EIOPA RISK-FREE CURVE AUGUST-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 August 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_20230831_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_20230831_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	1.00000	1.00000	1.00000	1.00000	1.00000	
LLP	20.00000	20.00000	20.00000	20.00000	20.00000	
Convergence	40.00000	40.00000	40.00000	40.00000	40.00000	
UFR	3.45000	3.45000	3.45000	3.45000	3.45000	
alpha	0.11312	0.11312	0.11312	0.11312	0.11312	

5 rows × 106 columns

The country selected is:

In [100...
```
country = "Turkey"
```

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     0.000000
LLP             9.000000
Convergence    51.000000
UFR             5.500000
alpha           0.164348
CRA            10.000000
Name: Turkey_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    8.0
8    9.0
Name: Turkey_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   -12.849459
2     7.480230
3    -2.619455
4     0.824166
5    -0.093618
6    -0.340462
7     0.745429
8    -0.724825
Name: Turkey_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	0.197479
1	0.192255
2	0.191074
3	0.190492
4	0.190274
5	0.190172
6	0.189970
7	0.189868
8	0.189912
9	0.189477
10	0.188500
11	0.187084
12	0.185307
13	0.183230
14	0.180905

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_20230831_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	0.19748
1	0.19225
2	0.19107
3	0.19049
4	0.19027

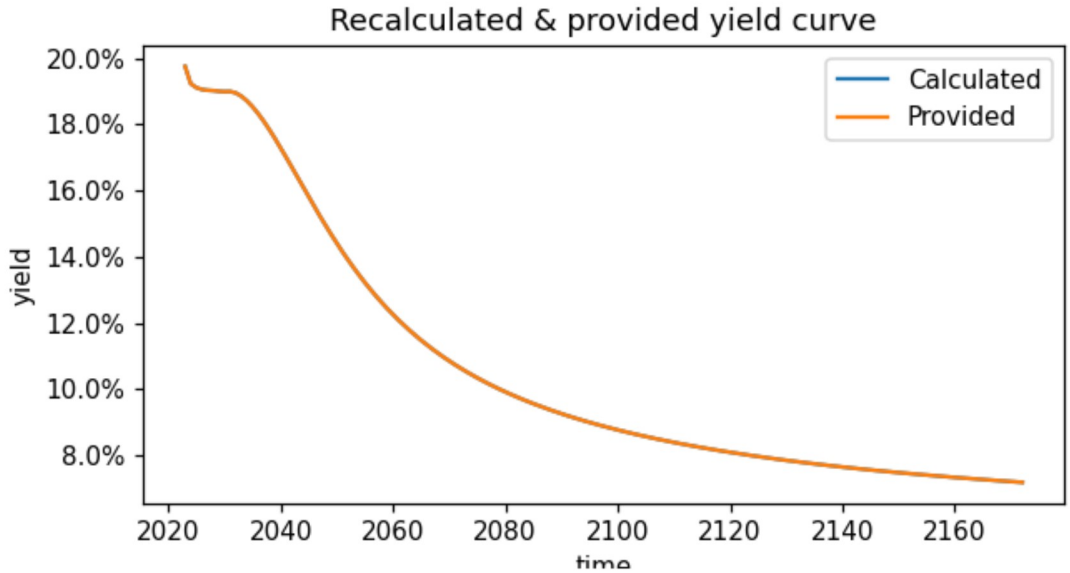
In [117…

```
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	0.005979
1	0.047016
2	0.037848
3	0.024414

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
```

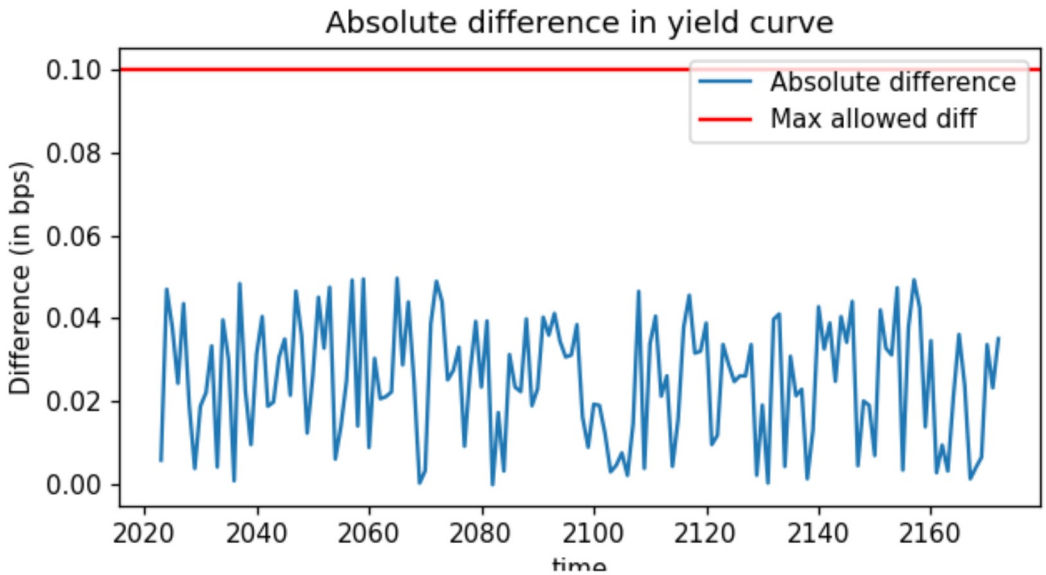
```
Test passed
Test passed
```

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 August 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_20230831_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230831_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      19.748
2      19.225
3      19.107
4      19.049
5      19.027
6      19.017
7      18.997
8      18.987
9      18.991
10     18.948
11     18.850
12     18.708
13     18.531
14     18.323
15     18.091
16     17.838
17     17.569
18     17.288
19     16.998
20     16.702
21     16.403
22     16.104
23     15.807
24     15.514
25     15.226
26     14.945
27     14.672
28     14.407
```

29	14.150
30	13.903
31	13.666
32	13.437
33	13.218
34	13.008
35	12.807
36	12.614
37	12.429
38	12.253
39	12.084
40	11.923
41	11.768
42	11.620
43	11.478
44	11.343
45	11.213
46	11.088
47	10.968
48	10.853
49	10.743
50	10.636
51	10.534
52	10.436
53	10.342
54	10.251
55	10.163
56	10.078
57	9.997
58	9.918
59	9.842
60	9.768
61	9.697
62	9.628
63	9.561
64	9.497
65	9.434
66	9.374
67	9.315
68	9.258
69	9.202
70	9.149
71	9.096
72	9.046
73	8.996
74	8.948
75	8.902
76	8.856
77	8.812
78	8.769
79	8.727
80	8.686
81	8.646
82	8.607
83	8.569
84	8.532
85	8.496
86	8.461
87	8.426

88	8.392
89	8.359
90	8.327
91	8.296
92	8.265
93	8.235
94	8.205
95	8.177
96	8.148
97	8.121
98	8.094
99	8.067
100	8.041
101	8.016
102	7.991
103	7.966
104	7.942
105	7.919
106	7.896
107	7.873
108	7.851
109	7.829
110	7.808
111	7.787
112	7.766
113	7.746
114	7.726
115	7.706
116	7.687
117	7.668
118	7.650
119	7.631
120	7.613
121	7.596
122	7.578
123	7.561
124	7.545
125	7.528
126	7.512
127	7.496
128	7.480
129	7.465
130	7.449
131	7.434
132	7.420
133	7.405
134	7.391
135	7.376
136	7.363
137	7.349
138	7.335
139	7.322
140	7.309
141	7.296
142	7.283
143	7.271
144	7.258
145	7.246
146	7.234

```
147     7.222
148     7.210
149     7.199
150     7.187
```