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 [218…

```
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 [219…

```
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 [220…

```
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 [221…

```
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 [222…

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

Parameter input

Parameters sheet

In [223...
```
param_raw.head()
```

Out[223...

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	0.108278	0.108278	0.108278	0.108278	0.108278	

5 rows × 106 columns

The country selected is:

In [224...
```
country = "United Kingdom"
```

In [225...
```
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 [226...
```
extra_param
```

Out[226...
```
Country
Coupon_freq     1.000000
LLP            50.000000
Convergence    40.000000
UFR             3.450000
alpha           0.087489
CRA             0.000000
Name: United Kingdom_Values, dtype: float64
```

In [227...
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [228…
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [229…
```
maturities_country.head(15)
```

Out[229…
```
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 [230…
```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector **Qb** provided as input

In [231…
```
Qb
```

Out[231…
```
Country
1    -7.276664
2    -7.395127
3     4.283807
4     0.313476
5    -0.542274
6     0.998425
7    -0.541668
8     0.051394
9     2.443213
10   -1.246985
11   -0.008848
12    0.118104
13   -0.014415
14   -0.013646
15   -0.302387
16    0.001391
```

```
17    0.001105
18    0.001014
19    0.000973
20    0.155701
21   -0.006230
22   -0.005737
23   -0.005371
24   -0.004988
25   -0.004624
26   -0.004261
27   -0.003922
28   -0.003527
29   -0.003354
30   -0.347101
31    0.012466
32    0.011552
33    0.010938
34    0.010301
35    0.009723
36    0.009177
37    0.008666
38    0.008186
39    0.007736
40    0.007307
41    0.006920
42    0.006486
43    0.006324
44    0.005240
45    0.007675
46   -0.002954
47    0.035260
48   -0.108835
49    0.427809
50   -0.236663
Name: United Kingdom Values, dtype: float64
```

In [232…

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

In [233…

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

Back to the top

Calibration parameters and calibration vector provided by EIOPA

In [234…

```
# 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 [235…

```
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 [236…

```
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 [237…
```
r_Target.head(15)
```

Out[237…

	Recalculated rates
0	0.059135
1	0.056602
2	0.053574
3	0.051068
4	0.049057
5	0.047432
6	0.046164
7	0.045194
8	0.044505
9	0.044063
10	0.043745
11	0.043500
12	0.043301
13	0.043126
14	0.042956

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 [238…
```
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 [239…]
```
target_curve = pd.DataFrame(target_curve,columns=['Given rates'])
```

EIOPA curve provided

Yield curve provided by EIOPA

In [240…]
```
target_curve.head()
```

Out[240…]

	Given rates
0	0.05913
1	0.05660
2	0.05357
3	0.05107
4	0.04906

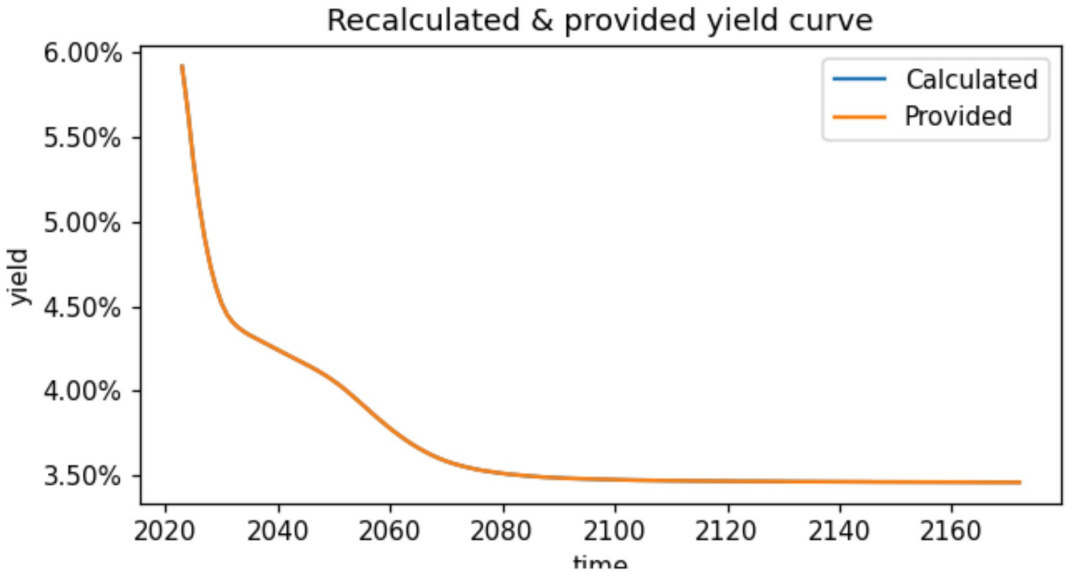
In [241...

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

In [242...

```
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 [243...

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

EIOPA curve comparison

Absolute difference in bps

In [244...

```
test_statistics_bdp.head()
```

Out[244...

	Abs diff in bps
0	0.050000
1	0.022909
2	0.042954
3	0.016620

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 [245...

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

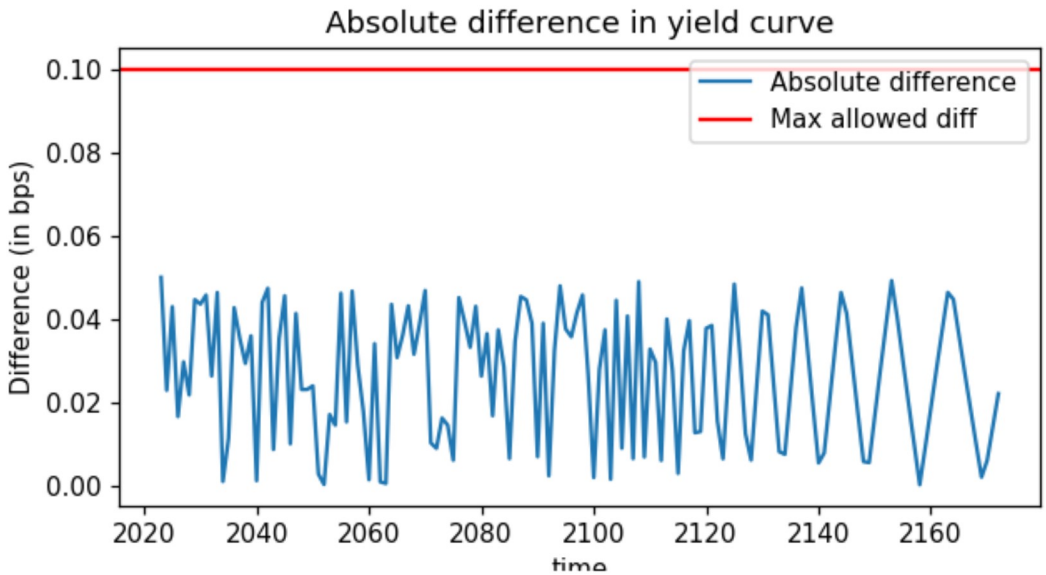
```
Test passed
Test passed
```

In [246...

```
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 [247...

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

Out[247...

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [248...

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

Out[248...

```
Country
1      5.913
2      5.660
3      5.357
4      5.107
5      4.906
6      4.743
7      4.616
8      4.519
9      4.451
10     4.406
11     4.375
12     4.350
13     4.330
14     4.313
15     4.296
16     4.278
17     4.261
18     4.243
19     4.226
20     4.209
21     4.192
22     4.175
23     4.158
24     4.141
25     4.122
26     4.103
27     4.082
28     4.059
```

29	4.035
30	4.009
31	3.981
32	3.952
33	3.923
34	3.893
35	3.863
36	3.835
37	3.807
38	3.780
39	3.754
40	3.730
41	3.707
42	3.685
43	3.665
44	3.647
45	3.629
46	3.614
47	3.599
48	3.587
49	3.575
50	3.565
51	3.556
52	3.548
53	3.541
54	3.534
55	3.529
56	3.523
57	3.519
58	3.514
59	3.510
60	3.507
61	3.504
62	3.501
63	3.498
64	3.496
65	3.493
66	3.491
67	3.490
68	3.488
69	3.486
70	3.485
71	3.484
72	3.482
73	3.481
74	3.480
75	3.479
76	3.479
77	3.478
78	3.477
79	3.476
80	3.476
81	3.475
82	3.474
83	3.474
84	3.473
85	3.473
86	3.472
87	3.472

88	3.472
89	3.471
90	3.471
91	3.471
92	3.470
93	3.470
94	3.470
95	3.469
96	3.469
97	3.469
98	3.469
99	3.468
100	3.468
101	3.468
102	3.468
103	3.468
104	3.467
105	3.467
106	3.467
107	3.467
108	3.467
109	3.466
110	3.466
111	3.466
112	3.466
113	3.466
114	3.466
115	3.465
116	3.465
117	3.465
118	3.465
119	3.465
120	3.465
121	3.465
122	3.465
123	3.464
124	3.464
125	3.464
126	3.464
127	3.464
128	3.464
129	3.464
130	3.464
131	3.464
132	3.463
133	3.463
134	3.463
135	3.463
136	3.463
137	3.463
138	3.463
139	3.463
140	3.463
141	3.463
142	3.462
143	3.462
144	3.462
145	3.462
146	3.462

```
147    3.462
148    3.462
149    3.462
150    3.462
```