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 [32]:
```
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 [33]:

```
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 [34]:

```
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 [35]:

```
#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 [36]:

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

Parameter input

Parameters sheet

In [37]:
```
param_raw.head()
```

Out[37]:

	Euro_Maturities	Euro_Values	Austria_Maturities	Austria_Values	Belgium_Maturities	Be
Country						
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 [38]:
```
country = "Italy"
```

In [39]:
```
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 [40]:
```
extra_param
```

Out[40]:
```
Country
Coupon_freq     1.000000
LLP            20.000000
Convergence    40.000000
UFR             3.450000
alpha           0.115699
CRA            10.000000
Name: Italy_Values, dtype: float64
```

In [41]:
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [42]:
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [43]:
```
maturities_country.head(15)
```

Out[43]:
```
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 [44]:
```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector **Qb** provided as input

In [45]:
```
Qb
```

Out[45]:
```
Country
1    -8.096518
2     0.461906
3     1.746009
4     0.966140
5    -0.432827
6    -0.356785
7     0.653077
8    -0.350523
9    -0.365608
10    2.512354
11   -3.619813
12    2.495825
13   -0.024658
14   -0.023836
15   -1.577265
16    0.021297
```

```
17    0.020587
18    0.019900
19    0.019236
20    0.689163
Name: Italy Values, dtype: float64
```

In [46]:
```
curve_raw = pd.read_csv(selected_curves_file, sep=',',index_col=0)
```

In [47]:
```
curve_country = curve_raw.loc[:,country]
```

Back to the top

Calibration parameters and calibration vector provided by EIOPA

In [48]:
```
# 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 [49]:

```
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 matur
#
# 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 a
#
#
# Returns:
#    k x 1 ndarray. Represents the targeted rates for a zero-coupon bond. Each rate b
#
# 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 a
    #
    # 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 [50]:

```
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 [51]:
```
r_Target.head(15)
```

Out[51]:

	Recalculated rates
0	0.036730
1	0.033619
2	0.031276
3	0.029977
4	0.029317
5	0.028934
6	0.028716
7	0.028646
8	0.028661
9	0.028752
10	0.028899
11	0.028964
12	0.029032
13	0.029051
14	0.028947

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 [52]:
```
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 [53]:

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

EIOPA curve provided

Yield curve provided by EIOPA

In [54]:

```
target_curve.head()
```

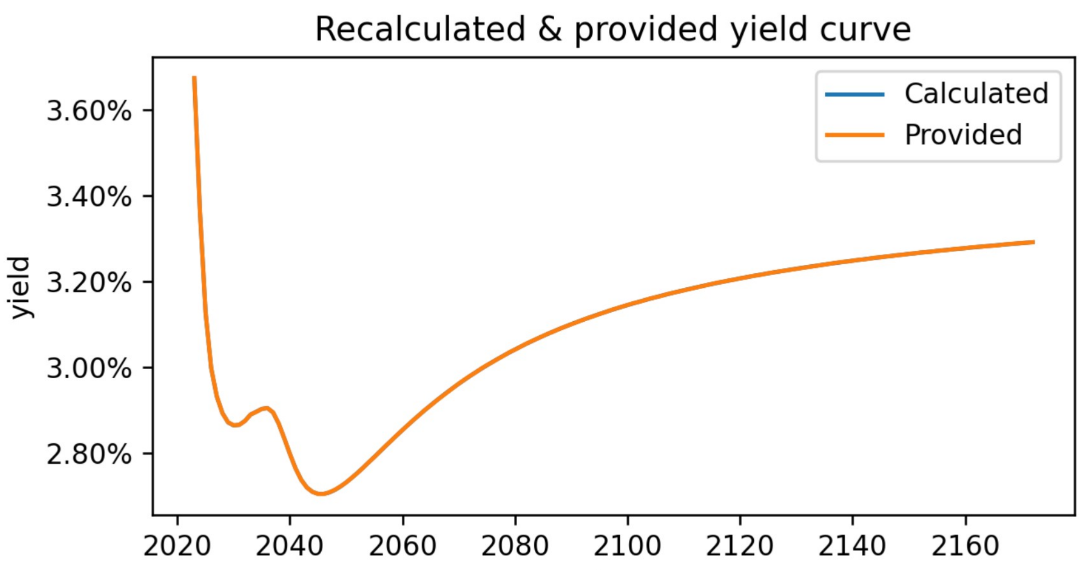
Out[54]:

	Given rates
0	0.03673
1	0.03362
2	0.03128
3	0.02998
4	0.02932

In [55]:
```
x_data_label = range(2023,2023+r_Target.shape[0],1)
```

In [56]:
```
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 [57]:
```
test_statistics_bdp = pd.DataFrame(abs(r_Target.values-target_curve.values)*10000, co
```

EIOPA curve comparison

Absolute difference in bps

In [58]:
```
test_statistics_bdp.head()
```

Out[58]:

	Abs diff in bps
0	0.001081
1	0.014663
2	0.041840
3	0.030074

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 [59]:

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

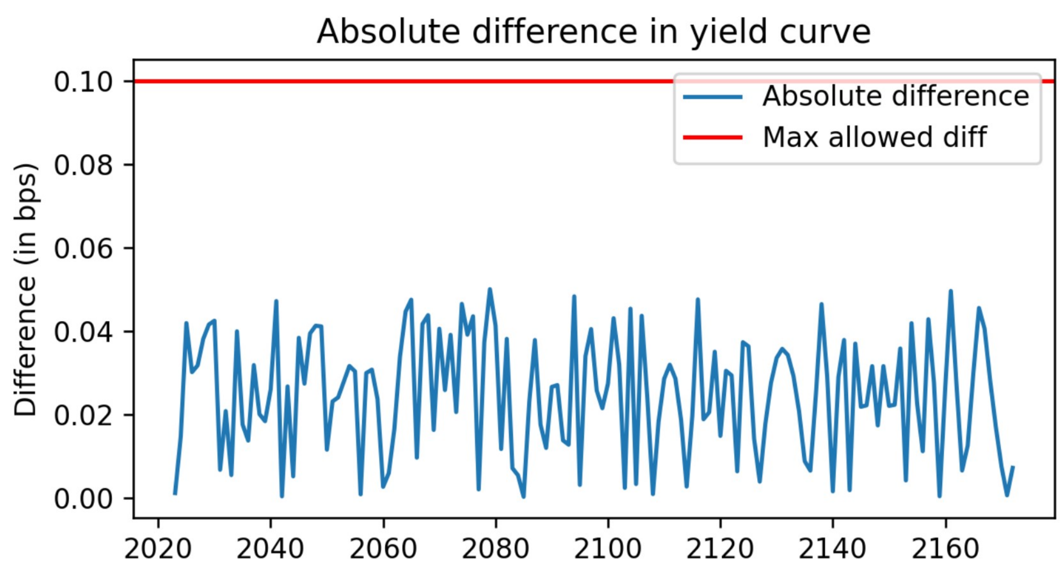
```
Test passed
Test passed
```

In [60]:

```
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 [61]:

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

Out[61]:

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [62]:

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

Out[62]:

```
Country
1       3.673
2       3.362
3       3.128
4       2.998
5       2.932
6       2.893
7       2.872
8       2.865
9       2.866
10      2.875
11      2.890
12      2.896
13      2.903
14      2.905
15      2.895
16      2.869
17      2.834
18      2.798
19      2.764
20      2.738
21      2.720
22      2.710
23      2.705
24      2.705
25      2.709
26      2.715
27      2.723
28      2.732
```

29	2.743
30	2.754
31	2.766
32	2.779
33	2.791
34	2.804
35	2.817
36	2.829
37	2.842
38	2.854
39	2.866
40	2.878
41	2.889
42	2.901
43	2.911
44	2.922
45	2.932
46	2.942
47	2.952
48	2.962
49	2.971
50	2.980
51	2.988
52	2.997
53	3.005
54	3.012
55	3.020
56	3.027
57	3.035
58	3.041
59	3.048
60	3.055
61	3.061
62	3.067
63	3.073
64	3.079
65	3.084
66	3.090
67	3.095
68	3.100
69	3.105
70	3.110
71	3.115
72	3.119
73	3.124
74	3.128
75	3.133
76	3.137
77	3.141
78	3.145
79	3.149
80	3.152
81	3.156
82	3.160
83	3.163
84	3.166
85	3.170
86	3.173
87	3.176

```
88     3.179
89     3.182
90     3.185
91     3.188
92     3.191
93     3.194
94     3.197
95     3.199
96     3.202
97     3.204
98     3.207
99     3.209
100    3.212
101    3.214
102    3.216
103    3.219
104    3.221
105    3.223
106    3.225
107    3.227
108    3.229
109    3.231
110    3.233
111    3.235
112    3.237
113    3.239
114    3.241
115    3.243
116    3.245
117    3.246
118    3.248
119    3.250
120    3.251
121    3.253
122    3.255
123    3.256
124    3.258
125    3.259
126    3.261
127    3.262
128    3.264
129    3.265
130    3.267
131    3.268
132    3.269
133    3.271
134    3.272
135    3.273
136    3.275
137    3.276
138    3.277
139    3.279
140    3.280
141    3.281
142    3.282
143    3.283
144    3.284
145    3.286
146    3.287
```

```
147    3.288
148    3.289
149    3.290
150    3.291
```