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 [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	0.116339	0.116339	0.116339	0.116339	0.116339	

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     1.000000
LLP            50.000000
Convergence    40.000000
UFR             3.450000
alpha           0.101119
CRA             0.000000
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      9.113511
2    -16.346058
3      5.864136
4     -1.822881
5      2.169531
6     -2.349000
7      2.477135
8     -0.008765
9     -1.426631
10     1.826740
11    -0.023789
12    -0.395209
13    -0.006935
14    -0.006704
15    -0.213447
16     0.002030
```

```
17     0.001962
18     0.001897
19     0.001833
20     0.151788
21    -0.004111
22    -0.003974
23    -0.003841
24    -0.003713
25    -0.003589
26    -0.003469
27    -0.003354
28    -0.003242
29    -0.003134
30    -0.308585
31     0.008368
32     0.008089
33     0.007819
34     0.007558
35     0.007306
36     0.007063
37     0.006827
38     0.006599
39     0.006379
40     0.006167
41     0.005961
42     0.005762
43     0.005570
44     0.005384
45     0.005205
46     0.005031
47     0.004863
48     0.004701
49     0.004544
50     0.129527
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	0.060620
1	0.059601
2	0.056296
3	0.053089
4	0.050279
5	0.047951
6	0.046014
7	0.044539
8	0.043399
9	0.042502
10	0.041854
11	0.041368
12	0.040963
13	0.040606
14	0.040274

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 [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	0.06062
1	0.05960
2	0.05630
3	0.05309
4	0.05028

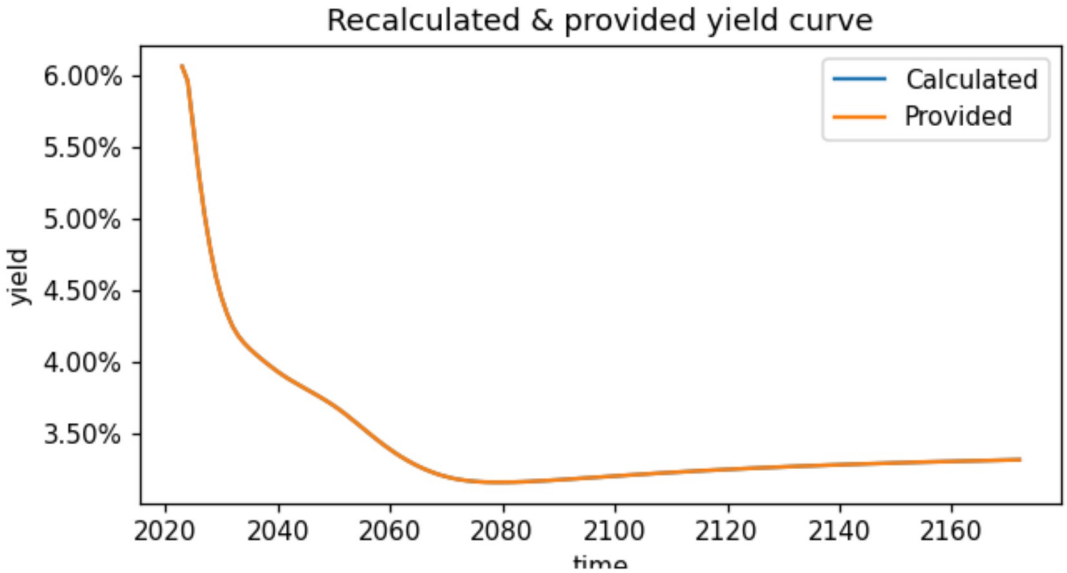
In [117…

```
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	0.000004
1	0.005115
2	0.041798
3	0.013632

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
```

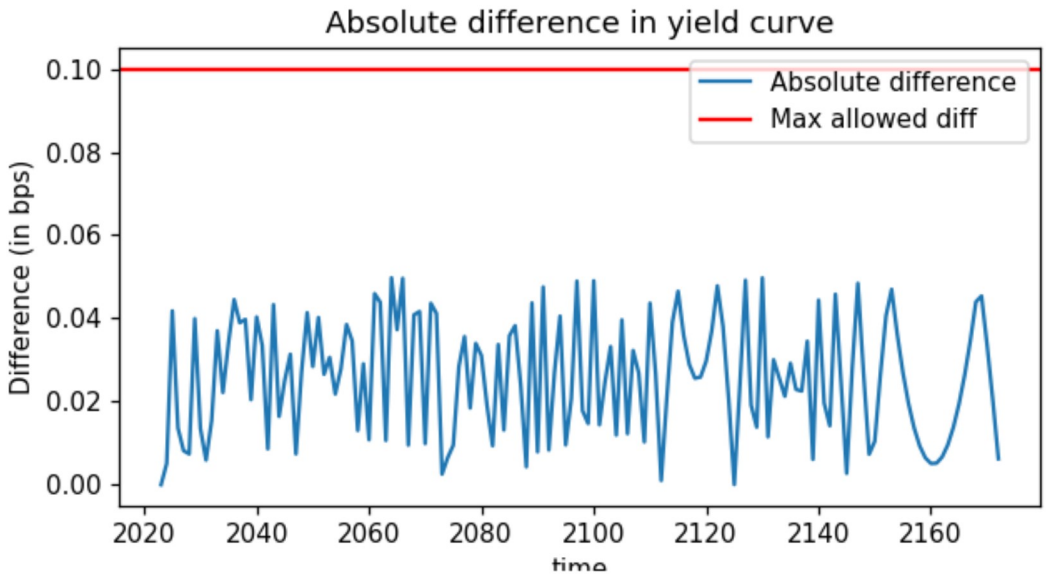
```
Test passed
Test passed
```

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 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 [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      6.062
2      5.960
3      5.630
4      5.309
5      5.028
6      4.795
7      4.601
8      4.454
9      4.340
10     4.250
11     4.185
12     4.137
13     4.096
14     4.061
15     4.027
16     3.996
17     3.965
18     3.936
19     3.909
20     3.884
21     3.861
22     3.838
23     3.816
24     3.794
25     3.772
26     3.749
27     3.724
28     3.699
```

29	3.671
30	3.642
31	3.611
32	3.579
33	3.547
34	3.514
35	3.483
36	3.452
37	3.422
38	3.394
39	3.367
40	3.342
41	3.319
42	3.297
43	3.278
44	3.259
45	3.243
46	3.228
47	3.215
48	3.204
49	3.194
50	3.186
51	3.180
52	3.175
53	3.171
54	3.168
55	3.166
56	3.165
57	3.165
58	3.165
59	3.165
60	3.166
61	3.167
62	3.169
63	3.171
64	3.173
65	3.175
66	3.177
67	3.179
68	3.182
69	3.184
70	3.187
71	3.190
72	3.192
73	3.195
74	3.198
75	3.200
76	3.203
77	3.206
78	3.209
79	3.211
80	3.214
81	3.216
82	3.219
83	3.221
84	3.224
85	3.226
86	3.229
87	3.231

88	3.233
89	3.236
90	3.238
91	3.240
92	3.242
93	3.245
94	3.247
95	3.249
96	3.251
97	3.253
98	3.255
99	3.257
100	3.259
101	3.260
102	3.262
103	3.264
104	3.266
105	3.267
106	3.269
107	3.271
108	3.273
109	3.274
110	3.276
111	3.277
112	3.279
113	3.280
114	3.282
115	3.283
116	3.285
117	3.286
118	3.287
119	3.289
120	3.290
121	3.291
122	3.293
123	3.294
124	3.295
125	3.297
126	3.298
127	3.299
128	3.300
129	3.301
130	3.302
131	3.304
132	3.305
133	3.306
134	3.307
135	3.308
136	3.309
137	3.310
138	3.311
139	3.312
140	3.313
141	3.314
142	3.315
143	3.316
144	3.317
145	3.318
146	3.319

```
147    3.319
148    3.320
149    3.321
150    3.322
```