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 [249…

```
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 [250…

```
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 [251…

```
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 [252…

```
#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 [253…

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

Parameter input

Parameters sheet

In [254...
```
param_raw.head()
```

Out[254...

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 [255...
```
country = "Italy"
```

In [256...
```
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 [257...
```
extra_param
```

Out[257...
```
Country
Coupon_freq     1.00000
LLP            20.00000
Convergence    40.00000
UFR             3.45000
alpha           0.11485
CRA            10.00000
Name: Italy_Values, dtype: float64
```

In [258...
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [259…
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [260…
```
maturities_country.head(15)
```

Out[260…
```
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 [261…
```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector **Qb** provided as input

In [262…
```
Qb
```

Out[262…
```
Country
1    -9.595510
2     2.153498
3     0.215865
4     1.148100
5     0.259863
6    -0.543009
7     0.896743
8    -1.551002
9     3.905817
10   -7.654708
11    8.688016
12   -3.554557
13   -0.008646
14   -0.008357
15   -0.988909
16    0.019430
```

```
17    0.018782
18    0.018156
19    0.017550
20    0.632869
Name: Italy Values, dtype: float64
```

In [263…

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

In [264…

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

Back to the top

Calibration parameters and calibration vector provided by EIOPA

In [265…

```
# 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 [266…

```
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 [267…

```
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 [268...
```
r_Target.head(15)
```

Out[268...

	Recalculated rates
0	0.037390
1	0.034003
2	0.031518
3	0.029961
4	0.029104
5	0.028650
6	0.028414
7	0.028336
8	0.028342
9	0.028424
10	0.028434
11	0.028642
12	0.028793
13	0.028804
14	0.028672

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 [269...
```
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 [270…

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

EIOPA curve provided

Yield curve provided by EIOPA

In [271…

```
target_curve.head()
```

Out[271…

	Given rates
0	0.03739
1	0.03400
2	0.03152
3	0.02996
4	0.02910

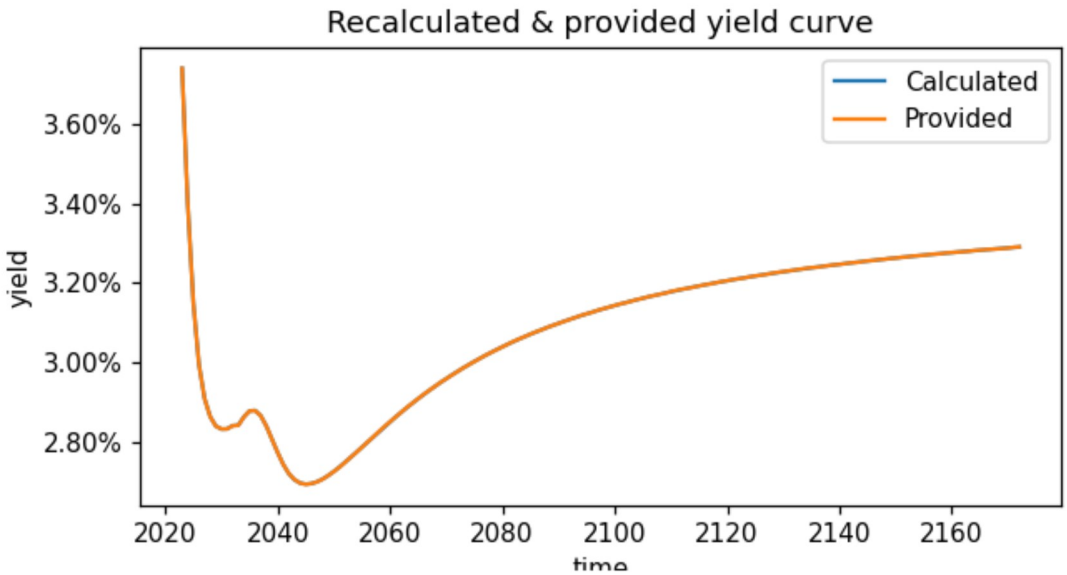
In [272...

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

In [273...

```
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 [274...

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

EIOPA curve comparison

Absolute difference in bps

In [275...

```
test_statistics_bdp.head()
```

Out[275...

	Abs diff in bps
0	9.533244e-07
1	3.476867e-02
2	2.154654e-02
3	5.658497e-03

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 [276...

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

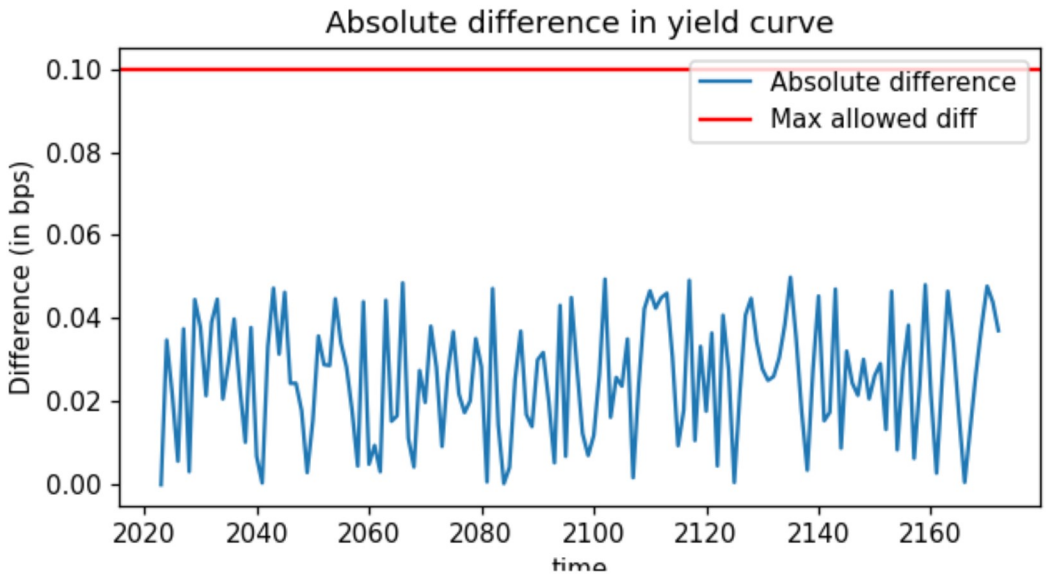
```
Test passed
Test passed
```

In [277...

```
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 [278...

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

Out[278...

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [279...

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

Out[279...

```
Country
1      3.739
2      3.400
3      3.152
4      2.996
5      2.910
6      2.865
7      2.841
8      2.834
9      2.834
10     2.842
11     2.843
12     2.864
13     2.879
14     2.880
15     2.867
16     2.841
17     2.808
18     2.775
19     2.745
20     2.722
21     2.707
22     2.698
23     2.695
24     2.697
25     2.701
26     2.708
27     2.717
28     2.727
```

29	2.738
30	2.750
31	2.763
32	2.775
33	2.788
34	2.801
35	2.814
36	2.827
37	2.840
38	2.852
39	2.864
40	2.876
41	2.888
42	2.899
43	2.910
44	2.921
45	2.931
46	2.941
47	2.951
48	2.960
49	2.969
50	2.978
51	2.987
52	2.995
53	3.003
54	3.011
55	3.019
56	3.026
57	3.033
58	3.040
59	3.047
60	3.054
61	3.060
62	3.066
63	3.072
64	3.078
65	3.083
66	3.089
67	3.094
68	3.099
69	3.104
70	3.109
71	3.114
72	3.119
73	3.123
74	3.127
75	3.132
76	3.136
77	3.140
78	3.144
79	3.148
80	3.151
81	3.155
82	3.159
83	3.162
84	3.166
85	3.169
86	3.172
87	3.175

88	3.179
89	3.182
90	3.185
91	3.187
92	3.190
93	3.193
94	3.196
95	3.198
96	3.201
97	3.204
98	3.206
99	3.209
100	3.211
101	3.213
102	3.216
103	3.218
104	3.220
105	3.222
106	3.225
107	3.227
108	3.229
109	3.231
110	3.233
111	3.235
112	3.237
113	3.239
114	3.240
115	3.242
116	3.244
117	3.246
118	3.247
119	3.249
120	3.251
121	3.252
122	3.254
123	3.256
124	3.257
125	3.259
126	3.260
127	3.262
128	3.263
129	3.265
130	3.266
131	3.268
132	3.269
133	3.270
134	3.272
135	3.273
136	3.274
137	3.276
138	3.277
139	3.278
140	3.279
141	3.280
142	3.282
143	3.283
144	3.284
145	3.285
146	3.286

```
147    3.287
148    3.288
149    3.290
150    3.291
```