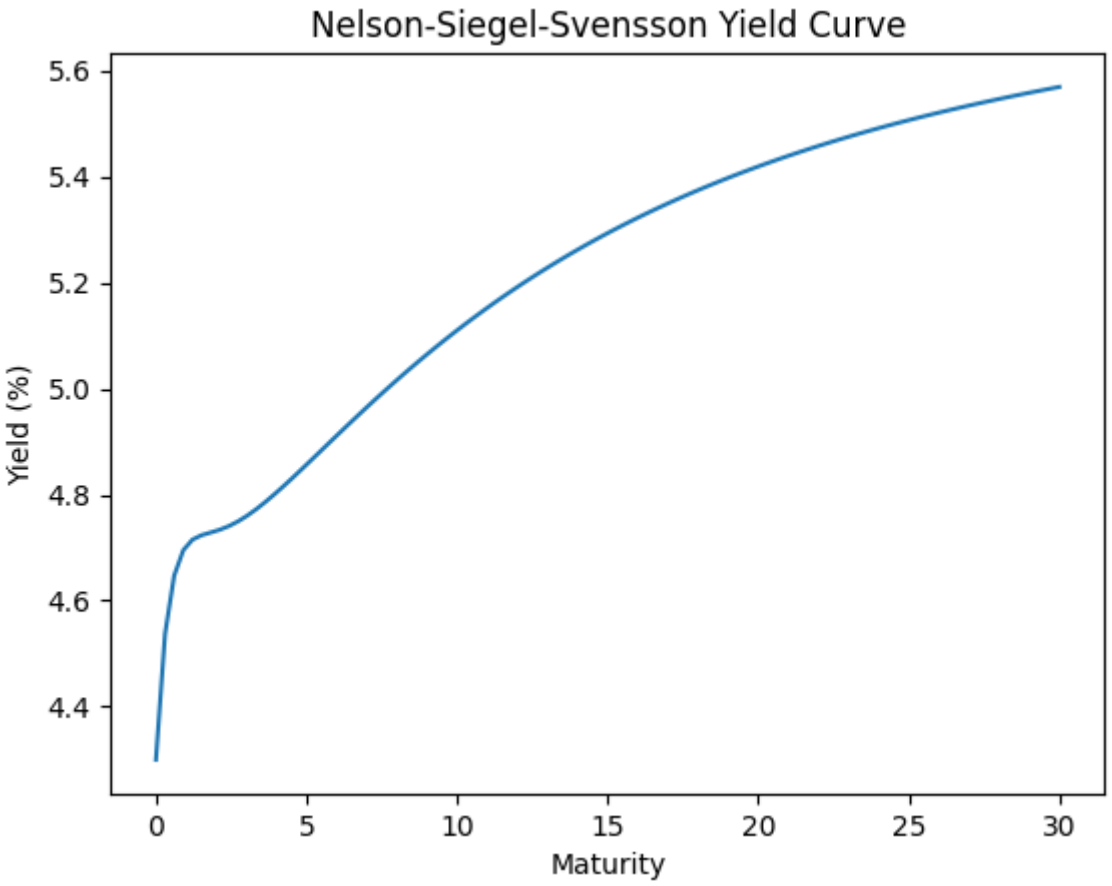
✓ 0.0s

```
First coupon is short, so we'll only receive 96.11% of it.
```

The next step will be computing the yield curve which will determine the various interest rates for each coupon. Since we are given the NSS parameters, we can then use the following snippet of code to render the curve:

```
from nelson_siegel_svensson import NelsonSiegelSvenssonCurve
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

yield_curve = NelsonSiegelSvenssonCurve(
    beta0=5.9/100,
    beta1=-1.6/100,
    beta2=-0.5/100,
    beta3=1/100,
    tau1=5,
    tau2=0.5
)
t = np.linspace(0, 30, 100)

plt.figure();
plt.plot(t, yield_curve(t)*100);
plt.title('Nelson-Siegel-Svensson Yield Curve');
plt.xlabel('Maturity');
plt.ylabel('Yield (%)');
```

Plotting the curve over the various maturities we get the following yield curve:



The Nelson-Siegel-Svensson (NSS) yield curve with the provided parameters can be analyzed as follows:

- **Long-term Level (β0):** The long-term level of the yield curve is determined by β0, which in this case is 5.9%. This parameter represents the convergence point of the yield curve as maturity extends to infinity.
- **Short-term Factor (β1):** The short-term factor, represented by β1, influences the initial slope of the yield curve. Here, β 1=−1.6% suggests an initial downward slope at short maturities.
- **Medium-term Factor (β2):** The medium-term factor, β2, further shapes the yield curve. With β2=−0.5%, the curve exhibits a slight upward bend in the medium term.
- **Long-term Factor (β3):** The long-term factor, β3, has a significant impact on the curve's behavior at longer maturities. In this case, β3=1% indicates an upward slope in the long term.
- **Decay Parameters (τ1 and τ2):** The decay parameters, τ1 and τ2, determine the speed at which the factors β1, β2, and β3 converge towards their respective levels. A larger τ value implies a slower convergence. Here,

τ1=5 years and τ2=0.5 years, suggesting that β1 and β2 converge slowly, while β3 converges more rapidly.

Overall, with these parameters, the yield curve exhibits an initial upward slope at short maturities meaning that short-term investments will yield higher returns, followed by a slight downward bend in the medium term meaning these types of bonds aren't as desirable as the ones described before due to a decrease in the interest rate growth, and eventually, an upward slope in the long term that will make these bonds more attractive than the medium term bonds due to having higher interest rates, product of this upward slope. The convergence of the factors is influenced by the decay parameters, with β1 and β2 converging more slowly compared to β3.

To compute the various cash flows for each scenario, we have used the following code (please note that it is incomplete, but the snippet is the core of the code):

```
for cpi_scenario in cpi_scenarios:

    #US CPI Index starting at settlement in DataFrame (monthly)
    cpi_df = pd.DataFrame({'cpi': cpi_scenario[:-1]}, index=cpi_dates_pandas_timestamps)

    inflation_scenario_rates = []
    cpi_scenario_spot_rates = []

    cashflows = np.zeros(number_of_ilb_cashflows)

    for i in range(0, number_of_ilb_cashflows):

        #start with cashflow 0
        coupon_date = coupon_dates[i]
        cpi = np.float64(cpi_df[  (cpi_df.index.month == coupon_date.month()) & (cpi_df.index.year == coupon_date.year()) ].cpi)

        #Calculate inflatio growth (index ratio) & current spot rate (via Fisher)
        inflation_index = cpi/issue_cpi
        inflation_rate = inflation_index - 1
        inflation_scenario_rates.append(inflation_rate)
        inflation_adjusted_coupon = coupon_rate * inflation_index

        #calculate nominal yield rate before inflation
        tenor = (i+1)/coupon_frequency
        market_yield = yield_curve(tenor)

        #Fisher affected spot rate (inflation)
        #spot_rate = (1+market_yield)/(1+inflation_rate) - 1
        spot_rate = market_yield #assume spot rate is nominal, not real spot rate
        cpi_scenario_spot_rates.append(spot_rate)

        #Cash flows computations
        discount_factor = 1 / ( (1+spot_rate/coupon_frequency)**(i+1) ) #fix discount factor
        pv_cashflow = face_value * inflation_adjusted_coupon/coupon_frequency * discount_factor

        if i == 0: # is short first coupon
            cashflows[i] = pv_cashflow * short_discount
        elif i == number_of_ilb_cashflows-1: # if last coupon
            redemption_amount = face_value * inflation_index
            cashflows[i] = pv_cashflow + (redemption_amount * discount_factor)
        else: # normal case
            cashflows[i] = pv_cashflow
```

Afterward, we can achieve a bird-eye view of the various cash flows for each scenario:

```
cashflows_df = pd.DataFrame(data=scenarios_cashflows, columns=coupon_dates)
cashflows_df
```

✓ 0.0s

	January 21st, 2021	July 21st, 2021	January 21st, 2022	July 21st, 2022	January 23rd, 2023	July 21st, 2023	January 22nd, 2024	July 22nd, 2024	January 21st, 2025	July 21st, 2025
0	837.752801	854.658166	840.666655	820.175501	802.529039	780.801980	759.904651	745.634293	730.067096	21616.945029
1	835.829398	848.190680	831.903350	810.786017	798.301584	775.646971	754.557931	742.023299	727.908680	21797.449232
2	840.397395	852.775060	833.206095	813.719586	795.387323	780.870421	760.111177	741.684574	727.773456	21635.028199
3	843.355086	857.919409	845.857671	824.898387	806.028551	787.326099	770.505406	750.276449	735.880416	21839.117588
4	839.322215	851.860899	831.748851	815.425450	801.649742	783.946337	764.623799	748.048070	731.250435	21962.323670
...	...	...	...	...	...	...	...	...	...	...
9995	836.762118	845.937118	829.411780	807.567006	790.043771	773.346366	758.082254	735.099446	718.702793	21468.774215
9996	839.009766	856.519429	837.522596	816.303361	798.532752	782.397942	767.662317	753.605460	739.385351	22150.427822
9997	840.618855	854.245754	839.338649	824.999704	809.450085	789.581828	772.949793	754.344177	733.838720	22048.547111
9998	838.630341	854.383166	832.958799	818.759452	797.660375	774.934222	754.469858	743.825073	727.198730	21832.363546
9999	837.196626	853.541398	838.615490	817.573139	800.403115	787.396291	767.530990	746.893441	734.491968	22024.817765

10000 rows × 10 columns

The columns up to the penultimate one represents regular coupon payments, while the last column represents the maturity redemption value. The rows indicate the amounts of cash flow for each simulated scenario. Variations between scenarios can be observed, influenced by the evolution of inflation rates and interest rates. The value in the last column (July 21st, 2025) represents the maturity redemption payment.

We can also get a statistical description per cash flow:

```
cashflows_df.describe()
```

✓ 0.0s

	January 21st, 2021	July 21st, 2021	January 21st, 2022	July 21st, 2022	January 23rd, 2023	July 21st, 2023	January 22nd, 2024	July 22nd, 2024	January 21st, 2025	July 21st, 2025
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	840.646757	855.976120	837.852693	820.117668	802.647333	785.304813	768.087407	751.042168	734.192539	21975.100039
std	2.580925	4.150390	5.159276	5.910428	6.524585	7.026674	7.467135	7.783661	8.168617	258.427869
min	830.926792	839.957277	819.766314	799.834773	779.444940	759.249639	741.437676	723.383823	704.546356	21023.038080
25%	838.932257	853.231139	834.350593	816.183600	798.299228	780.601630	763.075628	745.778961	728.722215	21799.364152
50%	840.607607	855.973126	837.880993	820.161785	802.665285	785.200961	768.035754	751.043762	734.154158	21972.547369
75%	842.365042	858.738892	841.384418	824.107935	807.046387	789.934066	773.106257	756.266290	739.532547	22146.086693
max	850.232153	872.503155	857.095533	841.993380	825.607624	809.515935	795.099289	779.041061	763.673900	22865.976469

The mean provides a central estimate for the interest receipt for each coupon payment date. In the context of cash flows, it represents the expected average value on each payment date, considering the 10,000 projected inflation scenarios.

The standard deviation measures the dispersion of values around the mean. The higher the standard deviation, the greater the variability. For instance, the last column, July 21st, 2025, has the highest standard deviation, indicating higher variability in the maturity redemption values.

Note: these values may not be properly seen here, so we recommend opening the attached Jupyter Notebook for better viewing.

Finally, we can compute the fair values by simply creating a column that is the sum of the discounted cash flows.

```
cashflows_df['B0'] = cashflows_df.values.sum(axis=1)
cashflows_df
```

✓ 0.0s

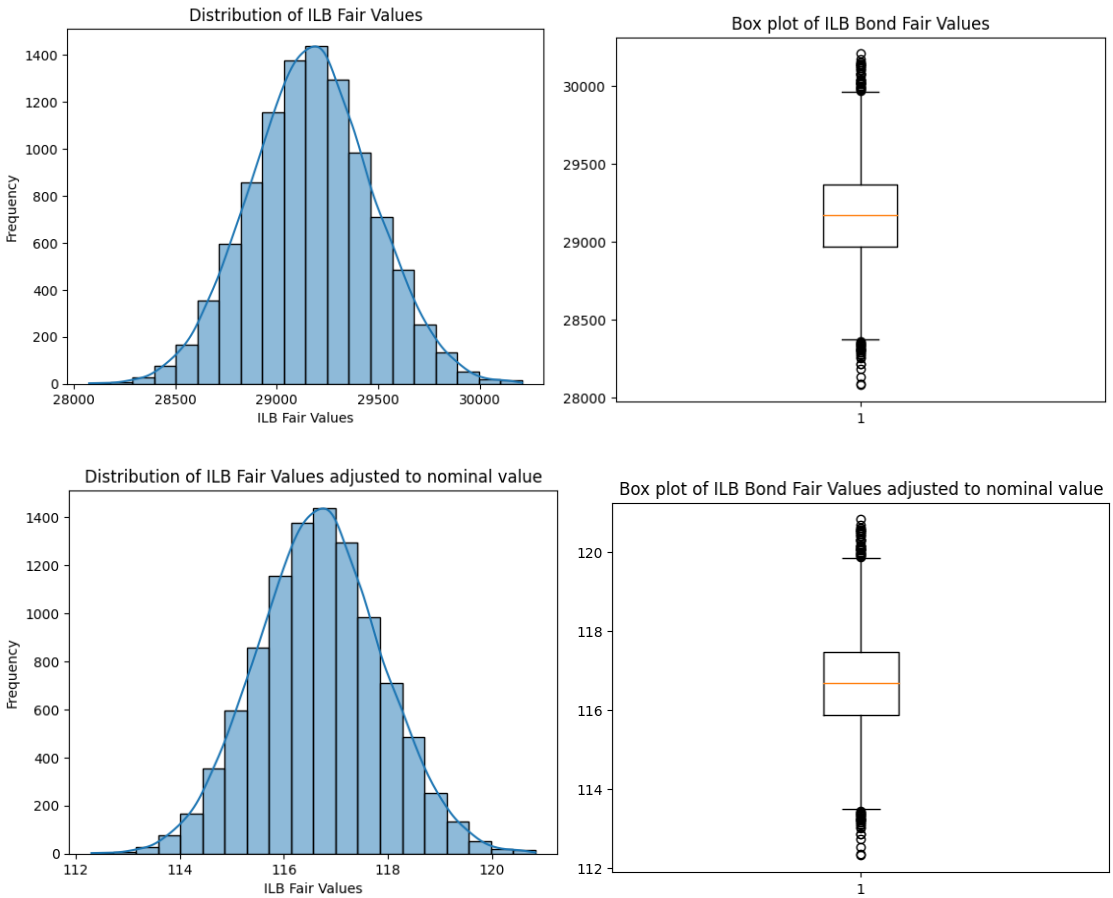
In the table below, we can see an overlook of all the fair values for all scenarios:

B0
28789.135212
28922.597141
28780.953287
29061.165063
29130.199467
...
28563.726867
29341.366796
29267.914675
28975.183562
29208.460222

Task D – Estimate and analyze the ILB price distribution, including interest rate and inflation risk measures

Price distribution

To analyze the fair values, we can simply plot the fair value column in the cash-flows dataframe. To better analyze it, we have decided to plot both a box plot and a price distribution (dollar amounts and adjusted to face value, respectively):



Looking at both plots, we can see that while there are some outlier scenarios, all scenarios resulted in the values above par. This is unsurprising since we're dealing with a Capital Indexed (so the reimbursement value is inflation-adjusted) ILB (all coupons are inflation-adjusted relative to the index) and, in general, the CPI Index has risen significantly since issuance.

Interest Rate Risk measures

The assessment of interest rate risk measures aims to provide a comprehensive understanding of the sensitivity of the inflation-linked bond to fluctuations in interest rates. For this analysis, we employ the Nelson-Siegel-Svensson methodology to model the term structure of interest rates since the yield curve serves as a crucial reference point for calculating the Fisher-Weil Durations of the bond at different points in time.

```
ilb_durations_beta = np.zeros((n_scenarios, 4))

for i, row in cashflows_df.iterrows():
    cfs = row[coupon_dates].values
    b0 = row['B0']
    #spot_rates = scenarios_spot_rates[i]

    dt = 1/coupon_frequency
    t = np.arange(1, number_of_ilb_cashflows + 1) * dt
    exp_tau1 = np.exp(-t / yield_curve.tau1)
    exp_tau2 = np.exp(-t / yield_curve.tau2)

    ilb_durations_beta[i, 0] = np.sum(cfs * t) / b0
    ilb_durations_beta[i, 1] = np.sum(cfs * t * (1 - exp_tau1) * yield_curve.tau1) / b0
    ilb_durations_beta[i, 2] = np.sum(cfs * t * (1 - exp_tau1 * (1 + t / yield_curve.tau1)) * yield_curve.tau1) / b0
    ilb_durations_beta[i, 3] = np.sum(cfs * t * (1 - exp_tau2 * (1 + t / yield_curve.tau2)) * yield_curve.tau2) / b0

ilb_durations_df = pd.DataFrame(ilb_durations_beta, columns=['D0', 'D1', 'D2', 'D3'])
cashflows_df = pd.concat([cashflows_df, ilb_durations_df], axis=1)
cashflows_df[['B0', 'D0', 'D1', 'D2', 'D3']]
```

✓ 5.4s

```
cashflows_df[['B0', 'D0', 'D1', 'D2', 'D3']].describe()
```

✓ 0.0s

	B0	D0	D1	D2	D3
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	29170.967538	4.367581	13.255662	5.383902	2.162073
std	299.274162	0.004328	0.016846	0.007789	0.002325
min	28076.357088	4.351367	13.192752	5.355098	2.153348
25%	28968.951728	4.364630	13.244313	5.378648	2.160494
50%	29170.440170	4.367611	13.255797	5.383955	2.162089
75%	29369.864302	4.370427	13.266802	5.389075	2.163609
max	30207.348014	4.382341	13.312853	5.410623	2.170021

The interpretation of Fisher-Weil Durations, represented by the variables D0, D1, D2, and D3, can be conducted by considering the percentage sensitivity of a bond's price to changes in interest rates.

The conventional duration (D0) is the percentage sensitivity of the bond's price to parallel Shift changes in interest rates. If interest rates increase or decrease by 1%, the bond's price is expected to move approximately 4% in the opposite direction. It is a basic measure of interest rate risk.

The first modified duration (D1) reflects the percentage sensitivity of the bond's price to changes in the short-term yield curve slope changes. A change in the short-term yield curve slope will result in an approximate 13% change in the bond's price.

In this section, we move on to the dollar durations associated with each beta parameter, providing a monetary perspective on the bond's sensitivity to changes in interest rates. Dollar durations, denoted as $D0, $D1, $D2, and $D3, quantify the financial impact of percentage variations in interest rates on the present values of the bond.

```
cashflows_df['$D0'] = -cashflows_df['B0'] * cashflows_df['D0']
cashflows_df['$D1'] = -cashflows_df['B0'] * cashflows_df['D1']
cashflows_df['$D2'] = -cashflows_df['B0'] * cashflows_df['D2']
cashflows_df['$D3'] = -cashflows_df['B0'] * cashflows_df['D3']
cashflows_df[['B0', '$D0', '$D1', '$D2', '$D3']]
```

✓ 0.0s

```
cashflows_df[['B0', '$D0', '$D1', '$D2', '$D3']].describe()
```

✓ 0.0s

	B0	$D0	$D1	$D2	$D3
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	29170.967538	-127407.687866	-386684.853318	-157055.628475	-63070.386625
std	299.274162	1419.230790	4399.245710	1809.036804	707.477525
min	28076.357088	-132330.987681	-401939.703203	-163324.335282	-65524.834609
25%	28968.951728	-128343.552526	-389595.019990	-158253.029426	-63537.557996
50%	29170.440170	-127404.027725	-386670.044602	-157044.830005	-63067.595788
75%	29369.864302	-126452.476164	-383712.246873	-155830.305783	-62593.614776
max	30207.348014	-122212.887799	-370542.948932	-150407.496545	-60479.258016

The average dollar duration for the beta zero coefficient of the NSS curve is approximately -$127407, suggesting that, on average, the bond's value decreases by $164,618 for a one percentage point parallel Shift variation in interest rates. A similar analysis can be done for the D1 dollar duration (slope Shift), but not the other parameters.

Convexity is a crucial metric in bond evaluation, providing additional insights into the risk associated with changes in interest rates. In this context, we calculated the bond's convexity concerning changes in interest rates using the Nelson-Siegel-Svensson curve.

```
ilb_convexities_beta = np.zeros((n_scenarios, 2))

for i, row in cashflows_df.iterrows():
    cfs = row[coupon_dates].values
    b0 = row['B0']

    dt = 1/coupon_frequency
    t = np.arange(1, number_of_ilb_cashflows + 1) * dt
    exp_tau1 = np.exp(-t / yield_curve.tau1)
    tau1_pwr = np.power(yield_curve.tau1, 2)

    ilb_convexities_beta[i, 0] = np.sum(cfs * np.power(t, 2)) / b0
    ilb_convexities_beta[i, 1] = np.sum(cfs * np.power(t, 2) * np.power((1-exp_tau1), 2) * tau1_pwr) / b0

cashflows_df = cashflows_df.assign(C0=ilb_convexities_beta[:,0], C1=ilb_convexities_beta[:,1])
cashflows_df[['B0', 'C0', 'C1']]
```

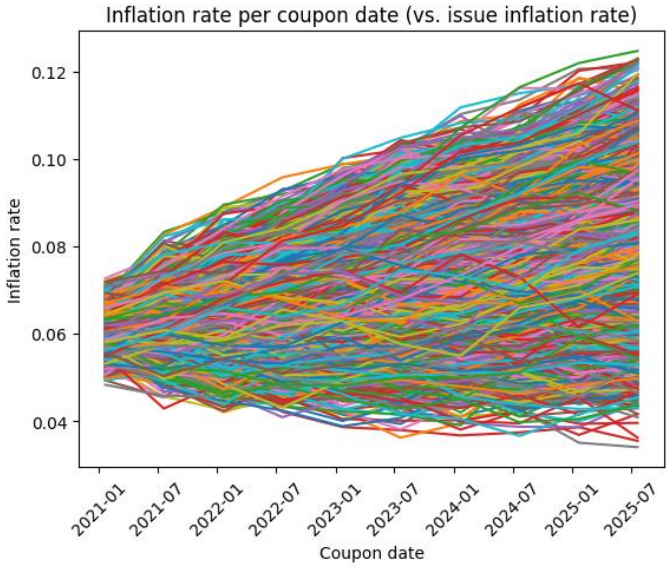
✓ 5.2s

```
cashflows_df[['B0', 'C0', 'C1']].describe()
```

✓ 0.0s

	B0	C0	C1
count	10000.000000	10000.000000	10000.000000
mean	29170.967538	20.704817	200.120158
std	299.274162	0.027861	0.308190
min	28076.357088	20.601307	198.991419
25%	28968.951728	20.686031	199.914240
50%	29170.440170	20.704974	200.121881
75%	29369.864302	20.723206	200.324884
max	30207.348014	20.799916	201.184146

In the process of evaluating an inflation-linked bond, it is crucial to understand the dynamics of inflation rates over time, as well as their variations across different simulation scenarios. The graph below presents the inflation rates at each coupon date relative to the initial inflation rate of the bond. Each line in the graph represents a simulation scenario.



Exercise 2 – Hedging Portfolio

We start by defining the function "*NSS_Sens*" which calculates various sensitivities and durations related to a Nelson-Siegel-Svensson (NSS) interest rate curve. The function takes parameters such as betas coefficients, tau coefficients, par value, coupon rate, and maturity. It then computes dollar and parametric durations for each beta coefficient. The main function call at the end calculates and returns the durations for a given coupon rate and maturity.

```
def NSS_Sens(beta0=0.08, beta1=-0.03, beta2=-0.01, beta3=0, tau1=3, tau2=1, par=100, coupon_rate=None, maturity=None):

    ct = par * np.repeat(coupon_rate, maturity)
    ct[-1] += par

    theta_i = np.arange(1, maturity + 1)
    r0t = np.zeros(maturity)

    nss_curve = NelsonSiegelSvenssonCurve(
        beta0=beta0,
        beta1=beta1,
        beta2=beta2,
        beta3=beta3,
        tau1=tau1,
        tau2=tau2
    )

    for i in range(maturity):
        r0t[i] = nss_curve(theta_i[i])

    # dollar-durations
    #$D0 -> $D0 * Delta B0 (usually 1% - percentage of how much changed) = dollar cost per percentage shift in B0
    beta0_sens = -np.sum(ct * theta_i * np.exp(-theta_i * r0t))
    beta1_sens = -np.sum(ct * theta_i * (1 - np.exp(-theta_i / tau1)) / (theta_i / tau1) * np.exp(-theta_i * r0t))
    beta2_sens = -np.sum(ct * theta_i * ((1 - np.exp(-theta_i / tau1)) / (theta_i / tau1) - np.exp(-theta_i / tau1)) * np.exp(-theta_i * r0t))
    beta3_sens = -np.sum(ct * theta_i * ((1 - np.exp(-theta_i / tau2)) / (theta_i / tau2) - np.exp(-theta_i / tau2)) * np.exp(-theta_i * r0t))

    # parametric durations
    B0 = np.sum(ct * np.exp(-theta_i * r0t))

    D0 = -(1 / B0) * beta0_sens #fisher-weil duration -> time interpretation (para beta 0)
    D1 = -(1 / B0) * beta1_sens #do not have interpretation in time; percentage variation
    D2 = -(1 / B0) * beta2_sens #do not have interpretation in time
    D3 = -(1 / B0) * beta3_sens #do not have interpretation in time

    return {
        'B0': B0,
        '$D': {'Beta0': beta0_sens, 'Beta1': beta1_sens, 'Beta2': beta2_sens, 'Beta3': beta3_sens},
        'D': {'Beta0': D0, 'Beta1': D1, 'Beta2': D2, 'Beta3': D3}
    }
```

Note: this function is a Python adaptation of the code generously given, and demonstrated, in class.

Task A – Compute the level, slope, and curvature durations and dollar durations of the target portfolio

For this question, we could divide the actions we took by six points. We have started by defining a function, "*get_number_of_coupons_since_valuation*," which helps us determine the number of coupons from the valuation date until maturity.

```
def get_number_of_coupons_since_valuation(maturity, frequency=1):
    return len(get_coupon_dates(valuation_date, maturity, frequency)[1:])
```

Our portfolio is represented in a Pandas' Dataframe ("*portfolio_instruments*"), encapsulating details like coupon rates, maturity dates, quantities, and the derived coupon counts.

Venturing into the portfolio, we iterate through its instruments, employing the NSS sensitivity function ("*NSS_Sens*") to unveil sensitivities and durations for each.

```
portfolio_instrument_values = []

for i in range(0, portfolio_instruments.shape[0]):
    portfolio_instrument = portfolio_instruments.iloc[i]
    portfolio_instrument_values.append(NSS_Sens(
        **yield_curve_params,
        coupon_rate=portfolio_instrument.coupon/100,
        maturity=portfolio_instrument.coupons
    ))
```

For a comprehensive risk assessment, we delve into computing various metrics – from bond prices (B0) to dollar durations ($D0, $D1, $D2, $D3) and parametric durations (D0, D1).

```
# Compute risk measures
portfolio_instruments['B0'] = [value['B0'] for value in portfolio_instrument_values]
portfolio_instruments['$D0'] = [value['$D']['Beta0'] for value in portfolio_instrument_values]
portfolio_instruments['$D1'] = [value['$D']['Beta1'] for value in portfolio_instrument_values]
portfolio_instruments['$D2'] = [value['$D']['Beta2'] for value in portfolio_instrument_values]
portfolio_instruments['$D3'] = [value['$D']['Beta3'] for value in portfolio_instrument_values]
portfolio_instruments['D0'] = [value['D']['Beta0'] for value in portfolio_instrument_values]
portfolio_instruments['D1'] = [value['D']['Beta1'] for value in portfolio_instrument_values]
```

To gauge the overall portfolio's worth, we calculate its present value by summing the product of bond prices and quantities.

```
portfolio_pv = (portfolio_instruments['B0'] * portfolio_instruments['quantity']).sum()
portfolio_pv
```

```
88705967.2539232
```

Transitioning to duration, we compute portfolio-level measures ("*portfolio_D0*", "*portfolio_D1*") and the spectrum of dollar durations ("*portfolio_Dollar_D0*" to "*portfolio_Dollar_D3*").

```
portfolio_instruments['weight'] = portfolio_instruments['quantity'] * portfolio_instruments['B0'] / portfolio_pv
portfolio_instruments
```

✓ 0.0s

```
portfolio_D0 = (portfolio_instruments['D0'] * portfolio_instruments['weight']).sum()
portfolio_D1 = (portfolio_instruments['D1'] * portfolio_instruments['weight']).sum()
portfolio_Dollar_D0 = (portfolio_instruments['$D0'] * portfolio_instruments['weight']).sum()
portfolio_Dollar_D1 = (portfolio_instruments['$D1'] * portfolio_instruments['weight']).sum()
portfolio_Dollar_D2 = (portfolio_instruments['$D2'] * portfolio_instruments['weight']).sum()
portfolio_Dollar_D3 = (portfolio_instruments['$D3'] * portfolio_instruments['weight']).sum()
```

After all these steps, we have achieved the following results for the portfolio:

```
Portfolio D0 7.585479686412395
Portfolio D1 3.3999509599563593
Portfolio $D0: -748.9729344912599
Portfolio $D1: -346.1140496496504
Portfolio $D2: -191.15643654596244
Portfolio $D3: -49.95181300234362
```

Task B – Compute level, slope, and curvature durations and dollar durations

We start by constructing a Dataframe ("*hedging_instruments*") containing essential details about hedging instruments, specifying coupon rates, and maturity dates. To provide a more nuanced perspective, we calculate the number of coupons since the valuation date for each hedging instrument.

```
hedging_instruments["coupons"] = hedging_instruments["maturity"] \
    .apply(lambda m: get_number_of_coupons_since_valuation(m))
```

Obtaining the following table:

	coupon	maturity	coupons
1	4.5	April 12th, 2026	5
2	5.0	December 28th, 2032	11
3	6.0	May 6th, 2035	14
4	6.0	October 10th, 2040	19
5	6.5	October 10th, 2051	30

After, we iteratively apply the NSS sensitivity function ("*NSS_Sens*") to unravel the intricacies of sensitivities and durations for each hedging instrument.

Then we shift our focus to risk quantification. We calculate various risk measures for the hedging instruments, encompassing bond prices (B0), dollar durations ($D0, $D1, $D2, $D3), and parametric durations (D0, D1).

```
hedging_instruments['B0'] = [value['B0'] for value in hedging_values]
hedging_instruments['$D0'] = [value['$D']['Beta0'] for value in hedging_values]
hedging_instruments['$D1'] = [value['$D']['Beta1'] for value in hedging_values]
hedging_instruments['$D2'] = [value['$D']['Beta2'] for value in hedging_values]
hedging_instruments['$D3'] = [value['$D']['Beta3'] for value in hedging_values]
hedging_instruments['D0'] = [value['D']['Beta0'] for value in hedging_values]
hedging_instruments['D1'] = [value['D']['Beta1'] for value in hedging_values]
```

Obtaining the following table that encapsulates the answers to the question proposed:

	coupon	maturity	coupons	B0	$D0	$D1	$D2	$D3	D0	D1
1	4.5	April 12th, 2026	5	98.519792	-369.110696	-256.866378	-86.454640	-48.036885	3.746564	2.607257
2	5.0	December 28th, 2032	11	98.049020	-850.678215	-376.096588	-244.706135	-47.802864	8.676050	3.835802
3	6.0	May 6th, 2035	14	106.512956	-1008.870331	-411.896844	-281.838808	-51.790502	9.471809	3.867106
4	6.0	October 10th, 2040	19	106.689206	-1276.296357	-424.561965	-318.647488	-51.878627	11.962751	3.979428
5	6.5	October 10th, 2051	30	113.899468	-1657.114117	-454.819602	-351.412409	-55.361594	14.548919	3.993167

Task C – Estimate the holdings of the hedging portfolio assuming the hedger wants to implement a self-financing hedging strategy

To hedge our portfolio, we used the following matrix equation:

$$\begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{pmatrix} = \begin{pmatrix} \$D_0^{H1} & \$D_0^{H2} & \$D_0^{H3} & \$D_0^{H4} & \$D_0^{H5} \\ \$D_1^{H1} & \$D_1^{H2} & \$D_1^{H3} & \$D_1^{H4} & \$D_1^{H5} \\ \$D_2^{H1} & \$D_2^{H2} & \$D_2^{H3} & \$D_2^{H4} & \$D_2^{H5} \\ \$D_3^{H1} & \$D_3^{H2} & \$D_3^{H3} & \$D_3^{H4} & \$D_3^{H5} \\ B_0^{H1} & B_0^{H2} & B_0^{H3} & B_0^{H4} & B_0^{H5} \end{pmatrix}^{-1} \times \begin{pmatrix} -\$D_0^P \\ -\$D_1^P \\ -\$D_2^P \\ -\$D_3^P \\ -PV_P \end{pmatrix}$$

The vector first vector represents the hedge ratios assigned to each hedging instrument. The matrix represents the inverse of a matrix containing the dollar durations ($D) and bond prices (B0) for the hedging instruments. The vector comprises the negative dollar durations and the present value of the portfolio. By multiplying the inverse matrix with this vector, we obtain the optimal hedge ratios. These ratios guide us in strategically allocating weights to each hedging instrument, effectively mitigating interest rate risk in the portfolio.

```
1  dollar_duration_matrix = hedging_instruments[['$D0','$D1','$D2','$D3']].T
2  dollar_duration_matrix
```

	1	2	3	4	5
$D0	-369.110696	-850.678215	-1008.870331	-1276.296357	-1657.114117
$D1	-256.866378	-376.096588	-411.896844	-424.561965	-454.819602
$D2	-86.454640	-244.706135	-281.838808	-318.647488	-351.412409
$D3	-48.036885	-47.802864	-51.790502	-51.878627	-55.361594

Here, we construct a matrix ("*dollar_duration_matrix*") containing the dollar durations ($D0 to $D3) for each hedging instrument. This matrix corresponds to the coefficients in the matrix equation.

```
1  hedging_instruments_bond_values = hedging_instruments[['B0']].T
2  hedging_instruments_bond_values
```

	1	2	3	4	5
B0	98.519792	98.04902	106.512956	106.689206	113.899468

This code generates a matrix ("*hedging_instruments_bond_values*") with the bond prices (B0) for each hedging instrument. This matrix is an integral part of the overall matrix equation.

Here, we combine the dollar duration matrix with the bond values matrix, creating the complete matrix required for the inversion process in the matrix equation.

```
dollar_duration_matrix.loc['B0'] = hedging_instruments_bond_values.iloc[0]
dollar_duration_matrix
```

✓ 0.0s

	1	2	3	4	5
$D0	-369.110696	-850.678215	-1008.870331	-1276.296357	-1657.114117
$D1	-256.866378	-376.096588	-411.896844	-424.561965	-454.819602
$D2	-86.454640	-244.706135	-281.838808	-318.647488	-351.412409
$D3	-48.036885	-47.802864	-51.790502	-51.878627	-55.361594
B0	98.519792	98.049020	106.512956	106.689206	113.899468

This code assembles the right-hand side vector ("*solutions*"), incorporating the negative dollar durations of the portfolio and the portfolio's present value.

```
solutions = -np.concatenate([portfolio_dollar_durations.T, [portfolio_pv]]).T
solutions
```

The NumPy linear algebra solver is employed to find the optimal hedge ratios (phis), representing the weights assigned to each hedging instrument:

```
1  phis = np.linalg.solve(dollar_duration_matrix, solutions)
2  for i in range(0, len(phis)): print(f"phi_{i} = {phis[i]}")
```

```
phi_0 = 6711058.475930818
phi_1 = 314116164.09006244
phi_2 = -484306031.66594076
phi_3 = 263094572.27366638
phi_4 = -70528371.23473544
```

Finally, the calculated hedge ratios are applied to the hedging instruments, and the difference between the portfolio's present value and the hedging portfolio's present value is evaluated. This difference serves as an indicator of the effectiveness of the hedging strategy.

```
1  print("Difference between portfolio PV and hedging portfolio PV:", round(hedging_instruments['B0'] @ phis + portfolio_pv, 5))
```

```
Difference between portfolio PV and hedging portfolio PV: -3e-05
```

Task D – Assuming that the yield curve changed

Part 1 – With no hedging strategy

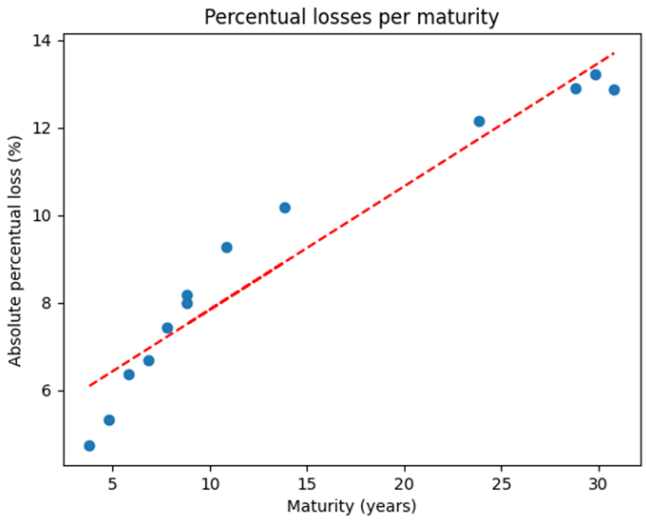
In response to the refined yield curve parameters, we recalculated the bond prices (B0*) for each instrument in our portfolio. This adjustment allows us to assess how changes in the yield curve impact individual bond valuations.

```
portfolio_instruments['B0*'] = portfolio_instruments.apply(
    lambda row: NSS_Sens(**new_yield_curve_params, coupon_rate=row['coupon']/100, maturity=round(count_years(valuation_date, row['maturity'])))['B0'],
    axis = 1
)
portfolio_instruments[['coupon', 'maturity', 'B0', 'B0*']]
```

	coupon	maturity	B0	B0*
1	4.00	December 1st, 2025	96.741840	92.160777
2	7.75	December 4th, 2026	112.078569	106.116564
3	4.00	December 6th, 2027	94.818814	88.784257
4	7.00	December 10th, 2028	111.184284	103.748234
5	5.75	December 3rd, 2029	104.097349	96.378853
6	5.50	December 9th, 2030	102.416585	94.241060
7	4.00	December 6th, 2032	89.786788	81.470311
8	4.75	December 3rd, 2035	94.346398	84.751810
9	4.50	December 3rd, 2030	95.321640	87.534081
10	5.00	December 4th, 2045	93.343343	81.996069
11	4.50	December 4th, 2050	84.936321	73.982577
12	4.00	December 1st, 2051	77.327677	67.109312
13	5.00	December 1st, 2052	91.767240	79.966453

We enhance our understanding by visualizing the absolute percentual losses relative to maturity. The scatter plot depicts how changes in the yield curve influence various bond maturities.

```
losses_per_maturity_x = portfolio_instruments['maturity'].apply(lambda m: count_years(valuation_date, m))
losses_per_maturity_y = abs((portfolio_instruments['B0*'] - portfolio_instruments['B0'])/portfolio_instruments['B0'])*100
```



Finally, we compute the expected losses resulting from the yield curve adjustments and update the portfolio's present value accordingly. This assessment provides insights into the monetary impact of the modified yield curve on our portfolio.

```
portfolio_losses_no_hedging = ((portfolio_instruments['B0*']-portfolio_instruments['B0']) * portfolio_instruments['quantity']).sum()
portfolio_new_pv_no_hedging = portfolio_pv + portfolio_losses_no_hedging

print(f"Expected losses for the shifts in yield curve: {portfolio_losses_no_hedging}")
print(f"Previous portfolio PV was {portfolio_pv}, now will be {portfolio_new_pv_no_hedging}")
print(f"Percentual losses: {(portfolio_new_pv_no_hedging-portfolio_pv)/portfolio_pv * 100}%")
```

✓ 0.0s

```
Expected losses for the shifts in yield curve: -6914865.942429821
Previous portfolio PV was 88705967.2539232, now will be 81791101.31149337
Percentual losses: -7.795265816374944%
```

Part 2 – With hedging strategy

We extend our analysis to the hedging instruments, recalculating bond prices (B0*) under the refined yield curve parameters. This step ensures a comprehensive examination of the impact on portfolio and hedging instruments.

```
hedging_instruments['B0*'] = hedging_instruments.apply(
    lambda row: NSS_Sens(**new_yield_curve_params, coupon_rate=row['coupon']/100, maturity=round(count_days(valuation_date, row['maturity'])))['B0'],
    axis = 1
)
```

	coupon	maturity	B0	B0*
1	4.5	April 12th, 2026	98.519792	68.795238
2	5.0	December 28th, 2032	98.049020	76.439154
3	6.0	May 6th, 2035	106.512956	91.726984
4	6.0	October 10th, 2040	106.689206	91.726984
5	6.5	October 10th, 2051	113.899468	99.370900

Incorporating the hedging instruments, we assess the overall impact on the portfolio. The code computes the portfolio loss after hedging, accounting for both bond price changes and the adjusted portfolio present value. This final measure provides a comprehensive understanding of the effectiveness of our hedging strategy in mitigating losses associated with yield curve shifts.

```
portfolio_loss = (hedging_instruments['B0*']-hedging_instruments['B0']) @ phis \
    + (portfolio_new_pv_no_hedging-portfolio_pv)

print(f"Portfolio loss after hedging was: {portfolio_loss/portfolio_pv*100}%")
```

Portfolio loss after hedging was: -3094.802045919286%

The result obtained is too big and it mirrors an error made during the calculations, this error was probably when calculating the phis or the asset pricing.

Conclusions

This analysis conducted a comprehensive evaluation of an inflation-linked bond (ILB) and a corresponding hedging strategy aimed at managing interest rate risk.

For the ILB, the assessment involved calculating accrued interest, simulating inflation scenarios, and analyzing cash flows to understand variability influenced by inflation and interest rates. Additionally, bond price distribution analysis was performed to assess interest rate risk using a specific methodology.

In parallel, the hedging strategy was developed to minimize interest rate risk by conducting a sensitivity analysis of hedging instruments and portfolio metrics. Optimal hedge ratios were computed to mitigate risk effectively, and results included key metrics such as level, slope, and curvature durations, along with dollar durations for comprehensive risk assessment.

Finally, the impact of changes in the yield curve was evaluated, both with and without the hedging strategy. Visualizations and calculations demonstrated the effectiveness of the hedging strategy in mitigating losses associated with yield curve shifts.

Overall, the analysis underscored the critical importance of robust risk management strategies in investment portfolios, particularly when dealing with complex financial instruments like ILBs. By integrating historical data, future projections, and risk assessments, investors can make informed decisions to optimize portfolio performance and minimize potential losses in dynamic market conditions.