EIOPA RISK-FREE CURVE MAY-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 May 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_20230531_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_20230531_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 [156…

```
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 [157…

```
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 [158…

```
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 [159…

```
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 [160…

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

Parameter input

Parameters sheet

In [161...
```
param_raw.head()
```

Out[161...

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	0.110654	0.110654	0.110654	0.110654	0.110654	

5 rows × 106 columns

The country selected is:

In [162...
```
country = "United Kingdom"
```

In [163...
```
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 [164...
```
extra_param
```

Out[164...
```
Country
Coupon_freq     1.000000
LLP            50.000000
Convergence    40.000000
UFR             3.450000
alpha           0.077586
CRA             0.000000
Name: United Kingdom_Values, dtype: float64
```

In [165...
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [166...
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [167...
```
maturities_country.head(15)
```

Out[167...
```
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 [168...
```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector **Qb** provided as input

In [169...
```
Qb
```

Out[169...
```
Country
1    -11.340681
2     -6.808431
3      7.699075
4     -4.088759
5      1.223079
6      0.761351
7      0.155821
8      0.039680
9      2.184775
10    -0.989751
11    -0.017364
12     0.033486
13    -0.017750
14    -0.016611
15    -0.401236
16     0.003316
```

```
17     0.002777
18     0.002557
19     0.002406
20     0.195569
21    -0.006604
22    -0.005918
23    -0.005422
24    -0.004896
25    -0.004397
26    -0.003898
27    -0.003426
28    -0.002882
29    -0.002621
30    -0.384027
31     0.014901
32     0.013667
33     0.012839
34     0.011984
35     0.011211
36     0.010485
37     0.009808
38     0.009175
39     0.008583
40     0.008020
41     0.007518
42     0.006941
43     0.006782
44     0.005186
45     0.009066
46    -0.007401
47     0.052180
48    -0.172077
49     0.663396
50    -0.426110
Name: United Kingdom Values, dtype: float64
```

In [170…

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

In [171…

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

Back to the top

Calibration parameters and calibration vector provided by EIOPA

In [172…

```
# 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 [173…

```
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 [174…

```
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 [175…
```
r_Target.head(15)
```

Out[175…

	Recalculated rates
0	0.054428
1	0.052222
2	0.049820
3	0.047891
4	0.046238
5	0.044904
6	0.043891
7	0.043153
8	0.042652
9	0.042343
10	0.042137
11	0.041988
12	0.041871
13	0.041767
14	0.041662

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_20230531_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 [176…
```
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 [177…

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

EIOPA curve provided

Yield curve provided by EIOPA

In [178…

```
target_curve.head()
```

Out[178…

	Given rates
0	0.05443
1	0.05222
2	0.04982
3	0.04789
4	0.04624

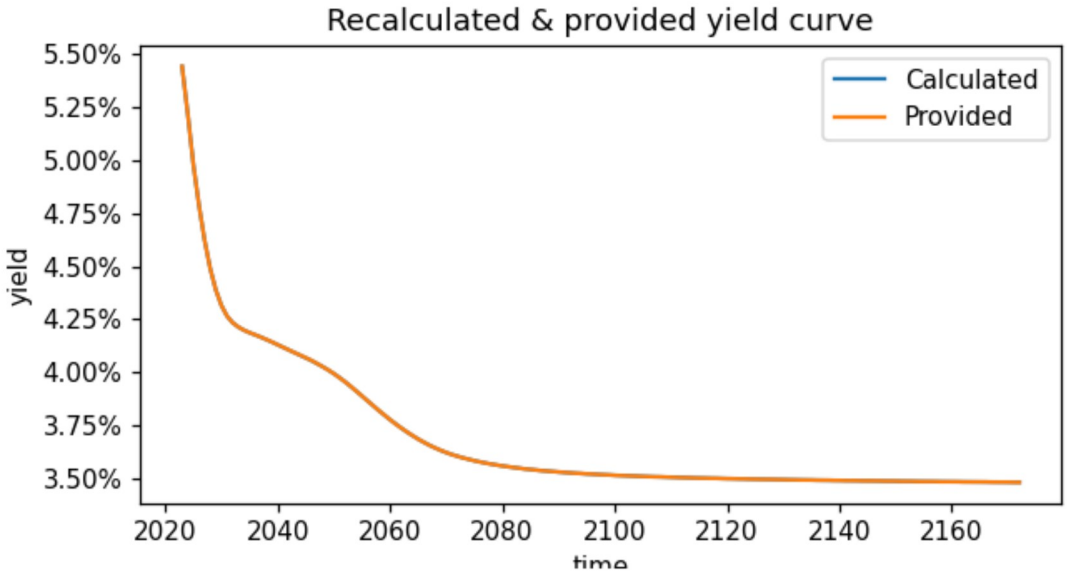
In [179…

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

In [180…

```
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 [181…

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

EIOPA curve comparison

Absolute difference in bps

In [182…

```
test_statistics_bdp.head()
```

Out[182…

	Abs diff in bps
0	0.020001
1	0.019104
2	0.004219
3	0.009368

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 [183...

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

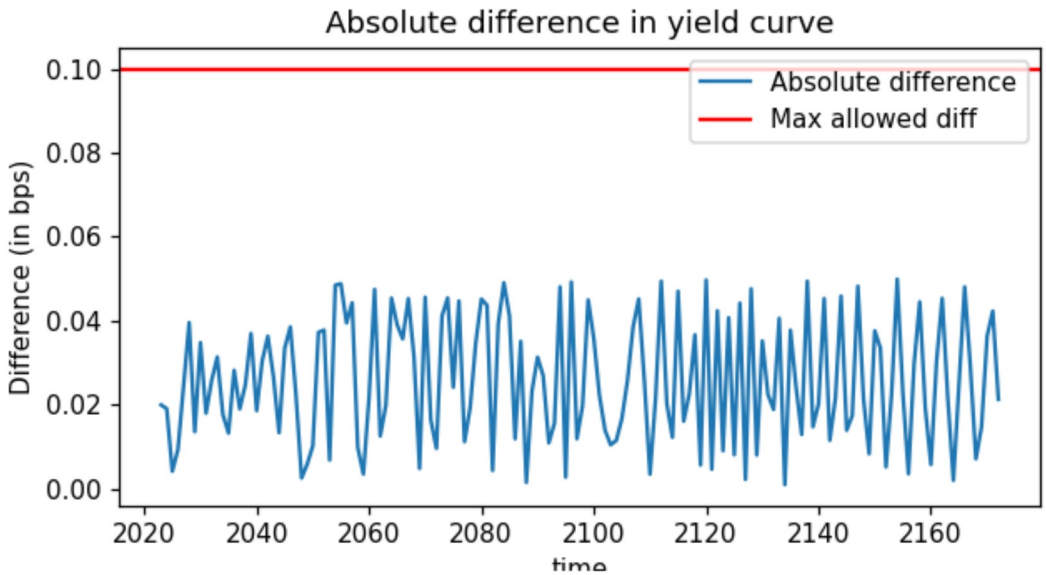
```
Test passed
Test passed
```

In [184...

```
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 May 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_20230531_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230531_Term_Structures.xlsx*.

In [185...

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

Out[185...

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [186...

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

Out[186...

```
Country
1      5.443
2      5.222
3      4.982
4      4.789
5      4.624
6      4.490
7      4.389
8      4.315
9      4.265
10     4.234
11     4.214
12     4.199
13     4.187
14     4.177
15     4.166
16     4.155
17     4.143
18     4.130
19     4.118
20     4.105
21     4.093
22     4.081
23     4.069
24     4.056
25     4.042
26     4.027
27     4.011
28     3.994
```

29	3.975
30	3.955
31	3.934
32	3.911
33	3.889
34	3.865
35	3.842
36	3.820
37	3.798
38	3.777
39	3.756
40	3.737
41	3.719
42	3.702
43	3.685
44	3.670
45	3.656
46	3.644
47	3.632
48	3.622
49	3.612
50	3.604
51	3.596
52	3.590
53	3.583
54	3.577
55	3.572
56	3.567
57	3.563
58	3.558
59	3.555
60	3.551
61	3.548
62	3.544
63	3.542
64	3.539
65	3.536
66	3.534
67	3.532
68	3.530
69	3.528
70	3.526
71	3.524
72	3.523
73	3.521
74	3.520
75	3.518
76	3.517
77	3.516
78	3.514
79	3.513
80	3.512
81	3.511
82	3.510
83	3.509
84	3.508
85	3.507
86	3.507
87	3.506

88	3.505
89	3.504
90	3.503
91	3.503
92	3.502
93	3.501
94	3.501
95	3.500
96	3.500
97	3.499
98	3.498
99	3.498
100	3.497
101	3.497
102	3.496
103	3.496
104	3.495
105	3.495
106	3.495
107	3.494
108	3.494
109	3.493
110	3.493
111	3.492
112	3.492
113	3.492
114	3.491
115	3.491
116	3.491
117	3.490
118	3.490
119	3.489
120	3.489
121	3.489
122	3.488
123	3.488
124	3.488
125	3.488
126	3.487
127	3.487
128	3.487
129	3.486
130	3.486
131	3.486
132	3.485
133	3.485
134	3.485
135	3.485
136	3.484
137	3.484
138	3.484
139	3.484
140	3.483
141	3.483
142	3.483
143	3.483
144	3.483
145	3.482
146	3.482

```
147    3.482
148    3.482
149    3.481
150    3.481
```