

EIOPA RISK-FREE CURVE APRIL-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 April 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_20230430_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_20230430_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	0.111906	0.111906	0.111906	0.111906	0.111906	0.111906

5 rows × 106 columns

The country selected is:

In [193...]

```
country = "United States"
```

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              30.000000
Convergence      40.000000
UFR              3.450000
alpha             0.084332
CRA               0.000000
Name: United States_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 States_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      -77.129446
2       41.325318
3      -6.208917
4       4.173391
5      -0.535197
6       0.003301
7      -0.098569
8       0.002839
9       0.082178
10      0.387897
11     -0.021728
12     -0.018544
13     -0.016539
14     -0.014243
15      0.067105
16     -0.014400
```

```
17    -0.012066
18    -0.009444
19    -0.008991
20    -0.963786
21    0.043520
22    0.037358
23    0.035076
24    0.028303
25    0.037376
26    -0.010913
27    0.155836
28    -0.479335
29    1.880280
30    -1.125328
Name: United States 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 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 [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. Ex. M_Target = [1; 2; 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 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	0.053770
1	0.045732
2	0.041722
3	0.039563
4	0.038377
5	0.037700
6	0.037300
7	0.037074
8	0.036966
9	0.036947
10	0.036989
11	0.037061
12	0.037139
13	0.037207
14	0.037253

[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_20230430_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	0.05377
1	0.04573
2	0.04172
3	0.03956
4	0.03838

In [210...]

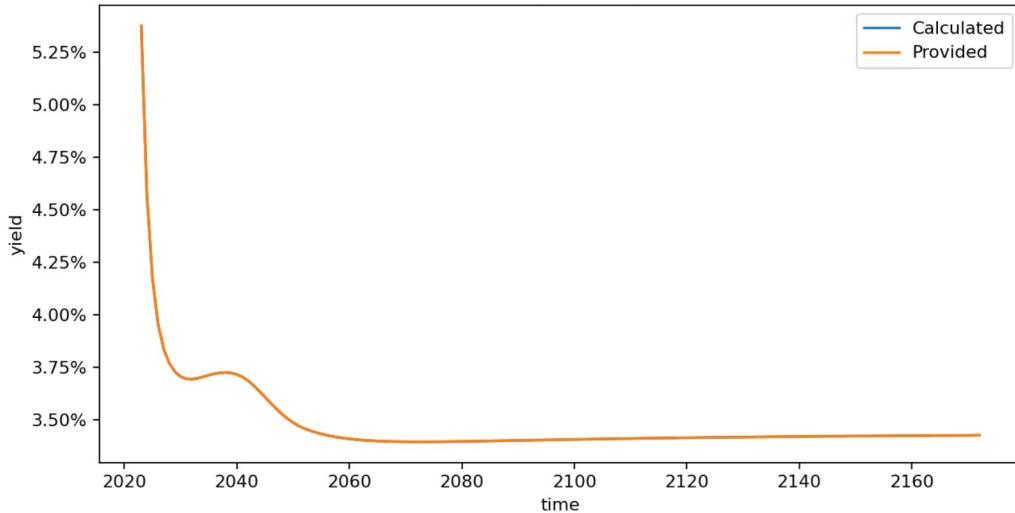
```
x_data_label = range(2023,2023+r_Target.shape[0],1)
```

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()
```

Recalculated & provided yield curve



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	0.000018
1	0.024886
2	0.024929
3	0.027222

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.

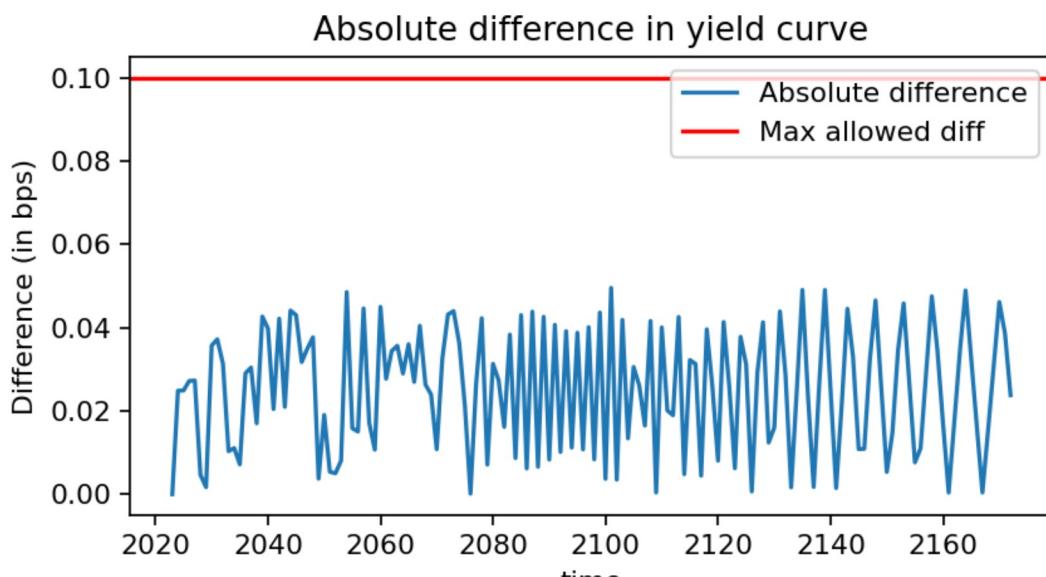
In [214...]

```
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 April 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_20230430_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230430_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	5.377
2	4.573
3	4.172
4	3.956
5	3.838
6	3.770
7	3.730
8	3.707
9	3.697
10	3.695
11	3.699
12	3.706
13	3.714
14	3.721
15	3.725
16	3.727
17	3.724
18	3.717
19	3.705
20	3.687
21	3.664
22	3.638
23	3.609
24	3.581
25	3.554
26	3.529
27	3.507
28	3.488

29	3.472
30	3.460
31	3.450
32	3.442
33	3.434
34	3.428
35	3.422
36	3.418
37	3.414
38	3.411
39	3.408
40	3.405
41	3.403
42	3.402
43	3.400
44	3.399
45	3.398
46	3.398
47	3.397
48	3.397
49	3.397
50	3.397
51	3.397
52	3.397
53	3.397
54	3.397
55	3.397
56	3.398
57	3.398
58	3.398
59	3.399
60	3.399
61	3.400
62	3.400
63	3.401
64	3.401
65	3.402
66	3.402
67	3.403
68	3.403
69	3.404
70	3.404
71	3.405
72	3.405
73	3.406
74	3.406
75	3.407
76	3.407
77	3.408
78	3.408
79	3.409
80	3.409
81	3.409
82	3.410
83	3.410
84	3.411
85	3.411
86	3.412
87	3.412

88	3.412
89	3.413
90	3.413
91	3.414
92	3.414
93	3.414
94	3.415
95	3.415
96	3.415
97	3.416
98	3.416
99	3.416
100	3.417
101	3.417
102	3.417
103	3.418
104	3.418
105	3.418
106	3.419
107	3.419
108	3.419
109	3.419
110	3.420
111	3.420
112	3.420
113	3.421
114	3.421
115	3.421
116	3.421
117	3.422
118	3.422
119	3.422
120	3.422
121	3.422
122	3.423
123	3.423
124	3.423
125	3.423
126	3.424
127	3.424
128	3.424
129	3.424
130	3.424
131	3.425
132	3.425
133	3.425
134	3.425
135	3.425
136	3.425
137	3.426
138	3.426
139	3.426
140	3.426
141	3.426
142	3.427
143	3.427
144	3.427
145	3.427
146	3.427

147 3.427
148 3.427
149 3.428
150 3.428