

EIOPA RISK-FREE CURVE JULY-23 RECALCULATION

The risk-free curve is one of the principal inputs into an economic scenario generator. This notebook recalculates the risk-free curve using the parameters that are claimed to be used. The European Insurance and Occupational Pensions Authority (EIOPA) publishes their own yield curve prediction. To do this they use the Smith & Wilson algorithm.

Summary

The goal of this test is to replicate the EIOPA yield curve. This test will use the methodology that EIOPA claims it is using and the calibration vector that they publish. If the test is passed, the user can be more confident, that EIOPA risk free rate (RFR) curve was generated using the described methodology/calibration and that the process was implemented correctly.

Table of Contents

1. [Note on Smith & Wilson algorithm](#)
2. [Data requirements](#)
3. [Success criteria](#)
4. [External dependencies](#)
5. [Calibration parameters and calibration vector provided by EIOPA](#)
6. [Smith & Wilson calculation functions](#)
7. [Generation of the risk-free curve](#)
8. [Test 1; Comparison test](#)
9. [Test 1; Success criteria](#)
10. [Test 1; Comparison test](#)
11. [Conclusion](#)

Note on Smith & Wilson algorithm

To replicate the calculations, this example uses a modified Smith&Wilson implementation (The original implementation is available on [GitHub](#)):

- [Python](#)
- [Matlab](#)
- [JavaScript](#)

Limitations of the implementation

Current implementation only looks at a single currency and with/without Volatility Adjustment (VA). The day count convention assumes that each year has the same number of days.

Data requirements

This script contains the EIOPA risk-free rate publication for July 2023. The publication can be found on the [EIOPA RFR website](#).

The observed maturities `M_Obs` and the calibrated vector `Qb` can be found in the Excel sheet `EIOPA_RFR_20230731_Qb_SW.xlsx`.

The target maturities (`T_Obs`), the additional parameters (`UFR` and `alpha`), and the given curve can be found in the Excel `EIOPA_RFR_20230731_Term_Structures.xlsx`, sheet `RFR_spot_no_VA` if the test looks at the curve without the Volatility Adjustment and the sheet `RFR_spot_with_VA` if the test looks at the curve with the Volatility Adjustment.

[Back to the top](#)

Success criteria

The following success criteria is defined:

- Maximum difference between the calculated curve and the one provided by EIOPA is less than 0.1 bps
- Average difference between the calculated curve and the one provided by EIOPA is less than 0.05 bps

In [1]:

```
test_statistics_max_diff_in_bps = 0.1
test_statistics_average_diff_in_bps = 0.05
```

The success function is called at the end of the test to confirm if the success criteria have been met.

```
In [2]: def SuccessTest(TestStatistics, threshold_max, threshold_mean):
    out1 = False
    out2 = False
    if max(TestStatistics) < threshold_max:
        print("Test passed")
        out1 = True
    else:
        print("Test failed")

    if np.mean(TestStatistics) < threshold_mean:
        print("Test passed")
        out2 = True
    else:
        print("Test failed")
    return [out1, out2]
```

[Back to the top](#)

External dependencies

This implementation uses three well established Python packages widely used in the financial industry. Pandas (<https://pandas.pydata.org/docs/>), Numpy (<https://numpy.org/doc/>), and Matplotlib (<https://matplotlib.org/stable/index.html>)

```
In [3]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
%matplotlib notebook
pd.options.display.max_rows = 150
```

Importing data

```
In [4]: #selected_param_file = 'Param_VA.csv'
#selected_curves_file = 'Curves_VA.csv'

selected_param_file = 'Param_no_VA.csv'
selected_curves_file = 'Curves_no_VA.csv'
```

```
In [5]: param_raw = pd.read_csv(selected_param_file, sep=',', index_col=0)
```

Parameter input

Parameters sheet

In [6]: `param_raw.head()`

Out[6]:

| | Euro_Maturities | Euro_Values | Austria_Maturities | Austria_Values | Belgium_Maturities | Be |
|--------------------|-----------------|-------------|--------------------|----------------|--------------------|-----------|
| Country | | | | | | |
| Coupon_freq | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |
| LLP | 20.000000 | 20.000000 | 20.000000 | 20.000000 | 20.000000 | 20.000000 |
| Convergence | 40.000000 | 40.000000 | 40.000000 | 40.000000 | 40.000000 | 40.000000 |
| UFR | 3.450000 | 3.450000 | 3.450000 | 3.450000 | 3.450000 | 3.450000 |
| alpha | 0.112203 | 0.112203 | 0.112203 | 0.112203 | 0.112203 | 0.112203 |

5 rows × 106 columns

The country selected is:

In [7]: `country = "Slovenia"`

In [8]:

```
maturities_country_raw = param_raw.loc[:,country+"_Maturities"].iloc[6:]
param_country_raw = param_raw.loc[:,country + "_Values"].iloc[6:]
extra_param = param_raw.loc[:,country + "_Values"].iloc[:6]
```

Extra parameters

Smith-Wilson calibration parameters

In [9]: `extra_param`

Out[9]:

| Country | |
|-------------|-----------|
| Coupon_freq | 1.000000 |
| LLP | 20.000000 |
| Convergence | 40.000000 |
| UFR | 3.450000 |
| alpha | 0.112203 |
| CRA | 10.000000 |

Name: Slovenia_Values, dtype: float64

In [10]: `relevant_positions = pd.notna(maturities_country_raw.values)`

```
In [11]: maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

```
In [12]: maturities_country.head(15)
```

```
Out[12]: Country
1      1.0
2      2.0
3      3.0
4      4.0
5      5.0
6      6.0
7      7.0
8      8.0
9      9.0
10     10.0
11     11.0
12     12.0
13     13.0
14     14.0
15     15.0
Name: Slovenia_Maturities, dtype: float64
```

```
In [13]: Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector **Qb** provided as input

```
In [14]: Qb
```

```
Out[14]: Country
1      -8.182191
2       0.658896
3       0.666484
4       0.727548
5       1.361903
6      -2.307430
7       2.493323
8      -2.331474
9       4.696474
10      -9.428838
11      11.003868
12      -4.824905
13      -0.004031
14      -0.003897
15      -0.761741
16       0.018179
```

```
17      0.017572
18      0.016986
19      0.016420
20      0.570069
Name: Slovenia Values, dtype: float64
```

```
In [15]: curve_raw = pd.read_csv(selected_curves_file, sep=',', index_col=0)
```

```
In [16]: curve_country = curve_raw.loc[:,country]
```

[Back to the top](#)

Calibration parameters and calibration vector provided by EIOPA

```
In [17]: # Maturity of observations:
M_Obs = np.transpose(np.array(maturities_country.values))

# Ultimate forward rate ufr represents the rate to which the rate curve will converge
ufr = extra_param.iloc[3]/100

# Convergence speed parameter alpha controls the speed at which the curve converges
alpha = extra_param.iloc[4]

# For which maturities do we want the SW algorithm to calculate the rates. In this case, we want rates for all maturities
M_Target = np.transpose(np.arange(1,151))

# Qb calibration vector published by EIOPA for the curve calibration:
Qb = np.transpose(np.array(Qb.values))
```

[Back to the top](#)

Smith & Wilson calculation functions

In this step, the independent version of the Smith&Wilson algorithm is implemented. To do this, two functions are taken from the publicly available repository and modified to accept the product of Q^*b instead of the calibration vector b .

In [18]:

```

def SWExtrapolate(M_Target, M_Obs, Qb, ufr, alpha):
    # SWEXTRAPOLATE Interpolate or/and extrapolate rates for targeted maturities using a
    # out = SWExtrapolate(M_Target, M_Obs, Qb, ufr, alpha) calculates the rates for maturities
    #
    # Arguments:
    #     M_Target = k x 1 ndarray. Each element represents a bond maturity of interest. Ex. M_Target = [1; 3]
    #     M_Obs = n x 1 ndarray. Observed bond maturities used for calibrating the calibration vector.
    #     Qb = n x 1 ndarray. Calibration vector calculated on observed bonds.
    #     ufr = 1 x 1 floating number. Representing the ultimate forward rate.
    #         Ex. ufr = 0.042
    #     alpha = 1 x 1 floating number. Representing the convergence speed parameter alpha.
    #
    #
    # Returns:
    #     k x 1 ndarray. Represents the targeted rates for a zero-coupon bond. Each rate is
    #
    # For more information see https://www.eiopa.europa.eu/sites/default/files/risk_free_curve.ipynb

def SWHeart(u, v, alpha):
    # SWHEART Calculate the heart of the Wilson function.
    # H = SWHeart(u, v, alpha) calculates the matrix H (Heart of the Wilson function) for maturities specified by vectors u and v. The formula is
    # taken from the EIOPA technical specifications paragraph 132.
    #
    # Arguments:
    #     u = n_1 x 1 vector of maturities. Ex. u = [1; 3]
    #     v = n_2 x 1 vector of maturities. Ex. v = [1; 2; 3; 5]
    #     alpha = 1 x 1 floating number representing the convergence speed parameter alpha.
    #
    # Returns:
    #     n_1 x n_2 matrix representing the Heart of the Wilson function for selected maturities.
    #
    # For more information see https://www.eiopa.europa.eu/sites/default/files/risk_free_curve.ipynb

    u_Mat = np.tile(u, [v.size, 1]).transpose()
    v_Mat = np.tile(v, [u.size, 1])
    return 0.5 * (alpha * (u_Mat + v_Mat) + np.exp(-alpha * (u_Mat + v_Mat)) - alpha)

H = SWHeart(M_Target, M_Obs, alpha) # Heart of the Wilson function from paragraph 132
p = np.exp(-np.log(1+ufr)* M_Target) + np.diag(np.exp(-np.log(1+ufr) * M_Target))
return p ** (-1/ M_Target) -1 # Convert obtained prices to rates and return price

```

[Back to the top](#)

Generation of the risk-free curve

The observed maturities, target maturities, and the model parameters provided by EIOPA are used to generate the target curve.

In [19]:

```

r_Target = SWExtrapolate(M_Target,M_Obs, Qb, ufr, alpha)
r_Target = pd.DataFrame(r_Target,columns=['Recalculated rates'])

```

Yield curve calculated

Yield curve calculated using the calibration vector **Qb**

In [20]:

```
r_Target.head(15)
```

Out[20]:

Recalculated rates

| | |
|----|----------|
| 0 | 0.039190 |
| 1 | 0.036105 |
| 2 | 0.033572 |
| 3 | 0.031876 |
| 4 | 0.030860 |
| 5 | 0.030269 |
| 6 | 0.029863 |
| 7 | 0.029679 |
| 8 | 0.029601 |
| 9 | 0.029608 |
| 10 | 0.029543 |
| 11 | 0.029721 |
| 12 | 0.029837 |
| 13 | 0.029811 |
| 14 | 0.029656 |

[Back to the top](#)

Test 1; Comparison test

Comparison of the calculated yield curve with the yield curve provided by EIOPA. The test is passed if the success criteria is reached.

The provided yield curve can be found in file *EIOPA_RFR_20230731_Term_Structures.xlsx*, sheet *RFR_spot_no_VA* if the test looks at the curve without the Volatility Adjustment and the sheet *RFR_spot_with_VA* if the test looks at the curve with the Volatility Adjustment.

In [21]:

```
target_curve = np.transpose(np.array(curve_country.values))
```

This implementation looks at two kinds of test statistics. The average deviation and the maximum deviation.

The average deviation is defined as:

$$S_{AVERAGE} = \frac{1}{T} \sum_{t=0}^T |r_{EIOPA}(t) - r_{EST}(t)|$$

The maximum deviation is defined as:

$$S_{MAX} = \max_t |r_{EIOPA}(t) - r_{EST}(t)|$$

Where T is the maximum maturity available.

The average difference test is successful if:

$$S_{AVERAGE} < 0.05bps$$

The maximum difference test is successful if:

$$S_{MAX} < 0.1bps$$

```
In [22]: target_curve = pd.DataFrame(target_curve,columns=['Given rates'])
```

EIOPA curve provided

Yield curve provided by EIOPA

```
In [23]: target_curve.head()
```

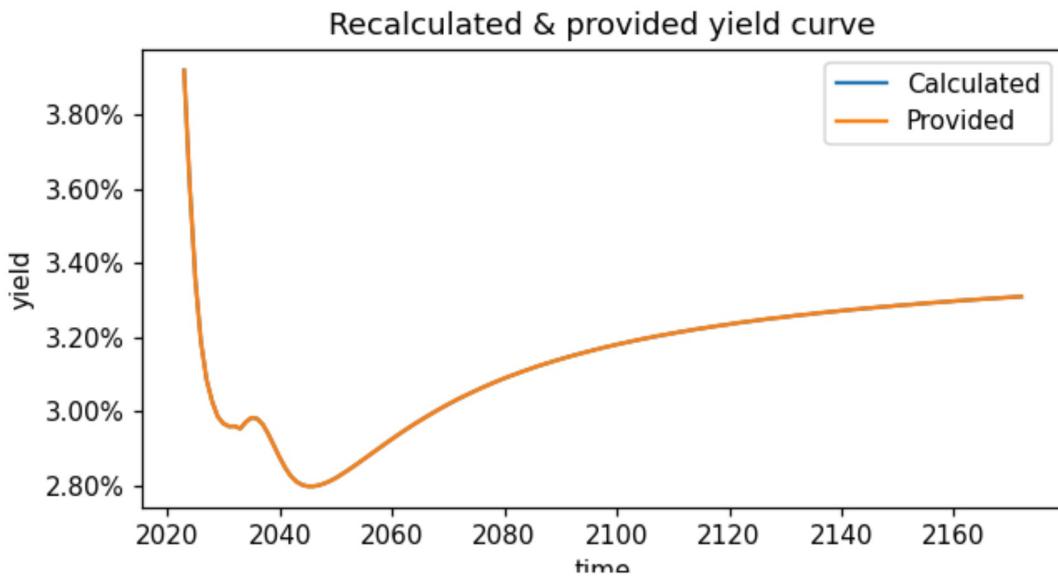
Out[23]: **Given rates**

| | |
|----------|---------|
| 0 | 0.03919 |
| 1 | 0.03611 |
| 2 | 0.03357 |
| 3 | 0.03188 |
| 4 | 0.03086 |

```
In [24]: x_data_label = range(2023,2023+r_Target.shape[0],1)
```

```
In [25]: fig, ax1 = plt.subplots(1,1)
ax1.plot(x_data_label, r_Target.values*100, color='tab:blue',label="Calculated")
ax1.plot(x_data_label, target_curve.values*100, color='tab:orange',label="Provided")

ax1.set_ylabel("yield")
ax1.set_title('Recalculated & provided yield curve')
ax1.set_xlabel("time")
ax1.legend()
ax1.yaxis.set_major_formatter(mtick.PercentFormatter())
fig.set_figwidth(6)
fig.set_figheight(3)
plt.show()
```



```
In [26]: test_statistics_bdp = pd.DataFrame(abs(r_Target.values-target_curve.values)*10000, co
```

EIOPA curve comparison

Absolute difference in bps

```
In [27]: test_statistics_bdp.head()
```

Out[27]: **Abs diff in bps**

| | |
|---|----------|
| 0 | 0.000005 |
| 1 | 0.046189 |
| 2 | 0.024584 |
| 3 | 0.040507 |

Abs diff in bps[Back to the top](#)

Test 1; Success criteria

The successful application of the success criteria marks the completion of the test.

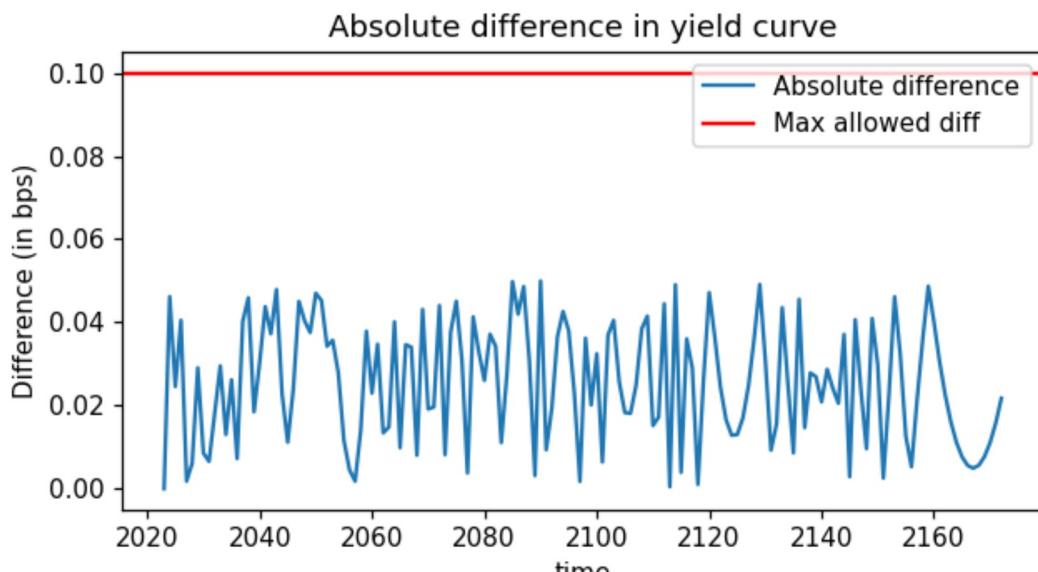
In [28]:

```
result1 = SuccessTest(test_statistics_bdp.values, test_statistics_max_diff_in_bps, tolerance)
```

```
Test passed  
Test passed
```

In [29]:

```
x_data_label = range(2023,2023+r_Target.shape[0],1)  
fig, ax1 = plt.subplots(1,1)  
ax1.plot(x_data_label, test_statistics_bdp, label= "Absolute difference")  
ax1.axhline(y = test_statistics_max_diff_in_bps, color = 'r', linestyle = '--',label='Max allowed diff')  
  
ax1.set_xlabel("time")  
ax1.set_ylabel("Difference (in bps)")  
ax1.set_title('Absolute difference in yield curve')  
ax1.legend()  
fig.set_figwidth(6)  
fig.set_figheight(3)  
  
plt.show()
```



[Back to the top](#)

Conclusion

This test checks the success criteria on the EIOPA curve generated for July 2023. If the tests are passed, it is likely that the curve was generated using the Smith & Wilson algorithm with the calibration vector that was provided in the file *EIOPA_RFR_20230731_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230731_Term_Structures.xlsx*.

In [30]:

```
pd.DataFrame(data = [result1], columns = ["Mean test", "Max test"], \
              index= ["Provided vs calculated"])
```

Out[30]:

| | Mean test | Max test |
|------------------------|-----------|----------|
| Provided vs calculated | True | True |

Final yield curve

Full yield curve provided by EIOPA in %

In [31]:

```
(curve_country*100).head(150)
```

Out[31]:

| Country | |
|---------|-------|
| 1 | 3.919 |
| 2 | 3.611 |
| 3 | 3.357 |
| 4 | 3.188 |
| 5 | 3.086 |
| 6 | 3.027 |
| 7 | 2.986 |
| 8 | 2.968 |
| 9 | 2.960 |
| 10 | 2.961 |
| 11 | 2.954 |
| 12 | 2.972 |
| 13 | 2.984 |
| 14 | 2.981 |
| 15 | 2.966 |
| 16 | 2.939 |
| 17 | 2.908 |
| 18 | 2.877 |
| 19 | 2.849 |
| 20 | 2.826 |
| 21 | 2.811 |
| 22 | 2.803 |
| 23 | 2.799 |
| 24 | 2.799 |
| 25 | 2.802 |
| 26 | 2.807 |
| 27 | 2.813 |
| 28 | 2.821 |

| | |
|----|-------|
| 29 | 2.831 |
| 30 | 2.840 |
| 31 | 2.851 |
| 32 | 2.861 |
| 33 | 2.872 |
| 34 | 2.883 |
| 35 | 2.894 |
| 36 | 2.905 |
| 37 | 2.916 |
| 38 | 2.926 |
| 39 | 2.937 |
| 40 | 2.947 |
| 41 | 2.957 |
| 42 | 2.967 |
| 43 | 2.976 |
| 44 | 2.985 |
| 45 | 2.994 |
| 46 | 3.003 |
| 47 | 3.012 |
| 48 | 3.020 |
| 49 | 3.028 |
| 50 | 3.036 |
| 51 | 3.043 |
| 52 | 3.050 |
| 53 | 3.057 |
| 54 | 3.064 |
| 55 | 3.071 |
| 56 | 3.077 |
| 57 | 3.084 |
| 58 | 3.090 |
| 59 | 3.096 |
| 60 | 3.101 |
| 61 | 3.107 |
| 62 | 3.112 |
| 63 | 3.118 |
| 64 | 3.123 |
| 65 | 3.128 |
| 66 | 3.132 |
| 67 | 3.137 |
| 68 | 3.141 |
| 69 | 3.146 |
| 70 | 3.150 |
| 71 | 3.154 |
| 72 | 3.158 |
| 73 | 3.162 |
| 74 | 3.166 |
| 75 | 3.170 |
| 76 | 3.174 |
| 77 | 3.177 |
| 78 | 3.181 |
| 79 | 3.184 |
| 80 | 3.187 |
| 81 | 3.191 |
| 82 | 3.194 |
| 83 | 3.197 |
| 84 | 3.200 |
| 85 | 3.203 |
| 86 | 3.206 |
| 87 | 3.208 |

| | |
|-----|-------|
| 88 | 3.211 |
| 89 | 3.214 |
| 90 | 3.216 |
| 91 | 3.219 |
| 92 | 3.222 |
| 93 | 3.224 |
| 94 | 3.226 |
| 95 | 3.229 |
| 96 | 3.231 |
| 97 | 3.233 |
| 98 | 3.235 |
| 99 | 3.238 |
| 100 | 3.240 |
| 101 | 3.242 |
| 102 | 3.244 |
| 103 | 3.246 |
| 104 | 3.248 |
| 105 | 3.250 |
| 106 | 3.252 |
| 107 | 3.253 |
| 108 | 3.255 |
| 109 | 3.257 |
| 110 | 3.259 |
| 111 | 3.261 |
| 112 | 3.262 |
| 113 | 3.264 |
| 114 | 3.266 |
| 115 | 3.267 |
| 116 | 3.269 |
| 117 | 3.270 |
| 118 | 3.272 |
| 119 | 3.273 |
| 120 | 3.275 |
| 121 | 3.276 |
| 122 | 3.278 |
| 123 | 3.279 |
| 124 | 3.280 |
| 125 | 3.282 |
| 126 | 3.283 |
| 127 | 3.284 |
| 128 | 3.286 |
| 129 | 3.287 |
| 130 | 3.288 |
| 131 | 3.289 |
| 132 | 3.291 |
| 133 | 3.292 |
| 134 | 3.293 |
| 135 | 3.294 |
| 136 | 3.295 |
| 137 | 3.296 |
| 138 | 3.298 |
| 139 | 3.299 |
| 140 | 3.300 |
| 141 | 3.301 |
| 142 | 3.302 |
| 143 | 3.303 |
| 144 | 3.304 |
| 145 | 3.305 |
| 146 | 3.306 |

147 3.307
148 3.308
149 3.309
150 3.310