

EIOPA RISK-FREE CURVE JUNE-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 June 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_20230630_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_20230630_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 [63]:

```
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 [64]:  
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 [65]:  
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 [66]:  
#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 [67]:  
param_raw = pd.read_csv(selected_param_file, sep=',', index_col=0)
```

Parameter input

Parameters sheet

In [68]: `param_raw.head()`

Out[68]:

	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	0.116339	0.116339	0.116339	0.116339	0.116339	0.116339

5 rows × 106 columns

The country selected is:

In [69]: `country = "Slovenia"`

In [70]:

```
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 [71]: `extra_param`

Out[71]:

```
Country
Coupon_freq      1.000000
LLP              20.000000
Convergence      40.000000
UFR              3.450000
alpha             0.116339
CRA               10.000000
Name: Slovenia_Values, dtype: float64
```

In [72]: `relevant_positions = pd.notna(maturities_country_raw.values)`

```
In [73]: maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

```
In [74]: maturities_country.head(15)
```

```
Out[74]: 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 [75]: Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector **Qb** provided as input

```
In [76]: Qb
```

```
Out[76]: Country
1      -1.253807
2      -4.033131
3      1.009861
4      0.515163
5      1.337815
6      -1.035960
7      0.799522
8      -0.419684
9      0.116155
10     0.986995
11     -1.622364
12     1.380541
13     -0.017246
14     -0.016671
15     -1.274046
16     0.019054
```

```
17    0.018419
18    0.017804
19    0.017210
20    0.630078
Name: Slovenia Values, dtype: float64
```

```
In [77]: curve_raw = pd.read_csv(selected_curves_file, sep=',', index_col=0)
```

```
In [78]: curve_country = curve_raw.loc[:,country]
```

[Back to the top](#)

Calibration parameters and calibration vector provided by EIOPA

```
In [79]: # 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, all
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 [80]:

```

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 [81]:

```

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 [82]:

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

Out[82]:

Recalculated rates

0	0.039830
1	0.037721
2	0.035013
3	0.032821
4	0.031323
5	0.030365
6	0.029697
7	0.029260
8	0.028969
9	0.028795
10	0.028695
11	0.028590
12	0.028506
13	0.028393
14	0.028191

[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_20230630_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 [83]:

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

EIOPA curve provided

Yield curve provided by EIOPA

```
In [85]: target_curve.head()
```

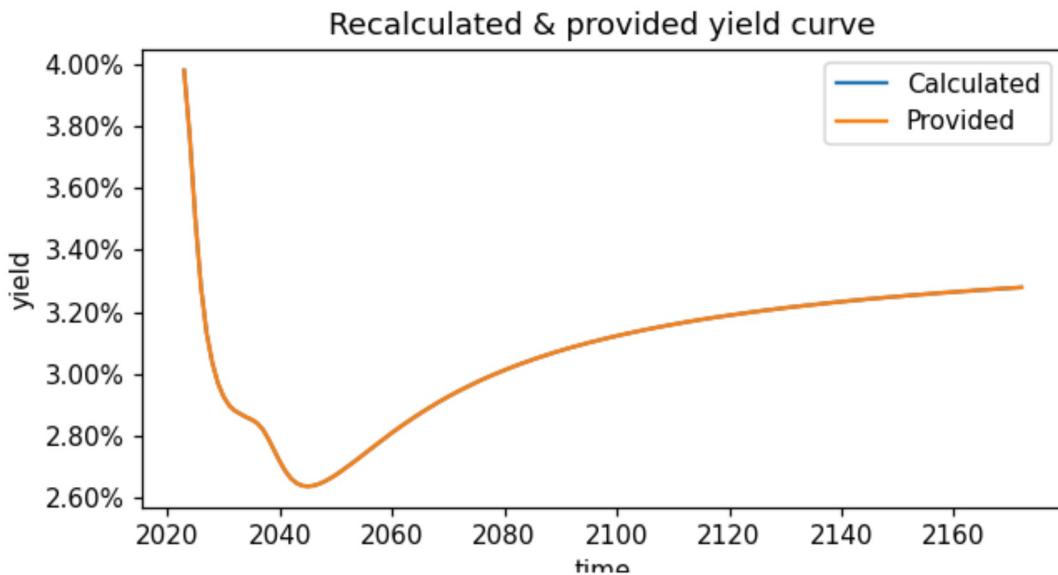
Out[85]: **Given rates**

0	0.03983
1	0.03772
2	0.03501
3	0.03282
4	0.03132

```
In [86]: x_data_label = range(2023,2023+r_Target.shape[0],1)
```

```
In [87]: 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 [88]: test_statistics_bdp = pd.DataFrame(abs(r_Target.values-target_curve.values)*10000, co
```

EIOPA curve comparison

Absolute difference in bps

```
In [89]: test_statistics_bdp.head()
```

Out[89]: **Abs diff in bps**

0	8.571525e-07
1	9.984847e-03
2	2.859551e-02
3	1.251408e-02

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.

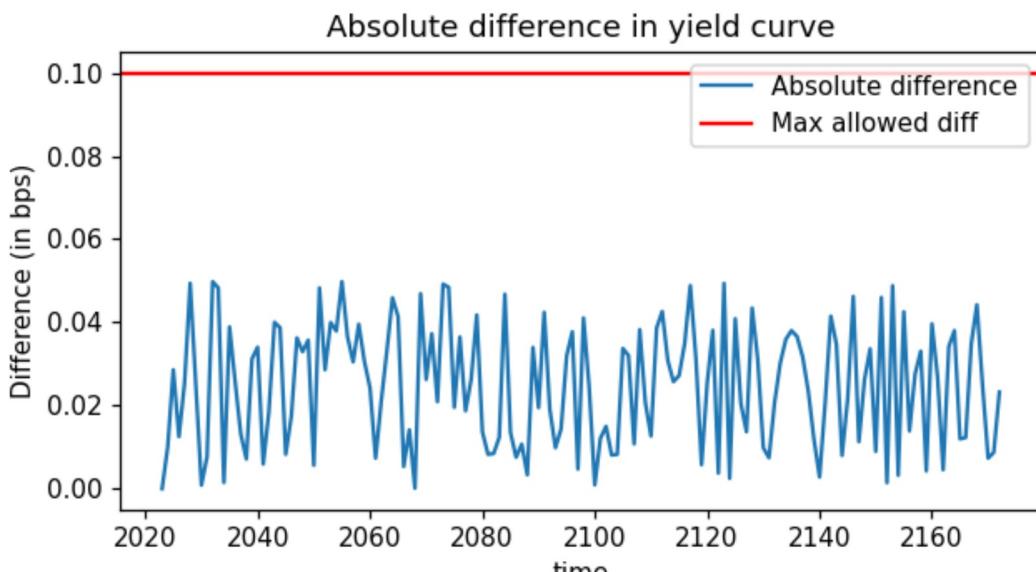
```
In [90]: result1 = SuccessTest(test_statistics_bdp.values, test_statistics_max_diff_in_bps, tolerance=0.01)
```

Test passed
Test passed

```
In [91]: 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 June 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_20230630_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230630_Term_Structures.xlsx*.

In [92]:

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

Out[92]:

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [93]:

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

Out[93]:

Country	
1	3.983
2	3.772
3	3.501
4	3.282
5	3.132
6	3.037
7	2.970
8	2.926
9	2.897
10	2.879
11	2.869
12	2.859
13	2.851
14	2.839
15	2.819
16	2.788
17	2.752
18	2.716
19	2.684
20	2.660
21	2.644
22	2.636
23	2.634
24	2.636
25	2.642
26	2.650
27	2.660
28	2.671

29	2.684
30	2.697
31	2.710
32	2.724
33	2.738
34	2.753
35	2.767
36	2.781
37	2.794
38	2.808
39	2.821
40	2.834
41	2.846
42	2.859
43	2.870
44	2.882
45	2.893
46	2.904
47	2.915
48	2.925
49	2.935
50	2.944
51	2.953
52	2.962
53	2.971
54	2.980
55	2.988
56	2.996
57	3.003
58	3.011
59	3.018
60	3.025
61	3.032
62	3.038
63	3.045
64	3.051
65	3.057
66	3.063
67	3.069
68	3.074
69	3.080
70	3.085
71	3.090
72	3.095
73	3.100
74	3.104
75	3.109
76	3.113
77	3.118
78	3.122
79	3.126
80	3.130
81	3.134
82	3.138
83	3.142
84	3.145
85	3.149
86	3.152
87	3.156

88	3.159
89	3.162
90	3.166
91	3.169
92	3.172
93	3.175
94	3.178
95	3.181
96	3.183
97	3.186
98	3.189
99	3.191
100	3.194
101	3.196
102	3.199
103	3.201
104	3.204
105	3.206
106	3.208
107	3.211
108	3.213
109	3.215
110	3.217
111	3.219
112	3.221
113	3.223
114	3.225
115	3.227
116	3.229
117	3.231
118	3.233
119	3.235
120	3.237
121	3.238
122	3.240
123	3.242
124	3.243
125	3.245
126	3.247
127	3.248
128	3.250
129	3.251
130	3.253
131	3.254
132	3.256
133	3.257
134	3.259
135	3.260
136	3.262
137	3.263
138	3.264
139	3.266
140	3.267
141	3.268
142	3.270
143	3.271
144	3.272
145	3.273
146	3.275

147 3.276
148 3.277
149 3.278
150 3.279