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

1. Note on Smith & Wilson algorithm
2. Data requirements
3. Success criteria
4. External dependencies
5. Calibration parameters and calibration vector provided by EIOPA
6. Smith & Wilson calculation functions
7. Generation of the risk-free curve
8. Test 1; Comparison test
9. Test 1; Success criteria
10. Test 1; Comparison test
11. Conclusion

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 [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]:

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 [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     1.00000
LLP            20.00000
Convergence    40.00000
UFR             3.45000
alpha           0.11312
CRA            10.00000
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    -13.199240
2      7.574708
3     -5.549199
4      5.970534
5     -5.205231
6      9.582742
7    -15.583880
8     19.799045
9    -21.638632
10    22.001530
11   -17.514407
12     8.008726
13    -0.039665
14    -0.038342
15    -2.052936
16     0.021852
```

```
17     0.021123
18     0.020419
19     0.019738
20     0.687608
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 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 [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	0.038840
1	0.035167
2	0.032814
3	0.031046
4	0.030127
5	0.029601
6	0.029454
7	0.029156
8	0.029289
9	0.029201
10	0.029446
11	0.029431
12	0.029472
13	0.029551
14	0.029533

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 [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()
```

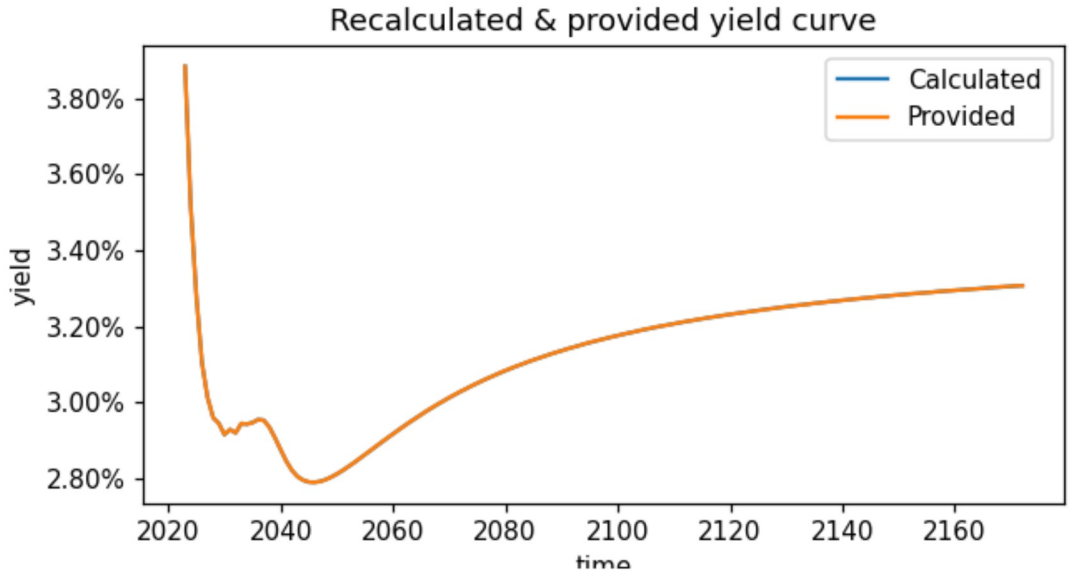
Out[54]:

	Given rates
0	0.03884
1	0.03517
2	0.03281
3	0.03105
4	0.03013

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, c
```

EIOPA curve comparison

Absolute difference in bps

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

Out[58]:

	Abs diff in bps
0	8.030045e-07
1	3.363435e-02
2	4.456042e-02
3	3.522865e-02

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.

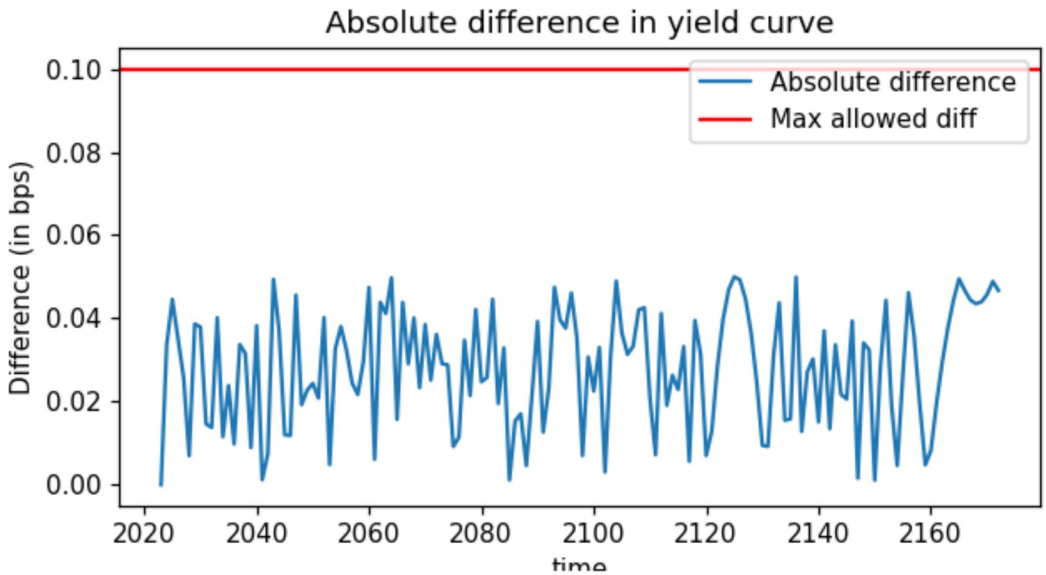
In [59]:
```
result1 = SuccessTest(test_statistics_bdp.values, test_statistics_max_diff_in_bps, te
```

```
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 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 [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      3.884
2      3.517
3      3.281
4      3.105
5      3.013
6      2.960
7      2.945
8      2.916
9      2.929
10     2.920
11     2.945
12     2.943
13     2.947
14     2.955
15     2.953
16     2.935
17     2.907
18     2.876
19     2.846
20     2.822
21     2.805
22     2.796
23     2.791
24     2.790
25     2.792
26     2.797
27     2.804
28     2.812
```

29	2.821
30	2.831
31	2.841
32	2.852
33	2.863
34	2.874
35	2.885
36	2.896
37	2.907
38	2.917
39	2.928
40	2.938
41	2.949
42	2.958
43	2.968
44	2.978
45	2.987
46	2.996
47	3.004
48	3.013
49	3.021
50	3.029
51	3.036
52	3.044
53	3.051
54	3.058
55	3.065
56	3.071
57	3.078
58	3.084
59	3.090
60	3.096
61	3.101
62	3.107
63	3.112
64	3.117
65	3.122
66	3.127
67	3.132
68	3.136
69	3.141
70	3.145
71	3.149
72	3.154
73	3.158
74	3.162
75	3.165
76	3.169
77	3.173
78	3.176
79	3.180
80	3.183
81	3.186
82	3.190
83	3.193
84	3.196
85	3.199
86	3.202
87	3.204

88	3.207
89	3.210
90	3.213
91	3.215
92	3.218
93	3.220
94	3.223
95	3.225
96	3.227
97	3.230
98	3.232
99	3.234
100	3.236
101	3.238
102	3.240
103	3.242
104	3.244
105	3.246
106	3.248
107	3.250
108	3.252
109	3.254
110	3.256
111	3.257
112	3.259
113	3.261
114	3.262
115	3.264
116	3.266
117	3.267
118	3.269
119	3.270
120	3.272
121	3.273
122	3.275
123	3.276
124	3.278
125	3.279
126	3.280
127	3.282
128	3.283
129	3.284
130	3.286
131	3.287
132	3.288
133	3.289
134	3.290
135	3.292
136	3.293
137	3.294
138	3.295
139	3.296
140	3.297
141	3.298
142	3.299
143	3.300
144	3.302
145	3.303
146	3.304

```
147    3.305
148    3.306
149    3.307
150    3.307
```