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 [239…

```
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 [240…

```
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 [241…

```
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 [242…

```
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 [243…

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

Parameter input

Parameters sheet

In [244...
```
param_raw.head()
```

Out[244...

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	0.117071	0.117071	0.117071	0.117071	0.117071	

5 rows × 106 columns

The country selected is:

In [245...
```
country = "Italy"
```

In [246...
```
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 [247...
```
extra_param
```

Out[247...
```
Country
Coupon_freq     1.000000
LLP            20.000000
Convergence    40.000000
UFR             3.450000
alpha           0.117071
CRA            10.000000
Name: Italy_Values, dtype: float64
```

In [248...
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [249…
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [250…
```
maturities_country.head(15)
```

Out[250…
```
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 [251…
```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector Qb provided as input

In [252…
```
Qb
```

Out[252…
```
Country
1     11.236337
2    -13.445829
3      6.127884
4     -0.027641
5     -0.579326
6     -1.079164
7      1.441260
8      0.093102
9     -1.834204
10     5.383635
11    -7.375500
12     4.365883
13    -0.040046
14    -0.035400
15    -1.816156
16     0.027633
```

```
17     0.054314
18    -0.057801
19     0.354471
20     0.517303
Name: Italy Values, dtype: float64
```

In [253…

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

In [254…

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

Back to the top

Calibration parameters and calibration vector provided by EIOPA

In [255…

```
# 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 [256…

```
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 [257…

```
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 [258...
```
r_Target.head(15)
```

Out[258...

	Recalculated rates
0	0.033660
1	0.034849
2	0.033934
3	0.033425
4	0.033208
5	0.032999
6	0.032809
7	0.032761
8	0.032779
9	0.032819
10	0.032898
11	0.032752
12	0.032607
13	0.032428
14	0.032118

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 [259...
```
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 [260…]
```
target_curve = pd.DataFrame(target_curve,columns=['Given rates'])
```

EIOPA curve provided

Yield curve provided by EIOPA

In [261…]
```
target_curve.head()
```

Out[261…]

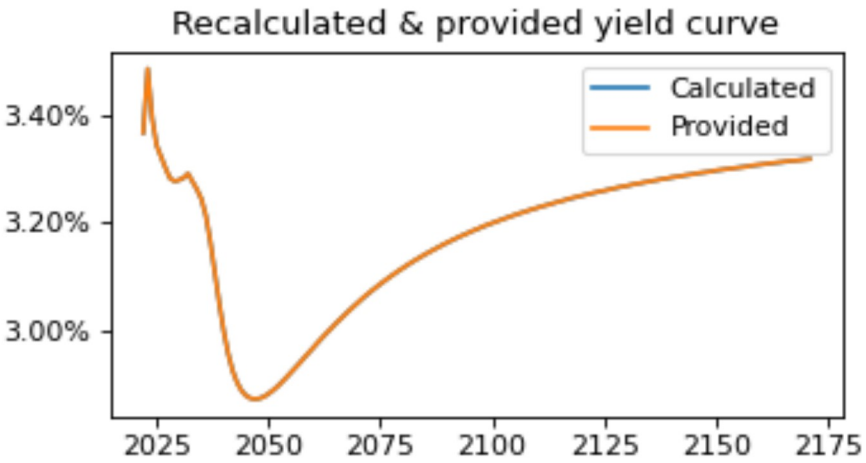
	Given rates
0	0.03366
1	0.03485
2	0.03393
3	0.03342
4	0.03321

In [262…]
```
x_data_label = range(2022,2022+r_Target.shape[0],1)
```

In [263…

```
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 [264…

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

EIOPA curve comparison

Absolute difference in bps

In [265…

```
test_statistics_bdp.head()
```

Out[265…

	Abs diff in bps
0	6.826475e-07
1	7.134948e-03
2	4.009261e-02
3	4.545553e-02
4	1.724305e-02

Back to the top

Test 1; Success criteria

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

In [266...

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

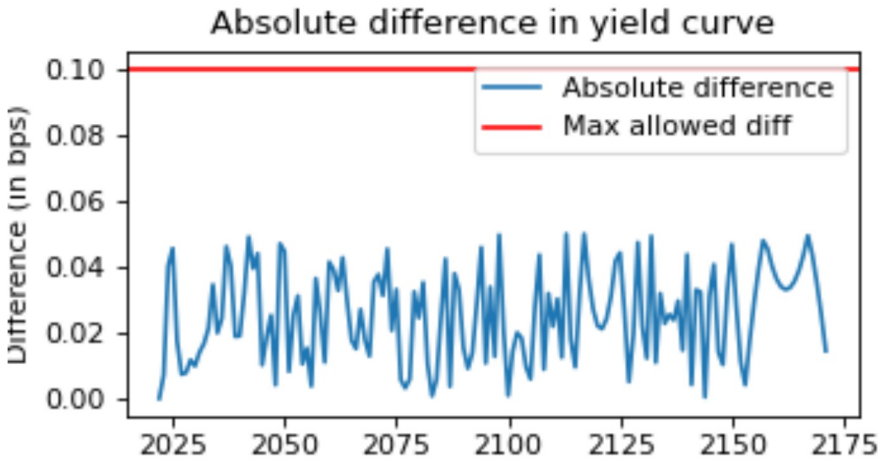
```
Test passed
Test passed
```

In [267...

```
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 [268...

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

Out[268...

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [269...

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

Out[269...

```
Country
1      3.366
2      3.485
3      3.393
4      3.342
5      3.321
6      3.300
7      3.281
8      3.276
9      3.278
10     3.282
11     3.290
12     3.275
13     3.261
14     3.243
15     3.212
16     3.164
17     3.106
18     3.049
19     2.997
20     2.955
21     2.923
22     2.901
23     2.886
24     2.877
25     2.872
26     2.871
27     2.872
28     2.875
29     2.881
30     2.887
31     2.894
32     2.902
33     2.911
34     2.920
35     2.929
36     2.938
37     2.948
38     2.957
39     2.966
40     2.976
```

41	2.985
42	2.994
43	3.002
44	3.011
45	3.019
46	3.027
47	3.035
48	3.043
49	3.050
50	3.058
51	3.065
52	3.072
53	3.078
54	3.085
55	3.091
56	3.097
57	3.103
58	3.109
59	3.114
60	3.120
61	3.125
62	3.130
63	3.135
64	3.140
65	3.144
66	3.149
67	3.153
68	3.158
69	3.162
70	3.166
71	3.170
72	3.174
73	3.177
74	3.181
75	3.185
76	3.188
77	3.192
78	3.195
79	3.198
80	3.201
81	3.204
82	3.207
83	3.210
84	3.213
85	3.216
86	3.218
87	3.221
88	3.224
89	3.226
90	3.229
91	3.231
92	3.233
93	3.236
94	3.238
95	3.240
96	3.243
97	3.245
98	3.247
99	3.249

100	3.251
101	3.253
102	3.255
103	3.257
104	3.258
105	3.260
106	3.262
107	3.264
108	3.266
109	3.267
110	3.269
111	3.271
112	3.272
113	3.274
114	3.275
115	3.277
116	3.278
117	3.280
118	3.281
119	3.283
120	3.284
121	3.285
122	3.287
123	3.288
124	3.289
125	3.291
126	3.292
127	3.293
128	3.294
129	3.296
130	3.297
131	3.298
132	3.299
133	3.300
134	3.301
135	3.302
136	3.303
137	3.305
138	3.306
139	3.307
140	3.308
141	3.309
142	3.310
143	3.311
144	3.312
145	3.313
146	3.314
147	3.314
148	3.315
149	3.316
150	3.317