EIOPA RISK-FREE CURVE FEBRUARY-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 February 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_20230228_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_20230228_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 [437…

```
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 [437…

```
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 [437…

```
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 [437…

```
#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 [437…

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

Parameter input

Parameters sheet

In [437...
```
param_raw.head()
```

Out[437...

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	0.11601	0.11601	0.11601	0.11601	0.11601	

5 rows × 106 columns

The country selected is:

In [437...
```
country = "United States"
```

In [437...
```
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 [437...
```
extra_param
```

Out[437...
```
Country
Coupon_freq     1.000000
LLP            30.000000
Convergence    40.000000
UFR             3.450000
alpha           0.109125
CRA             0.000000
Name: United States_Values, dtype: float64
```

In [438...
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [438...
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

Maturity vector

Vector of maturities used in the calibration

In [438...
```
maturities_country.head(15)
```

Out[438...
```
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 States_Maturities, dtype: float64
```

In [438...
```
Qb = param_country_raw.iloc[relevant_positions]
```

Calibration vector

Vector Qb provided as input

In [438...
```
Qb
```

Out[438...
```
Country
1     -6.780702
2     -9.277703
3      9.420431
4     -2.646062
5      1.343089
6      0.006569
7      0.458689
8     -0.010351
9     -0.901078
10     0.850882
11    -0.007949
12    -0.007684
13    -0.007427
14    -0.007180
15    -0.002022
16    -0.006879
```

```
17    -0.006650
18    -0.006428
19    -0.006213
20    -0.520537
NaN    0.011577
NaN    0.011191
NaN    0.010818
NaN    0.010457
NaN    0.010108
NaN    0.009771
NaN    0.009445
NaN    0.009130
NaN    0.008826
NaN    0.271691
Name: United States Values, dtype: float64
```

In [438...

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

In [438...

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

Back to the top

Calibration parameters and calibration vector provided by EIOPA

In [438...

```
# 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 [438…

```
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 [438…

```
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 [439…

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

Out[439…

	Recalculated rates
0	0.053380
1	0.048995
2	0.044325
3	0.041394
4	0.039405
5	0.038081
6	0.037210
7	0.036638
8	0.036231
9	0.035921
10	0.035723
11	0.035601
12	0.035513
13	0.035434
14	0.035343

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_20230228_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 [439…

```
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 [439…

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

EIOPA curve provided

Yield curve provided by EIOPA

In [439…

```
target_curve.head()
```

Out[439…

	Given rates
0	0.05338
1	0.04900
2	0.04432
3	0.04139
4	0.03941

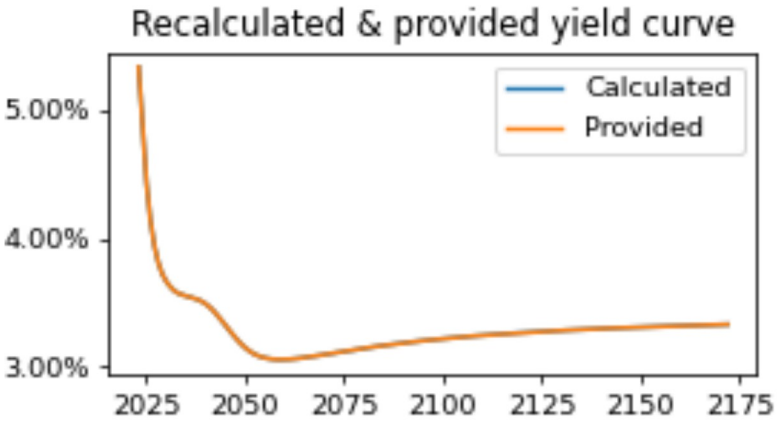
In [439…

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

In [439…

```
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 [439…

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

EIOPA curve comparison

Absolute difference in bps

In [439…

```
test_statistics_bdp.head()
```

Out[439…

	Abs diff in bps
0	3.319144e-07
1	4.631384e-02
2	4.666250e-02
3	4.426631e-02
4	4.770124e-02

Back to the top

Test 1; Success criteria

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

In [439…

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

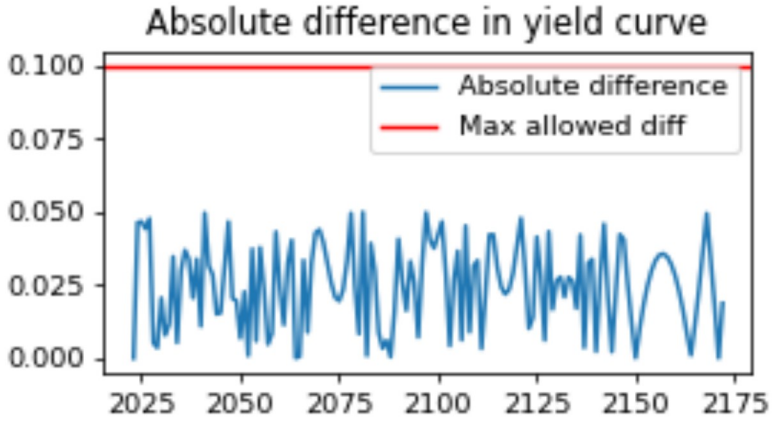
```
Test passed
Test passed
```

In [439…

```
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 February 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_20230228_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230228_Term_Structures.xlsx*.

In [440...

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

Out[440...

	Mean test	Max test
Provided vs calculated	True	True

Final yield curve

Full yield curve provided by EIOPA in %

In [440...

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

Out[440...

```
Country
1      5.338
2      4.900
3      4.432
4      4.139
5      3.941
6      3.808
7      3.721
8      3.664
9      3.623
10     3.592
11     3.572
12     3.560
13     3.551
14     3.543
15     3.534
16     3.523
17     3.508
18     3.488
19     3.463
20     3.433
21     3.398
22     3.359
23     3.319
24     3.279
25     3.241
26     3.206
27     3.174
28     3.146
29     3.122
30     3.102
31     3.087
32     3.075
33     3.067
34     3.061
35     3.057
36     3.055
37     3.055
38     3.055
39     3.057
40     3.059
```

41	3.062
42	3.066
43	3.070
44	3.074
45	3.079
46	3.084
47	3.089
48	3.094
49	3.099
50	3.104
51	3.109
52	3.114
53	3.119
54	3.124
55	3.129
56	3.133
57	3.138
58	3.143
59	3.147
60	3.152
61	3.156
62	3.161
63	3.165
64	3.169
65	3.173
66	3.177
67	3.181
68	3.185
69	3.188
70	3.192
71	3.195
72	3.199
73	3.202
74	3.205
75	3.208
76	3.212
77	3.215
78	3.218
79	3.220
80	3.223
81	3.226
82	3.229
83	3.231
84	3.234
85	3.236
86	3.239
87	3.241
88	3.244
89	3.246
90	3.248
91	3.250
92	3.253
93	3.255
94	3.257
95	3.259
96	3.261
97	3.263
98	3.265
99	3.266

100	3.268
101	3.270
102	3.272
103	3.274
104	3.275
105	3.277
106	3.279
107	3.280
108	3.282
109	3.283
110	3.285
111	3.286
112	3.288
113	3.289
114	3.291
115	3.292
116	3.293
117	3.295
118	3.296
119	3.297
120	3.299
121	3.300
122	3.301
123	3.302
124	3.303
125	3.305
126	3.306
127	3.307
128	3.308
129	3.309
130	3.310
131	3.311
132	3.312
133	3.313
134	3.314
135	3.315
136	3.316
137	3.317
138	3.318
139	3.319
140	3.320
141	3.321
142	3.322
143	3.323
144	3.324
145	3.325
146	3.325
147	3.326
148	3.327
149	3.328
150	3.329