

EIOPA RISK-FREE CURVE MAY-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 May 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_20230531_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_20230531_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 [218...]

```
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 [219...]

```
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 [220...]

```
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 [221...]

```
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 [222...]

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

Parameter input

Parameters sheet

In [223...]

```
param_raw.head()
```

Out[223...]

	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	0.110654	0.110654	0.110654	0.110654	0.110654	0.110654

5 rows × 106 columns

The country selected is:

In [224...]

```
country = "Italy"
```

In [225...]

```
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 [226...]

```
extra_param
```

Out[226...]

```
Country
Coupon_freq      1.000000
LLP              20.000000
Convergence      40.000000
UFR              3.450000
alpha             0.110654
CRA               10.000000
Name: Italy_Values, dtype: float64
```

In [227...]

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

In [228...]

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

Maturity vector

Vector of maturities used in the calibration

In [229...]

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

Out[229...]

```
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: Italy_Maturities, dtype: float64
```

In [230...]

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

Calibration vector

Vector **Qb** provided as input

In [231...]

```
Qb
```

Out[231...]

```
Country
1      -10.723324
2       2.431863
3       0.247369
4       1.278669
5       0.285183
6      -0.597697
7       0.985473
8      -1.700809
9       4.274947
10      -8.375521
11      9.506108
12      -3.902860
13      -0.006909
14      -0.012108
15      -1.068289
16      0.020972
```

```
17      0.051873
18     -0.072923
19      0.386861
20      0.363031
Name: Italy Values, dtype: float64
```

```
In [232...]: curve_raw = pd.read_csv(selected_curves_file, sep=',', index_col=0)
```

```
In [233...]: curve_country = curve_raw.loc[:, country]
```

[Back to the top](#)

Calibration parameters and calibration vector provided by EIOPA

```
In [234...]: # 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 [235...]

```

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 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 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
    #
    # 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 [236...]

```

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 [237...]

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

Out[237...]

Recalculated rates

0	0.039290
1	0.035903
2	0.033418
3	0.031861
4	0.031004
5	0.030550
6	0.030314
7	0.030236
8	0.030242
9	0.030324
10	0.030334
11	0.030542
12	0.030693
13	0.030704
14	0.030572

[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_20230531_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 [238...]

```
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 [239...]

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

EIOPA curve provided

Yield curve provided by EIOPA

In [240...]

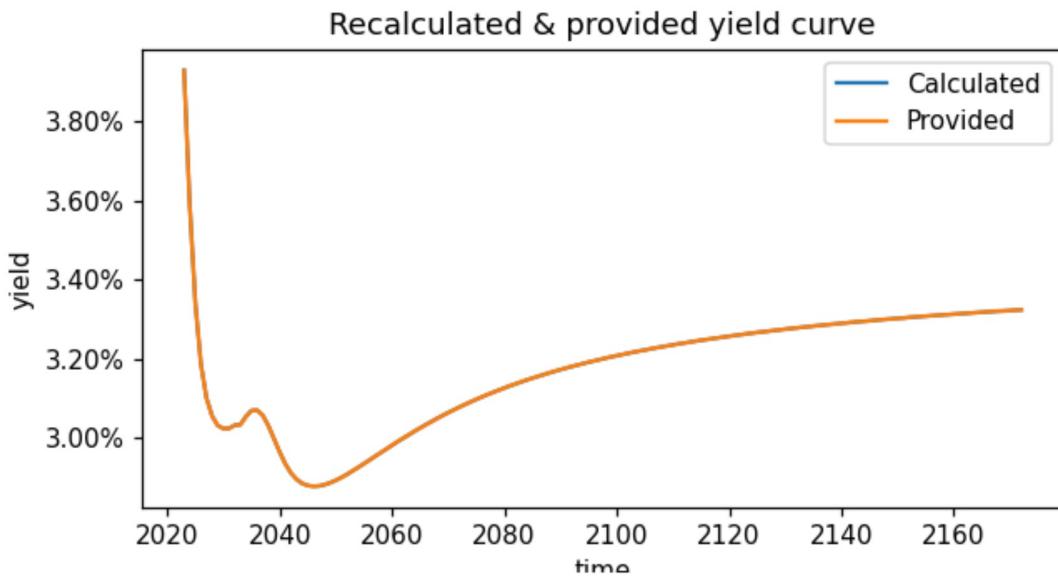
```
target_curve.head()
```

Out[240...]

	Given rates
0	0.03929
1	0.03590
2	0.03342
3	0.03186
4	0.03100

```
In [241...]:  
x_data_label = range(2023,2023+r_Target.shape[0],1)
```

```
In [242...]:  
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 [243...]:  
test_statistics_bdp = pd.DataFrame(abs(r_Target.values-target_curve.values)*10000, co
```

EIOPA curve comparison

Absolute difference in bps

```
In [244...]:  
test_statistics_bdp.head()
```

```
Out[244...]:  
Abs diff in bps
```

0	0.000001
1	0.034766
2	0.021549
3	0.005656

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.

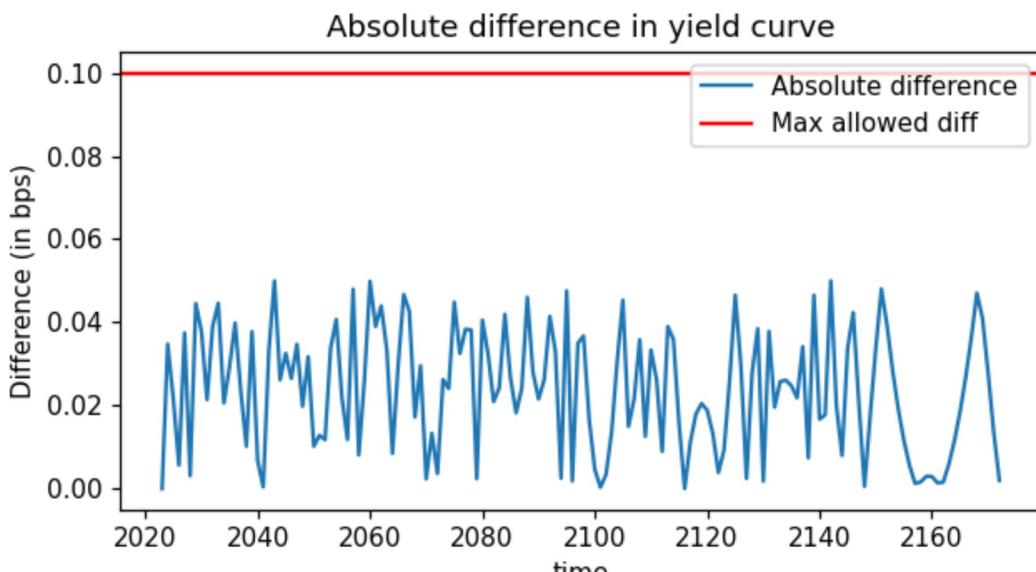
In [245...]

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

```
Test passed  
Test passed
```

In [246...]

```
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 May 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_20230531_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230531_Term_Structures.xlsx*.

In [247...]

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

Out[247...]

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [248...]

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

Out[248...]

Country	
1	3.929
2	3.590
3	3.342
4	3.186
5	3.100
6	3.055
7	3.031
8	3.024
9	3.024
10	3.032
11	3.033
12	3.054
13	3.069
14	3.070
15	3.057
16	3.031
17	2.998
18	2.965
19	2.935
20	2.912
21	2.896
22	2.885
23	2.880
24	2.878
25	2.879
26	2.882
27	2.887
28	2.893

29	2.900
30	2.908
31	2.917
32	2.926
33	2.935
34	2.944
35	2.954
36	2.963
37	2.972
38	2.982
39	2.991
40	3.000
41	3.008
42	3.017
43	3.025
44	3.033
45	3.041
46	3.049
47	3.057
48	3.064
49	3.071
50	3.078
51	3.085
52	3.091
53	3.098
54	3.104
55	3.110
56	3.115
57	3.121
58	3.126
59	3.132
60	3.137
61	3.142
62	3.147
63	3.151
64	3.156
65	3.160
66	3.165
67	3.169
68	3.173
69	3.177
70	3.181
71	3.184
72	3.188
73	3.192
74	3.195
75	3.198
76	3.202
77	3.205
78	3.208
79	3.211
80	3.214
81	3.217
82	3.220
83	3.222
84	3.225
85	3.228
86	3.230
87	3.233

88	3.235
89	3.238
90	3.240
91	3.242
92	3.245
93	3.247
94	3.249
95	3.251
96	3.253
97	3.255
98	3.257
99	3.259
100	3.261
101	3.263
102	3.265
103	3.267
104	3.268
105	3.270
106	3.272
107	3.273
108	3.275
109	3.277
110	3.278
111	3.280
112	3.281
113	3.283
114	3.284
115	3.286
116	3.287
117	3.288
118	3.290
119	3.291
120	3.293
121	3.294
122	3.295
123	3.296
124	3.298
125	3.299
126	3.300
127	3.301
128	3.302
129	3.303
130	3.305
131	3.306
132	3.307
133	3.308
134	3.309
135	3.310
136	3.311
137	3.312
138	3.313
139	3.314
140	3.315
141	3.316
142	3.317
143	3.318
144	3.319
145	3.320
146	3.321

147 3.321
148 3.322
149 3.323
150 3.324