

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 [187...]

```
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 [188...]

```
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 [189...]

```
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 [190...]

```
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 [191...]

```
param_raw = pd.read_csv(selected_param_file, sep=',', index_col=0)
```

Parameter input

Parameters sheet

In [192...]

```
param_raw.head()
```

Out[192...]

	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	0.108242	0.108242	0.108242	0.108242	0.108242	0.108242

5 rows × 106 columns

The country selected is:

In [193...]

```
country = "United Kingdom"
```

In [194...]

```
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 [195...]

```
extra_param
```

Out[195...]

Country	
Coupon_freq	1.000000
LLP	50.000000
Convergence	40.000000
UFR	3.450000
alpha	0.087718
CRA	0.000000
Name:	United Kingdom_Values, dtype: float64

In [196...]

```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [197...]

```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [198...]

```
maturities_country.head(15)
```

Out[198...]

```
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: United Kingdom_Maturities, dtype: float64
```

In [199...]

```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector **Qb** provided as input

In [200...]

```
Qb
```

Out[200...]

```
Country
1      1.104838
2     -17.799558
3      11.135278
4     -3.350177
5      1.518820
6     -1.443938
7      1.998050
8      0.006595
9      0.162040
10     0.416511
11     -0.019788
12     -0.204071
13     -0.008776
14     -0.008395
15     -0.179079
16      0.000323
```

```
17      0.000198
18      0.000174
19      0.000191
20      0.114265
21     -0.004873
22     -0.004486
23     -0.004189
24     -0.003879
25     -0.003583
26     -0.003287
27     -0.003010
28     -0.002682
29     -0.002548
30     -0.325988
31      0.011806
32      0.010971
33      0.010401
34      0.009812
35      0.009276
36      0.008769
37      0.008294
38      0.007846
39      0.007426
40      0.007024
41      0.006661
42      0.006253
43      0.006105
44      0.005065
45      0.007430
46     -0.002850
47      0.034140
48     -0.105308
49      0.414054
50     -0.224940
Name: United Kingdom Values, dtype: float64
```

```
In [201...]: curve_raw = pd.read_csv(selected_curves_file, sep=',', index_col=0)
```

```
In [202...]: curve_country = curve_raw.loc[:, country]
```

[Back to the top](#)

Calibration parameters and calibration vector provided by EIOPA

In [203...]

```
# Maturity of observations:  
M_Obs = np.transpose(np.array(maturities_country.values))  
  
# Ultimate forward rate ufr represents the rate to which the curve will converge  
ufr = extra_param.iloc[3]/100  
  
# Convergence speed parameter alpha controls the speed at which the curve converges to the ultimate forward rate  
alpha = extra_param.iloc[4]  
  
# For which maturities do we want the SW algorithm to calculate the rates. In this case 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 [204...]

```

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.
    #     M_Obs = n x 1 ndarray. Observed bond maturities used for calibrating the curve.
    #     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 of the
    #
    #
    # 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 of the
    #
    # Returns:
    #     n_1 x n_2 matrix representing the Heart of the Wilson function for selected
    #
    # 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 [205...]

```

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 [206...]

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

Out[206...]

Recalculated rates

0	0.059646
1	0.057491
2	0.054140
3	0.051433
4	0.049175
5	0.047321
6	0.045831
7	0.044721
8	0.043907
9	0.043312
10	0.042877
11	0.042541
12	0.042263
13	0.042018
14	0.041794

[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 [207...]

```
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 [208...]

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

EIOPA curve provided

Yield curve provided by EIOPA

In [209...]

```
target_curve.head()
```

Out[209...]

	Given rates
0	0.05965
1	0.05749
2	0.05414
3	0.05143
4	0.04917

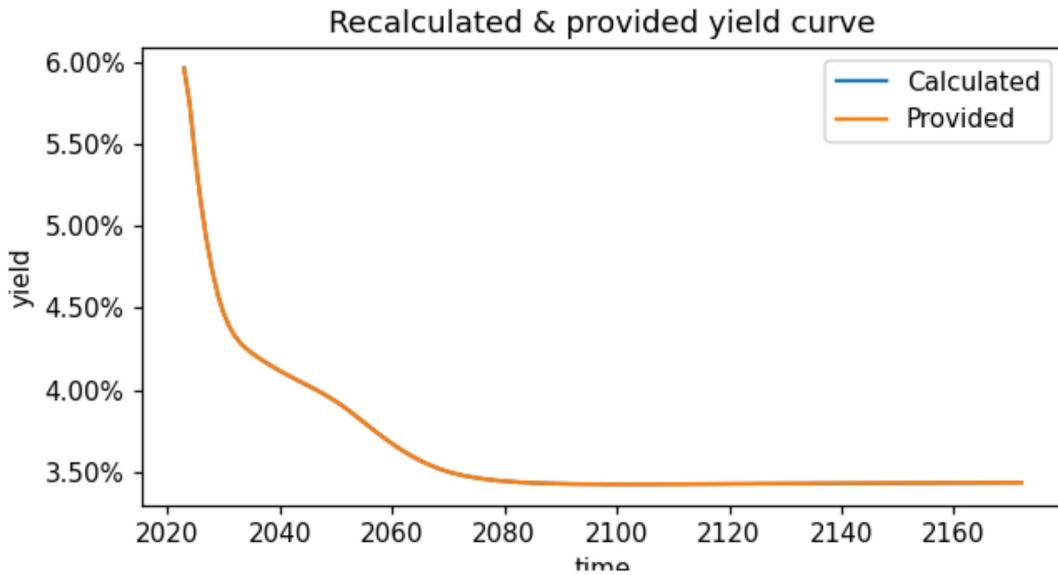
In [210...]

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

In [211...]

```
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 [212...]

```
test_statistics_bdp = pd.DataFrame(abs(r_Target.values-target_curve.values)*10000, co
```

EIOPA curve comparison

Absolute difference in bps

In [213...]

```
test_statistics_bdp.head()
```

Out[213...]

	Abs diff in bps
0	0.039999
1	0.013809
2	0.000588
3	0.027765

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.

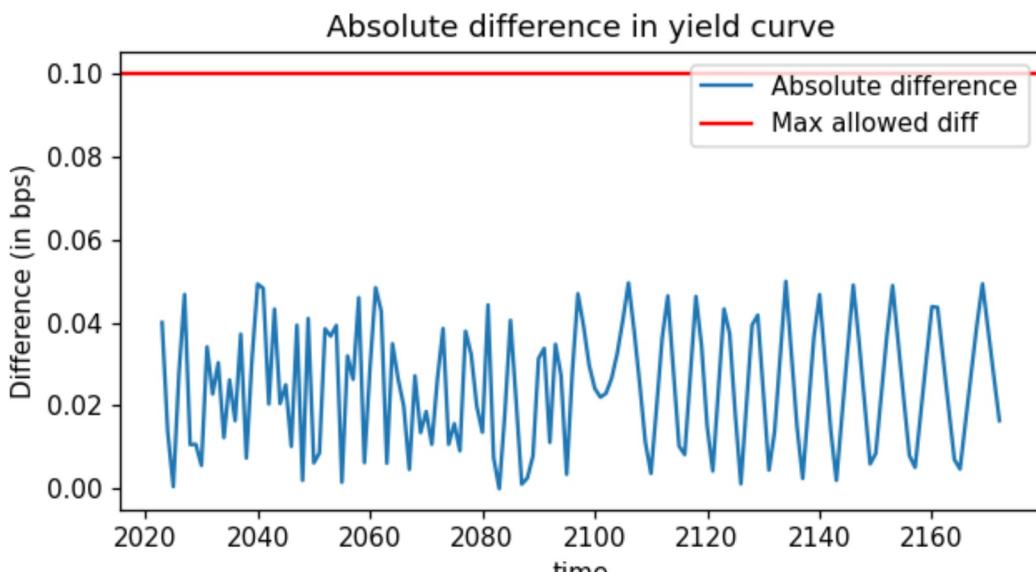
In [214...]

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

```
Test passed  
Test passed
```

In [215...]

```
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 [216...]

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

Out[216...]

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [217...]

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

Out[217...]

Country	
1	5.965
2	5.749
3	5.414
4	5.143
5	4.917
6	4.732
7	4.583
8	4.472
9	4.391
10	4.331
11	4.288
12	4.254
13	4.226
14	4.202
15	4.179
16	4.158
17	4.137
18	4.118
19	4.098
20	4.080
21	4.063
22	4.045
23	4.028
24	4.010
25	3.991
26	3.972
27	3.952
28	3.930

29	3.907
30	3.882
31	3.856
32	3.829
33	3.802
34	3.775
35	3.748
36	3.721
37	3.696
38	3.672
39	3.649
40	3.627
41	3.606
42	3.587
43	3.569
44	3.552
45	3.537
46	3.523
47	3.511
48	3.500
49	3.490
50	3.482
51	3.474
52	3.468
53	3.462
54	3.457
55	3.453
56	3.449
57	3.445
58	3.442
59	3.439
60	3.437
61	3.435
62	3.433
63	3.432
64	3.430
65	3.429
66	3.428
67	3.427
68	3.426
69	3.426
70	3.425
71	3.425
72	3.424
73	3.424
74	3.424
75	3.424
76	3.423
77	3.423
78	3.423
79	3.423
80	3.423
81	3.423
82	3.423
83	3.423
84	3.424
85	3.424
86	3.424
87	3.424

88	3.424
89	3.424
90	3.424
91	3.425
92	3.425
93	3.425
94	3.425
95	3.425
96	3.425
97	3.426
98	3.426
99	3.426
100	3.426
101	3.426
102	3.427
103	3.427
104	3.427
105	3.427
106	3.427
107	3.428
108	3.428
109	3.428
110	3.428
111	3.428
112	3.428
113	3.429
114	3.429
115	3.429
116	3.429
117	3.429
118	3.430
119	3.430
120	3.430
121	3.430
122	3.430
123	3.430
124	3.430
125	3.431
126	3.431
127	3.431
128	3.431
129	3.431
130	3.431
131	3.432
132	3.432
133	3.432
134	3.432
135	3.432
136	3.432
137	3.432
138	3.432
139	3.433
140	3.433
141	3.433
142	3.433
143	3.433
144	3.433
145	3.433
146	3.433

147 3.434
148 3.434
149 3.434
150 3.434