

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 [571...]

```
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 [572...]

```
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 [573...]

```
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 [574...]

```
#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 [575...]

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

Parameter input

Parameters sheet

In [576...]

```
param_raw.head()
```

Out[576...]

	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	0.115699	0.115699	0.115699	0.115699	0.115699	0.115699

5 rows × 106 columns

The country selected is:

In [577...]

```
country = "Turkey"
```

In [578...]

```
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 [579...]

```
extra_param
```

Out[579...]

```
Country
Coupon_freq      0.000000
LLP              9.000000
Convergence      51.000000
UFR              5.500000
alpha             0.107605
CRA               10.000000
Name: Turkey_Values, dtype: float64
```

In [580...]

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

In [581...]

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

Maturity vector

Vector of maturities used in the calibration

In [582...]

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

Out[582...]

```
Country
1    1.0
2    2.0
3    3.0
4    4.0
5    5.0
6    6.0
7    8.0
8    9.0
Name: Turkey_Maturities, dtype: float64
```

In [583...]

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

Calibration vector

Vector **Qb** provided as input

In [584...]

```
Qb
```

Out[584...]

```
Country
1    17.835008
2   -18.554346
3     2.471782
4   -3.779119
5    2.589646
6    1.280136
7   -18.230335
8   16.651995
Name: Turkey_Values, dtype: float64
```

In [585...]

```
curve_raw = pd.read_csv(selected_curves_file, sep=',', index_col=0)
```

In [586...]

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

[Back to the top](#)

Calibration parameters and calibration vector provided

bv, EIOPA

In [587...]

```
# 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 to the target  
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 [588...]

```

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 [589...]

```

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 [590...]

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

Out[590...]

Recalculated rates

0	0.171136
1	0.176664
2	0.175775
3	0.171118
4	0.163782
5	0.154851
6	0.144534
7	0.132733
8	0.120417
9	0.110461
10	0.102749
11	0.096650
12	0.091741
13	0.087731
14	0.084413

[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 [591...]

```
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 [592...]

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

EIOPA curve provided

Yield curve provided by EIOPA

In [593...]

```
target_curve.head()
```

Out[593...]

	Given rates
0	0.17114
1	0.17666
2	0.17577
3	0.17112
4	0.16378

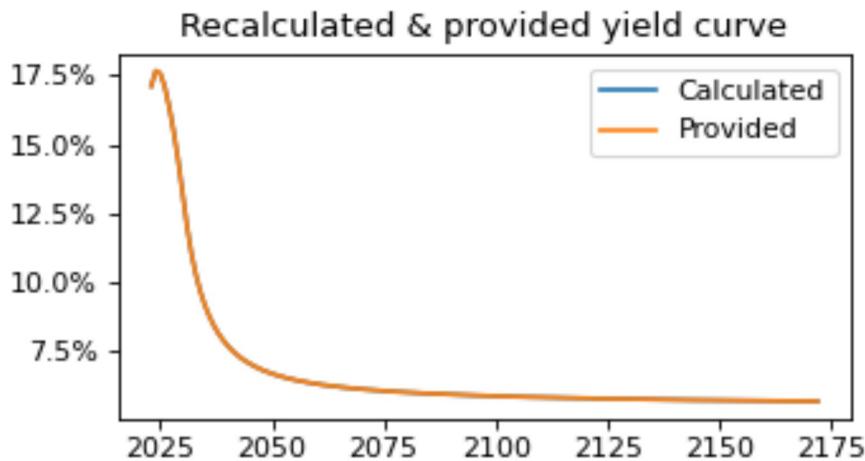
In [594...]

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

In [595...]

```
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 [596...]

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

EIOPA curve comparison

Absolute difference in bps

In [597...]

```
test_statistics_bdp.head()
```

Out[597...]

	Abs diff in bps
0	0.043456
1	0.040145
2	0.049046
3	0.023468
4	0.024991

0	0.043456
1	0.040145
2	0.049046
3	0.023468
4	0.024991

[Back to the top](#)

Test 1; Success criteria

The successful application of the success criteria marks the completion of the test.

In [598...]

```
result1 = SuccessTest(test_statistics_bdp.values, test_statistics_max_diff_in_bps, te
```

Test passed

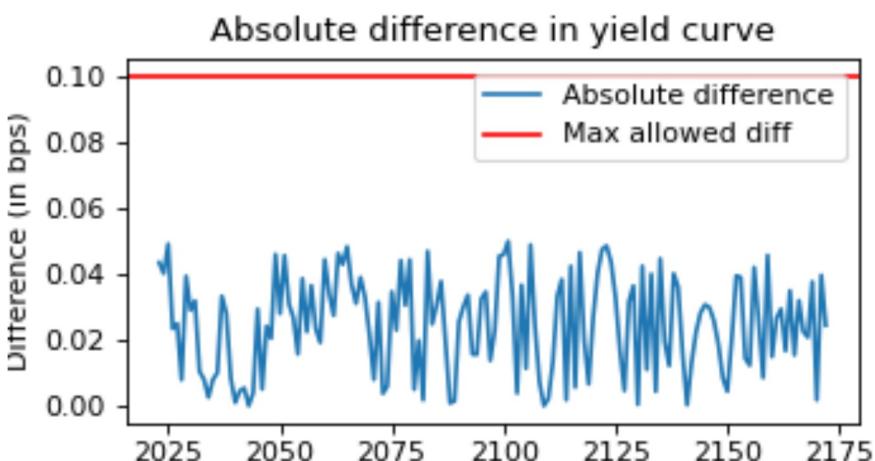
Test passed

In [599...]

```
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 [600...]

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

Out[600...]

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [601...]

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

Out[601...]

Country	
1	17.114
2	17.666
3	17.577
4	17.112
5	16.378
6	15.485
7	14.453
8	13.273
9	12.042
10	11.046
11	10.275
12	9.665
13	9.174
14	8.773
15	8.441
16	8.164
17	7.929
18	7.729
19	7.557
20	7.408
21	7.278
22	7.164
23	7.063
24	6.974
25	6.894
26	6.823
27	6.758
28	6.700
29	6.647
30	6.599
31	6.555
32	6.515
33	6.478
34	6.443
35	6.411
36	6.382
37	6.354

38	6.328
39	6.304
40	6.282
41	6.260
42	6.240
43	6.222
44	6.204
45	6.187
46	6.171
47	6.155
48	6.141
49	6.127
50	6.114
51	6.101
52	6.089
53	6.077
54	6.066
55	6.056
56	6.045
57	6.035
58	6.026
59	6.017
60	6.008
61	5.999
62	5.991
63	5.983
64	5.976
65	5.968
66	5.961
67	5.954
68	5.947
69	5.941
70	5.934
71	5.928
72	5.922
73	5.916
74	5.911
75	5.905
76	5.900
77	5.895
78	5.889
79	5.885
80	5.880
81	5.875
82	5.870
83	5.866
84	5.862
85	5.857
86	5.853
87	5.849
88	5.845
89	5.841
90	5.837
91	5.834
92	5.830
93	5.826
94	5.823
95	5.820
96	5.816

97	5.813
98	5.810
99	5.807
100	5.804
101	5.801
102	5.798
103	5.795
104	5.792
105	5.789
106	5.786
107	5.784
108	5.781
109	5.778
110	5.776
111	5.773
112	5.771
113	5.769
114	5.766
115	5.764
116	5.762
117	5.759
118	5.757
119	5.755
120	5.753
121	5.751
122	5.749
123	5.747
124	5.745
125	5.743
126	5.741
127	5.739
128	5.737
129	5.735
130	5.733
131	5.732
132	5.730
133	5.728
134	5.726
135	5.725
136	5.723
137	5.721
138	5.720
139	5.718
140	5.717
141	5.715
142	5.714
143	5.712
144	5.711
145	5.709
146	5.708
147	5.706
148	5.705
149	5.704
150	5.702

Name: Turkev. dtvbe: float64