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

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 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 [94]:
```
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 [95]:
```
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 [96]:
```
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 [97]:
```
#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 [98]:
```
param_raw = pd.read_csv(selected_param_file, sep=',',index_col=0)
```

Parameter input

Parameters sheet

In [99]:
```
param_raw.head()
```

Out[99]:

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	0.115699	0.115699	0.115699	0.115699	0.115699	

5 rows × 106 columns

The country selected is:

In [100…
```
country = "United Kingdom"
```

In [101…
```
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 [102…
```
extra_param
```

Out[102…
```
Country
Coupon_freq     1.00000
LLP            50.00000
Convergence    40.00000
UFR             3.45000
alpha           0.10184
CRA             0.00000
Name: United Kingdom_Values, dtype: float64
```

In [103…
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [104...
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [105...
```
maturities_country.head(15)
```

Out[105...
```
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 Kingdom_Maturities, dtype: float64
```

In [106...
```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector **Qb** provided as input

In [107...
```
Qb
```

Out[107...
```
Country
1    -12.858461
2      3.031613
3      2.252501
4     -1.332939
5      0.194861
6      0.260170
7     -0.398310
8      0.027357
9      1.640398
10    -0.640048
11    -0.009481
12    -0.116784
13    -0.005051
14    -0.004883
15    -0.170723
16     0.001281
```

```
17     0.001239
18     0.001197
19     0.001158
20     0.128748
21    -0.003364
22    -0.003251
23    -0.003143
24    -0.003038
25    -0.002937
26    -0.002839
27    -0.002744
28    -0.002653
29    -0.002564
30    -0.328044
31     0.008614
32     0.008327
33     0.008049
34     0.007781
35     0.007521
36     0.007271
37     0.007028
38     0.006794
39     0.006567
40     0.006348
41     0.006136
42     0.005932
43     0.005734
44     0.005543
45     0.005358
46     0.005179
47     0.005007
48     0.004840
49     0.004678
50     0.143601
Name: United Kingdom Values, dtype: float64
```

In [108…

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

In [109…

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

Back to the top

Calibration parameters and calibration vector provided by EIOPA

In [110…

```
# 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 [111…

```
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 [112…

```
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 [113…
```
r_Target.head(15)
```

Out[113…

	Recalculated rates
0	0.048295
1	0.045096
2	0.042685
3	0.041000
4	0.039627
5	0.038517
6	0.037653
7	0.037008
8	0.036593
9	0.036390
10	0.036287
11	0.036226
12	0.036179
13	0.036130
14	0.036072

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 [114…
```
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 [115…]
```
target_curve = pd.DataFrame(target_curve,columns=['Given rates'])
```

EIOPA curve provided

Yield curve provided by EIOPA

In [116…]
```
target_curve.head()
```

Out[116…]

	Given rates
0	0.04830
1	0.04510
2	0.04268
3	0.04100
4	0.03963

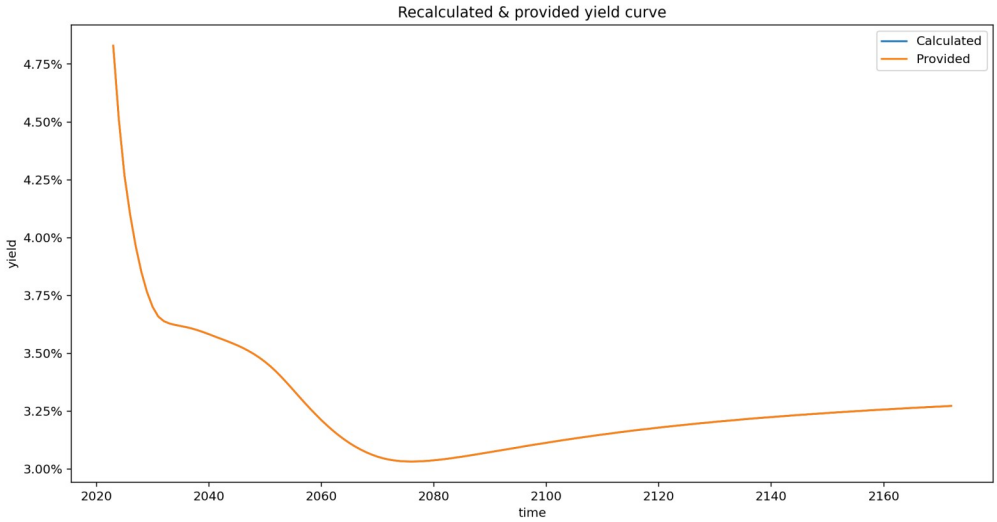
In [117…

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

In [118…

```
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 [119…

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

EIOPA curve comparison

Absolute difference in bps

In [120…

```
test_statistics_bdp.head()
```

Out[120…

	Abs diff in bps
0	0.049960
1	0.044400
2	0.046265
3	0.000088

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 [121…

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

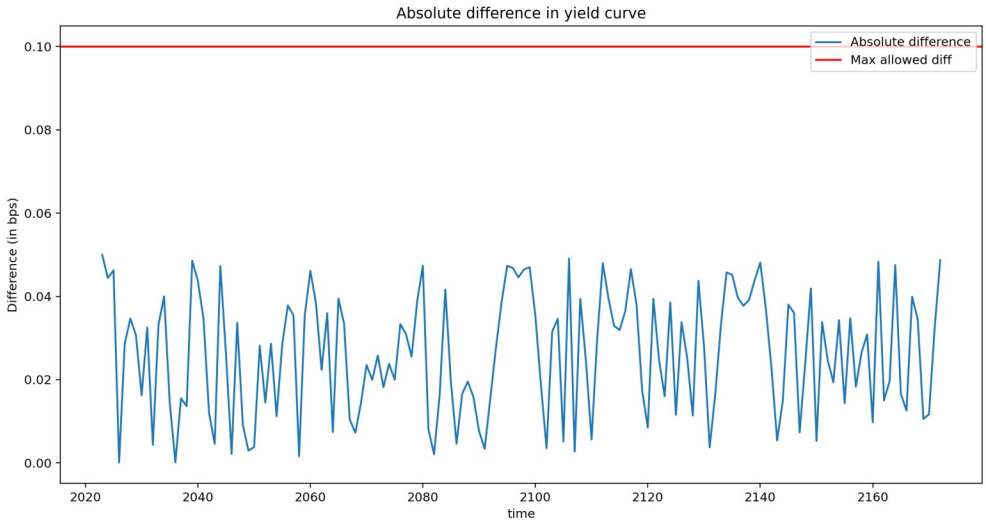
```
Test passed
Test passed
```

In [122…

```
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 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 [123...

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

Out[123...

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [124...

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

Out[124...

```
Country
1      4.830
2      4.510
3      4.268
4      4.100
5      3.963
6      3.852
7      3.765
8      3.701
9      3.659
10     3.639
11     3.629
12     3.623
13     3.618
14     3.613
15     3.607
16     3.600
17     3.592
18     3.583
19     3.573
20     3.564
21     3.555
22     3.545
23     3.535
24     3.524
25     3.511
26     3.497
27     3.481
28     3.463
```

29	3.443
30	3.420
31	3.395
32	3.369
33	3.342
34	3.314
35	3.287
36	3.261
37	3.236
38	3.211
39	3.189
40	3.167
41	3.147
42	3.129
43	3.112
44	3.097
45	3.084
46	3.072
47	3.062
48	3.053
49	3.046
50	3.041
51	3.037
52	3.034
53	3.033
54	3.032
55	3.033
56	3.034
57	3.035
58	3.038
59	3.040
60	3.043
61	3.046
62	3.050
63	3.053
64	3.057
65	3.061
66	3.065
67	3.069
68	3.073
69	3.077
70	3.081
71	3.085
72	3.089
73	3.093
74	3.098
75	3.102
76	3.106
77	3.109
78	3.113
79	3.117
80	3.121
81	3.125
82	3.128
83	3.132
84	3.135
85	3.139
86	3.142
87	3.146

88	3.149
89	3.152
90	3.155
91	3.159
92	3.162
93	3.165
94	3.168
95	3.171
96	3.173
97	3.176
98	3.179
99	3.182
100	3.184
101	3.187
102	3.189
103	3.192
104	3.194
105	3.197
106	3.199
107	3.201
108	3.204
109	3.206
110	3.208
111	3.210
112	3.212
113	3.215
114	3.217
115	3.219
116	3.221
117	3.223
118	3.224
119	3.226
120	3.228
121	3.230
122	3.232
123	3.234
124	3.235
125	3.237
126	3.239
127	3.240
128	3.242
129	3.244
130	3.245
131	3.247
132	3.248
133	3.250
134	3.251
135	3.253
136	3.254
137	3.256
138	3.257
139	3.258
140	3.260
141	3.261
142	3.263
143	3.264
144	3.265
145	3.266
146	3.268

```
147    3.269
148    3.270
149    3.271
150    3.273
```