EIOPA RISK-FREE CURVE JUNE-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

Note on Smith & Wilson algorithm

To replicate the calculations, this example uses a modified Smith&Wilson implementation (The original implementation is availible 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 June 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_20230630_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_20230630_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 [157…

```
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 [158…

```
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 [159…

```
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 [160…

```
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 [161…

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

Parameter input

Parameters sheet

In [162...
```
param_raw.head()
```

Out[162...

Country	Euro_Maturities	Euro_Values	Austria_Maturities	Austria_Values	Belgium_Maturities	Be
Coupon_freq	1.000000	1.000000	1.000000	1.000000	1.000000	
LLP	20.000000	20.000000	20.000000	20.000000	20.000000	
Convergence	40.000000	40.000000	40.000000	40.000000	40.000000	
UFR	3.450000	3.450000	3.450000	3.450000	3.450000	
alpha	0.111987	0.111987	0.111987	0.111987	0.111987	

5 rows × 106 columns

The country selected is:

In [163...
```
country = "Turkey"
```

In [164...
```
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 [165...
```
extra_param
```

Out[165...
```
Country
Coupon_freq     0.000000
LLP             9.000000
Convergence    51.000000
UFR             5.500000
alpha           0.165319
CRA            10.000000
Name: Turkey_Values, dtype: float64
```

In [166...
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [167…
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [168…
```
maturities_country.head(15)
```

Out[168…
```
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 [169…
```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector **Qb** provided as input

In [170…
```
Qb
```

Out[170…
```
Country
1    2.763976
2    1.871045
3   -0.692405
4   -0.523172
5   -0.844859
6   -0.179319
7   -0.136517
8   -0.156069
Name: Turkey_Values, dtype: float64
```

In [171…
```
curve_raw = pd.read_csv(selected_curves_file, sep=',',index_col=0)
```

In [172…
```
curve_country = curve_raw.loc[:,country]
```

Back to the top

Calibration parameters and calibration vector provided

by EIOPA

In [173…

```
# Maturity of observations:
M_Obs = np.transpose(np.array(maturities_country.values))

# Ultimate froward 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 t
alpha = extra_param.iloc[4]

# For which maturities do we want the SW algorithm to calculate the rates. In this ca
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 [174…

```
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 matu
#
# Arguments:
#    M_Target = k x 1 ndarray. Each element represents a bond maturity of interest. E
#    M_Obs =    n x 1 ndarray. Observed bond maturities used for calibrating the cal
#    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 o
#
#
# Returns:
#    k x 1 ndarray. Represents the targeted rates for a zero-coupon bond. Each rate E
#
# For more information see https://www.eiopa.europa.eu/sites/default/files/risk_free_

    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 o
    #
    # 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_f

        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)) - al

    H = SWHeart(M_Target, M_Obs, alpha) # Heart of the Wilson function from paragraph
    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 [175…

```
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 [176…
```
r_Target.head(15)
```

Out[176…

	Recalculated rates
0	0.128955
1	0.137800
2	0.148035
3	0.157283
4	0.164767
5	0.170364
6	0.174457
7	0.177373
8	0.179220
9	0.180106
10	0.180217
11	0.179713
12	0.178709
13	0.177292
14	0.175536

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_20230630_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 [177…
```
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 [178...]
```
target_curve = pd.DataFrame(target_curve,columns=['Given rates'])
```

EIOPA curve provided

Yield curve provided by EIOPA

In [179...]
```
target_curve.head()
```

Out[179...]

	Given rates
0	0.12896
1	0.13780
2	0.14803
3	0.15728
4	0.16477

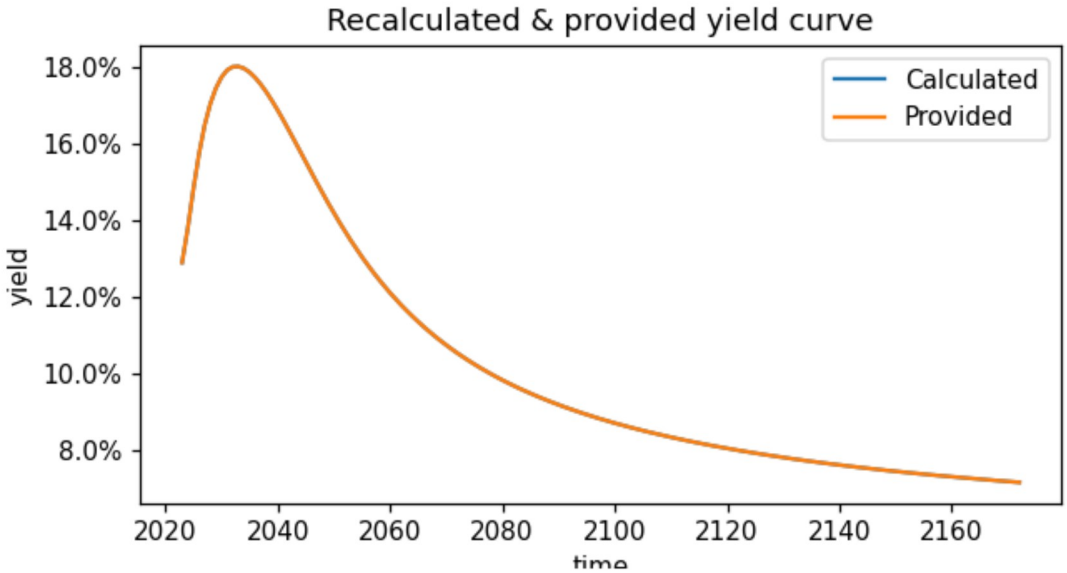
In [180…

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

In [181…

```
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 [182…

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

EIOPA curve comparison

Absolute difference in bps

In [183…

```
test_statistics_bdp.head()
```

Out[183…

	Abs diff in bps
0	0.047840
1	0.004590
2	0.048918
3	0.028604

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 [184...

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

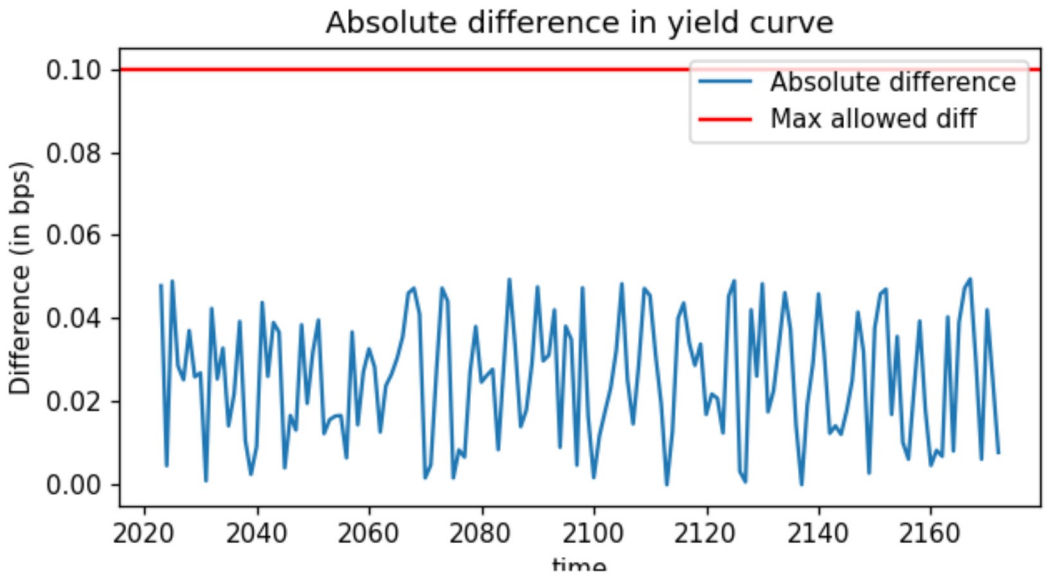
```
Test passed
Test passed
```

In [185...

```
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='

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 June 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_20230630_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230630_Term_Structures.xlsx*.

In [186...

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

Out[186...

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [187...

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

Out[187...

```
Country
1      12.896
2      13.780
3      14.803
4      15.728
5      16.477
6      17.036
7      17.446
8      17.737
9      17.922
10     18.011
11     18.022
12     17.971
13     17.871
14     17.729
15     17.554
16     17.350
17     17.124
18     16.880
19     16.623
20     16.355
21     16.082
22     15.805
23     15.527
24     15.251
25     14.978
26     14.710
27     14.449
28     14.194
```

29	13.948
30	13.709
31	13.479
32	13.258
33	13.045
34	12.841
35	12.645
36	12.458
37	12.278
38	12.106
39	11.942
40	11.784
41	11.633
42	11.489
43	11.350
44	11.218
45	11.090
46	10.968
47	10.851
48	10.739
49	10.631
50	10.527
51	10.427
52	10.331
53	10.239
54	10.150
55	10.064
56	9.981
57	9.901
58	9.824
59	9.750
60	9.678
61	9.608
62	9.541
63	9.475
64	9.412
65	9.351
66	9.292
67	9.234
68	9.178
69	9.124
70	9.072
71	9.021
72	8.971
73	8.923
74	8.876
75	8.830
76	8.785
77	8.742
78	8.700
79	8.659
80	8.619
81	8.580
82	8.542
83	8.505
84	8.468
85	8.433
86	8.398
87	8.365

88	8.332
89	8.299
90	8.268
91	8.237
92	8.207
93	8.177
94	8.149
95	8.120
96	8.093
97	8.066
98	8.039
99	8.013
100	7.988
101	7.963
102	7.938
103	7.915
104	7.891
105	7.868
106	7.845
107	7.823
108	7.802
109	7.780
110	7.759
111	7.739
112	7.718
113	7.699
114	7.679
115	7.660
116	7.641
117	7.623
118	7.605
119	7.587
120	7.569
121	7.552
122	7.535
123	7.518
124	7.502
125	7.486
126	7.470
127	7.454
128	7.439
129	7.423
130	7.409
131	7.394
132	7.379
133	7.365
134	7.351
135	7.337
136	7.324
137	7.310
138	7.297
139	7.284
140	7.271
141	7.258
142	7.246
143	7.234
144	7.221
145	7.210
146	7.198

```
147    7.186
148    7.175
149    7.163
150    7.152
```