

# EIOPA RISK-FREE CURVE AUGUST-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 August 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_20230831_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_20230831_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						
<b>Coupon_freq</b>	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
<b>LLP</b>	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000
<b>Convergence</b>	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000
<b>UFR</b>	3.450000	3.450000	3.450000	3.450000	3.450000	3.450000
<b>alpha</b>	0.108278	0.108278	0.108278	0.108278	0.108278	0.108278

5 rows × 106 columns

The country selected is:

In [193...]

```
country = "Italy"
```

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	20.000000
Convergence	40.000000
UFR	3.450000
alpha	0.108278
CRA	10.000000

Name: Italy\_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: Italy_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      -15.028763
2       8.624741
3      -6.270364
4       6.751033
5      -5.857230
6      10.755439
7     -17.471911
8      22.170048
9     -24.192902
10     24.559492
11    -19.539005
12      8.940833
13     -0.064700
14     -0.052024
15     -2.267938
16      0.025981
```

```
17      0.057874
18     -0.076210
19      0.417803
20      0.412044
Name: Italy 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, 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 [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	0.040841
1	0.037168
2	0.034816
3	0.033048
4	0.032129
5	0.031602
6	0.031455
7	0.031157
8	0.031290
9	0.031202
10	0.031447
11	0.031432
12	0.031473
13	0.031552
14	0.031534

---

[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\_20230831\_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
<b>0</b>	0.04084
<b>1</b>	0.03717
<b>2</b>	0.03481
<b>3</b>	0.03305
<b>4</b>	0.03213

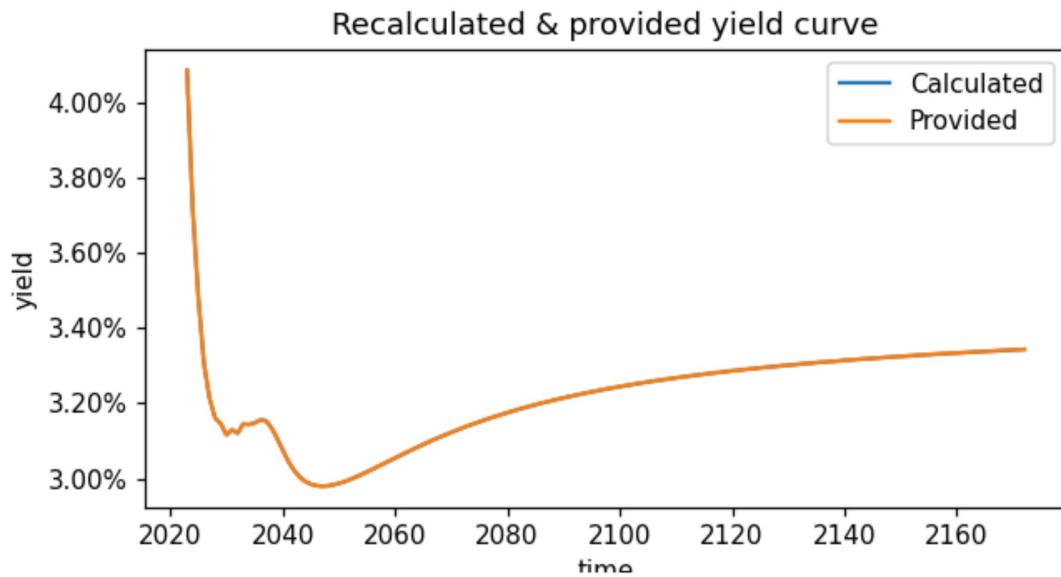
In [210...]

```
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, columns=['Abs diff in bps'])
```

### EIOPA curve comparison

Absolute difference in bps

In [213...]

```
test_statistics_bdp.head()
```

Out[213...]

	Abs diff in bps
0	0.012549
1	0.021165
2	0.056909
3	0.023036

**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.

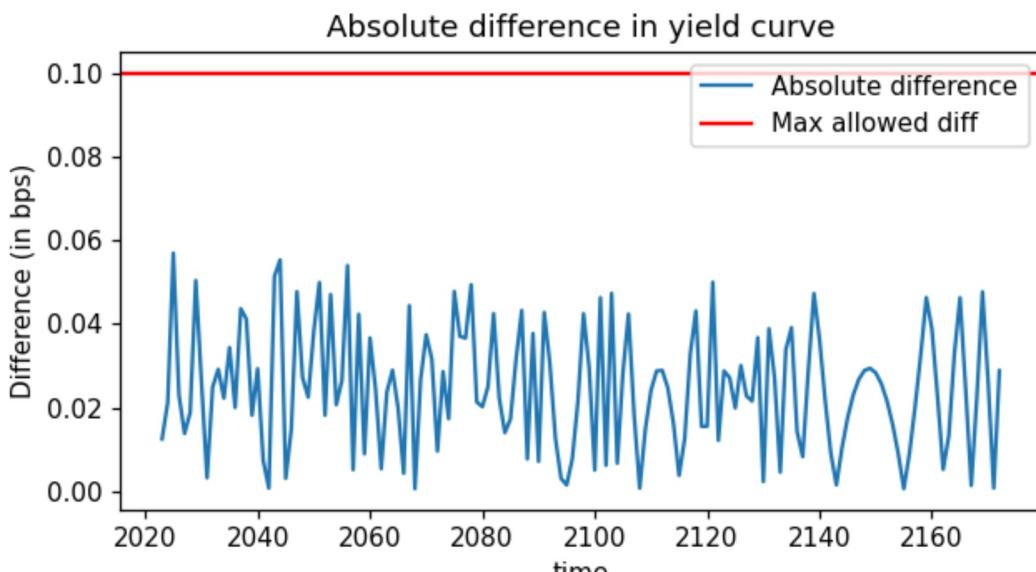
In [214...]

```
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 August 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\_20230831\_Qb\_SW.xlsx* and the parameters displayed in the file *EIOPA\_RFR\_20230831\_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	4.084
2	3.717
3	3.481
4	3.305
5	3.213
6	3.160
7	3.145
8	3.116
9	3.129
10	3.120
11	3.145
12	3.143
13	3.147
14	3.155
15	3.153
16	3.135
17	3.107
18	3.076
19	3.046
20	3.022
21	3.004
22	2.992
23	2.985
24	2.981
25	2.979
26	2.980
27	2.983
28	2.987

29	2.991
30	2.997
31	3.003
32	3.010
33	3.017
34	3.024
35	3.032
36	3.039
37	3.047
38	3.054
39	3.062
40	3.069
41	3.076
42	3.083
43	3.090
44	3.097
45	3.104
46	3.110
47	3.116
48	3.122
49	3.128
50	3.134
51	3.140
52	3.145
53	3.150
54	3.156
55	3.161
56	3.165
57	3.170
58	3.175
59	3.179
60	3.184
61	3.188
62	3.192
63	3.196
64	3.200
65	3.203
66	3.207
67	3.211
68	3.214
69	3.217
70	3.221
71	3.224
72	3.227
73	3.230
74	3.233
75	3.236
76	3.239
77	3.241
78	3.244
79	3.247
80	3.249
81	3.252
82	3.254
83	3.256
84	3.259
85	3.261
86	3.263
87	3.265

88	3.267
89	3.269
90	3.271
91	3.273
92	3.275
93	3.277
94	3.279
95	3.281
96	3.282
97	3.284
98	3.286
99	3.287
100	3.289
101	3.291
102	3.292
103	3.294
104	3.295
105	3.297
106	3.298
107	3.300
108	3.301
109	3.302
110	3.304
111	3.305
112	3.306
113	3.308
114	3.309
115	3.310
116	3.311
117	3.312
118	3.314
119	3.315
120	3.316
121	3.317
122	3.318
123	3.319
124	3.320
125	3.321
126	3.322
127	3.323
128	3.324
129	3.325
130	3.326
131	3.327
132	3.328
133	3.329
134	3.330
135	3.331
136	3.332
137	3.333
138	3.333
139	3.334
140	3.335
141	3.336
142	3.337
143	3.337
144	3.338
145	3.339
146	3.340

147 3.341  
148 3.341  
149 3.342  
150 3.343