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 [187…

```
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 [188…

```
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 [189…

```
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 [190…

```
#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 [191…

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

Parameter input

Parameters sheet

In [192...
```
param_raw.head()
```

Out[192...

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	0.11485	0.11485	0.11485	0.11485	0.11485	

5 rows × 106 columns

The country selected is:

In [193...
```
country = "United Kingdom"
```

In [194...
```
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 [195...
```
extra_param
```

Out[195...
```
Country
Coupon_freq     1.000000
LLP            50.000000
Convergence    40.000000
UFR             3.450000
alpha           0.091615
CRA             0.000000
Name: United Kingdom_Values, dtype: float64
```

In [196...
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [197...
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [198...
```
maturities_country.head(15)
```

Out[198...
```
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 [199...
```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector **Qb** provided as input

In [200...
```
Qb
```

Out[200...
```
Country
1    -6.747968
2    -4.233741
3     4.700777
4    -2.518200
5     0.734851
6     0.461474
7     0.092403
8     0.021753
9     1.355513
10   -0.607475
11   -0.009073
12    0.022799
13   -0.009717
14   -0.009393
15   -0.251393
16    0.000657
```

```
17    0.000635
18    0.000614
19    0.000593
20    0.123098
21   -0.004152
22   -0.004013
23   -0.003879
24   -0.003750
25   -0.003625
26   -0.003504
27   -0.003387
28   -0.003274
29   -0.003165
30   -0.249502
31    0.006247
32    0.006039
33    0.005838
34    0.005643
35    0.005455
36    0.005273
37    0.005097
38    0.004927
39    0.004763
40    0.004604
41    0.004450
42    0.004302
43    0.004159
44    0.004020
45    0.003886
46    0.003756
47    0.003631
48    0.003510
49    0.003393
50    0.093182
Name: United Kingdom Values, dtype: float64
```

In [201…

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

In [202…

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

Back to the top

Calibration parameters and calibration vector provided by EIOPA

In [203…

```
# 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 [204…

```
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 [205…

```
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 [206…

```
r_Target.head(15)
```

Out[206…

	Recalculated rates
0	0.052528
1	0.050322
2	0.047920
3	0.045991
4	0.044338
5	0.043004
6	0.041991
7	0.041253
8	0.040752
9	0.040443
10	0.040237
11	0.040088
12	0.039971
13	0.039867
14	0.039762

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 [207…

```
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 [208…

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

EIOPA curve provided

Yield curve provided by EIOPA

In [209…

```
target_curve.head()
```

Out[209…

	Given rates
0	0.05253
1	0.05032
2	0.04792
3	0.04599
4	0.04434

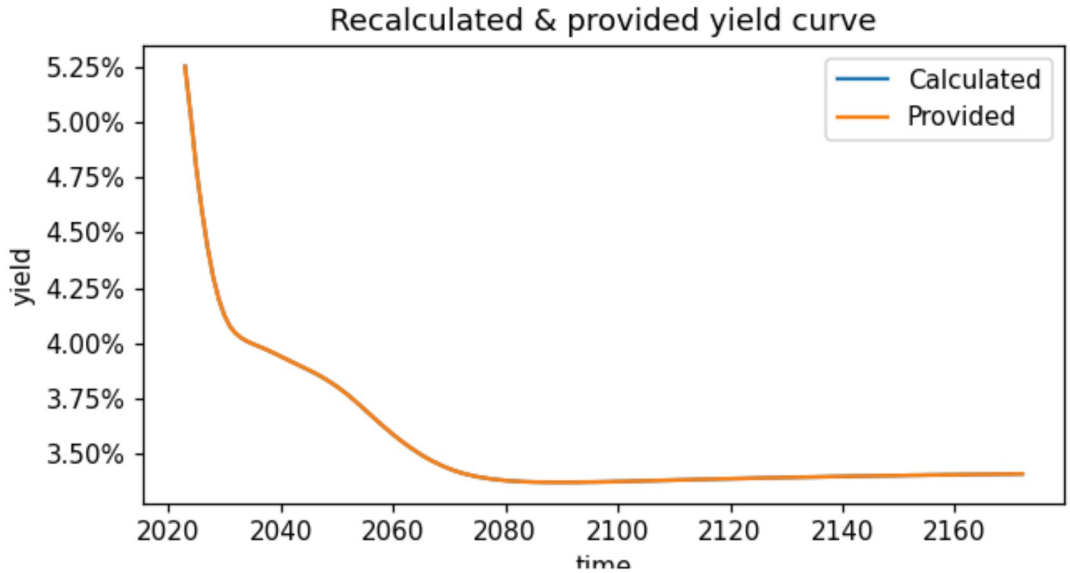
In [210…

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

In [211…

```
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 [212…

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

EIOPA curve comparison

Absolute difference in bps

In [213…

```
test_statistics_bdp.head()
```

Out[213…

	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 [214...

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

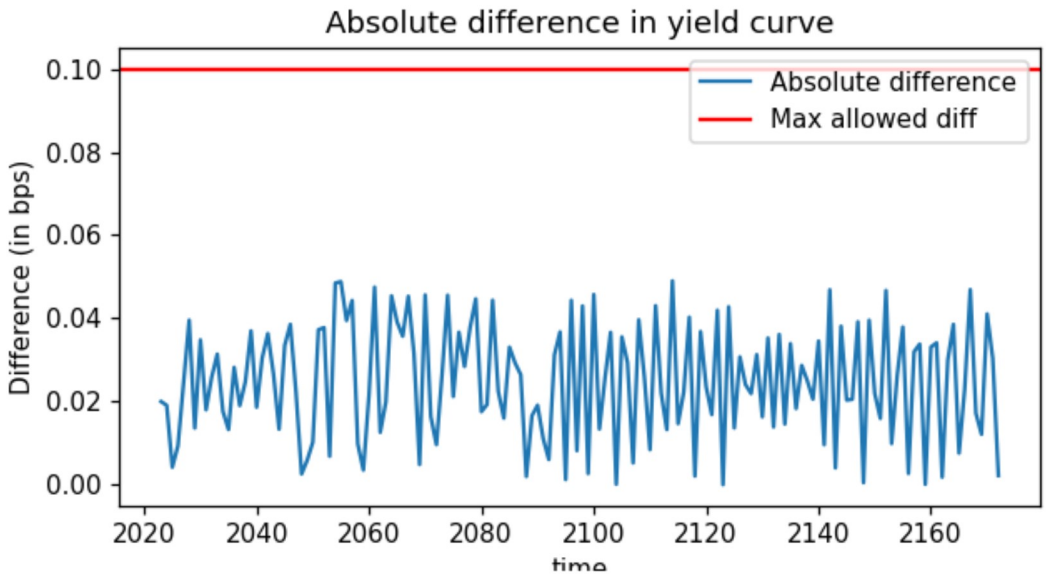
```
Test passed
Test passed
```

In [215...

```
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 [216...

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

Out[216...

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [217...

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

Out[217...

```
Country
1      5.253
2      5.032
3      4.792
4      4.599
5      4.434
6      4.300
7      4.199
8      4.125
9      4.075
10     4.044
11     4.024
12     4.009
13     3.997
14     3.987
15     3.976
16     3.965
17     3.953
18     3.940
19     3.928
20     3.915
21     3.903
22     3.891
23     3.879
24     3.866
25     3.852
26     3.837
27     3.821
28     3.804
```

29	3.785
30	3.765
31	3.744
32	3.721
33	3.699
34	3.675
35	3.652
36	3.630
37	3.608
38	3.587
39	3.566
40	3.547
41	3.529
42	3.512
43	3.495
44	3.480
45	3.466
46	3.454
47	3.442
48	3.432
49	3.422
50	3.414
51	3.407
52	3.401
53	3.395
54	3.391
55	3.387
56	3.383
57	3.381
58	3.378
59	3.376
60	3.375
61	3.373
62	3.372
63	3.371
64	3.371
65	3.370
66	3.370
67	3.370
68	3.370
69	3.370
70	3.370
71	3.370
72	3.371
73	3.371
74	3.371
75	3.372
76	3.372
77	3.373
78	3.374
79	3.374
80	3.375
81	3.375
82	3.376
83	3.377
84	3.377
85	3.378
86	3.379
87	3.379

88	3.380
89	3.381
90	3.381
91	3.382
92	3.383
93	3.383
94	3.384
95	3.384
96	3.385
97	3.386
98	3.386
99	3.387
100	3.387
101	3.388
102	3.389
103	3.389
104	3.390
105	3.390
106	3.391
107	3.391
108	3.392
109	3.392
110	3.393
111	3.393
112	3.394
113	3.394
114	3.395
115	3.395
116	3.396
117	3.396
118	3.397
119	3.397
120	3.398
121	3.398
122	3.398
123	3.399
124	3.399
125	3.400
126	3.400
127	3.400
128	3.401
129	3.401
130	3.402
131	3.402
132	3.402
133	3.403
134	3.403
135	3.403
136	3.404
137	3.404
138	3.404
139	3.405
140	3.405
141	3.405
142	3.406
143	3.406
144	3.406
145	3.407
146	3.407

```
147    3.407
148    3.407
149    3.408
150    3.408
```