EIOPA RISK-FREE CURVE DECEMBER-22 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 December 2022. 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_20221231_Qb_SW.xlsx*.

For this example, the curve without the volatility adjustment (VA) is used. It can be found in the sheet *SW_Qb_no_VA*. This example is focused on the EUR curve, but this example can be easily modified for any other curve.

The target maturities (`T_Obs`), the additional parameters (`UFR` and `alpha`), and the given curve can be found in the Excel *EIOPA_RFR_20221231_Term_Structures.xlsx*, sheet *RFR_spot_no_VA*.

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 [270…

```
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 [271…

```
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 [272…

```
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 [273…

```
#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 [274…

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

Parameter input

Parameters sheet

In [275...
```
param_raw.head()
```

Out[275...

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	0.120275	0.120275	0.120275	0.120275	0.120275	

5 rows × 106 columns

The country selected is:

In [276...
```
country = "Italy"
```

In [277...
```
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 [278...
```
extra_param
```

Out[278...
```
Country
Coupon_freq     1.000000
LLP            20.000000
Convergence    40.000000
UFR             3.450000
alpha           0.120275
CRA            10.000000
Name: Italy_Values, dtype: float64
```

In [279...
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [280…
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [281…
```
maturities_country.head(15)
```

Out[281…
```
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 [282…
```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector Qb provided as input

In [283…
```
Qb
```

Out[283…
```
Country
1     10.410356
2    -12.456054
3      5.667547
4     -0.016150
5     -0.539124
6     -1.012348
7      1.345744
8      0.092938
9     -1.729729
10     5.078942
11    -6.952002
12     4.111449
13    -0.028440
14    -0.027492
15    -1.727604
16     0.024249
```

```
17     0.023440
18     0.022658
19     0.021903
20     0.770104
Name: Italy Values, dtype: float64
```

In [284…

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

In [285…

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

Back to the top

Calibration parameters and calibration vector provided by EIOPA

In [286…

```
# 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 [287…

```
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 [288…

```
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 [289…
```
r_Target.head(15)
```

Out[289…

	Recalculated rates
0	0.031760
1	0.032949
2	0.032034
3	0.031525
4	0.031308
5	0.031099
6	0.030909
7	0.030861
8	0.030879
9	0.030919
10	0.030998
11	0.030852
12	0.030707
13	0.030528
14	0.030218

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_20221231_Term_Structures.xlsx*, sheet *RFR_spot_no_VA*.

In [290…
```
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 [291…
```
target_curve = pd.DataFrame(target_curve,columns=['Given rates'])
```

EIOPA curve provided

Yield curve provided by EIOPA

In [292…
```
target_curve.head()
```

Out[292…

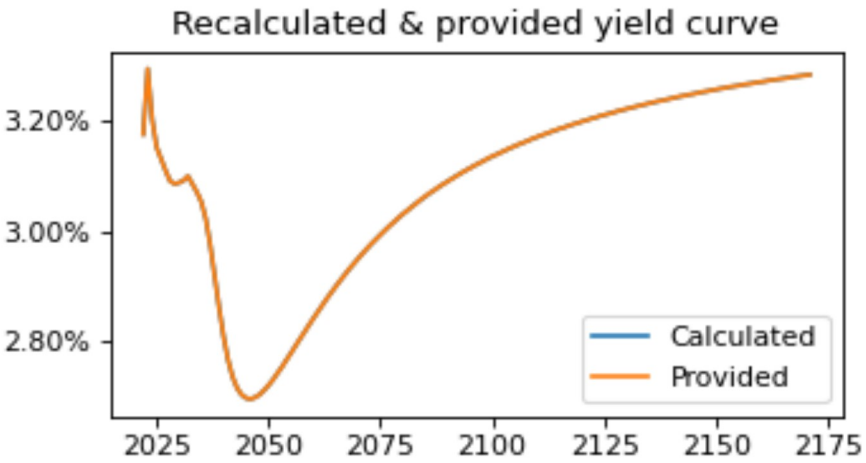
	Given rates
0	0.03176
1	0.03295
2	0.03203
3	0.03152
4	0.03131

In [293…
```
x_data_label = range(2022,2022+r_Target.shape[0],1)
```

In [294...

```
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 [295...

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

EIOPA curve comparison

Absolute difference in bps

In [296...

```
test_statistics_bdp.head()
```

Out[296...

	Abs diff in bps
0	3.099100e-07
1	7.135360e-03
2	4.009214e-02
3	4.545501e-02
4	1.724362e-02

Back to the top

Test 1; Success criteria

The successful application of the success criteria marks the completion of the test.

In [297...

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

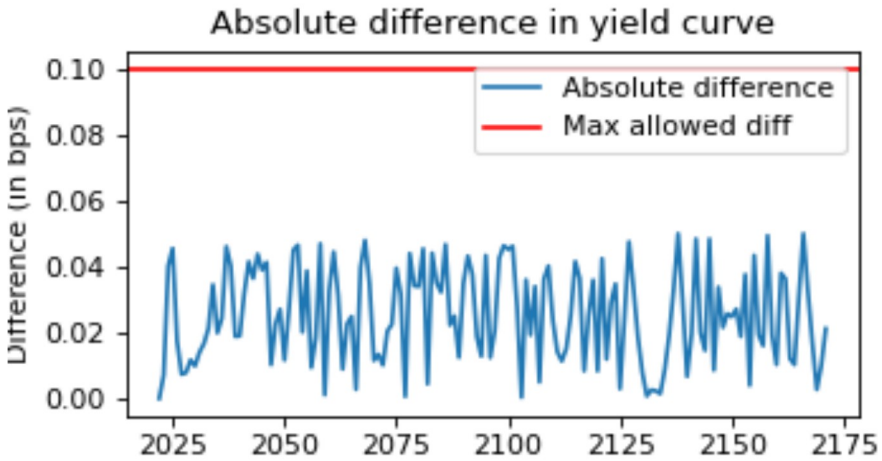
```
Test passed
Test passed
```

In [298...

```
x_data_label = range(2022,2022+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

The EIOPA curve generated for December 2022 has passed the success criteria. Based on the preformed tests, 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_20221231_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20221231_Term_Structures.xlsx*.

In [299...

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

Out[299...

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [300...

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

Out[300...

```
Country
1      3.176
2      3.295
3      3.203
4      3.152
5      3.131
6      3.110
7      3.091
8      3.086
9      3.088
10     3.092
11     3.100
12     3.085
13     3.071
14     3.053
15     3.022
16     2.974
17     2.916
18     2.859
19     2.807
20     2.765
21     2.735
22     2.715
23     2.703
24     2.697
25     2.695
26     2.698
27     2.703
28     2.711
29     2.720
30     2.730
31     2.742
32     2.753
33     2.766
34     2.778
35     2.791
36     2.804
37     2.816
38     2.829
39     2.841
40     2.853
```

41	2.865
42	2.877
43	2.888
44	2.899
45	2.910
46	2.920
47	2.931
48	2.940
49	2.950
50	2.959
51	2.968
52	2.977
53	2.985
54	2.993
55	3.001
56	3.009
57	3.016
58	3.024
59	3.031
60	3.037
61	3.044
62	3.050
63	3.057
64	3.063
65	3.069
66	3.074
67	3.080
68	3.085
69	3.090
70	3.095
71	3.100
72	3.105
73	3.110
74	3.114
75	3.119
76	3.123
77	3.127
78	3.132
79	3.136
80	3.139
81	3.143
82	3.147
83	3.151
84	3.154
85	3.158
86	3.161
87	3.164
88	3.168
89	3.171
90	3.174
91	3.177
92	3.180
93	3.183
94	3.186
95	3.188
96	3.191
97	3.194
98	3.196
99	3.199

100	3.201
101	3.204
102	3.206
103	3.209
104	3.211
105	3.213
106	3.215
107	3.218
108	3.220
109	3.222
110	3.224
111	3.226
112	3.228
113	3.230
114	3.232
115	3.234
116	3.236
117	3.237
118	3.239
119	3.241
120	3.243
121	3.245
122	3.246
123	3.248
124	3.249
125	3.251
126	3.253
127	3.254
128	3.256
129	3.257
130	3.259
131	3.260
132	3.262
133	3.263
134	3.264
135	3.266
136	3.267
137	3.268
138	3.270
139	3.271
140	3.272
141	3.274
142	3.275
143	3.276
144	3.277
145	3.278
146	3.280
147	3.281
148	3.282
149	3.283
150	3.284