# EIOPA RISK-FREE CURVE AUGUST-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

1. Note on Smith & Wilson algorithm
2. Data requirements
3. Success criteria
4. External dependencies
5. Calibration parameters and calibration vector provided by EIOPA
6. Smith & Wilson calculation functions
7. Generation of the risk-free curve
8. Test 1; Comparison test
9. Test 1; Success criteria
10. Test 1; Comparison test
11. Conclusion

# 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 August 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_20230831_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_20230831_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 [63]:
```
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 [64]:
```
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 [65]:
```
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 [66]:
```
#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 [67]:
```
param_raw = pd.read_csv(selected_param_file, sep=',',index_col=0)
```

### Parameter input

Parameters sheet

In [68]:
```
param_raw.head()
```

Out[68]:

| 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** | 0.11312 | 0.11312 | 0.11312 | 0.11312 | 0.11312 | |

5 rows × 106 columns

---

The country selected is:

In [69]:
```
country = "United Kingdom"
```

In [70]:
```
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 [71]:
```
extra_param
```

Out[71]:
```
Country
Coupon_freq     1.000000
LLP            50.000000
Convergence    40.000000
UFR             3.450000
alpha           0.096251
CRA             0.000000
Name: United Kingdom_Values, dtype: float64
```

---

In [72]:
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [73]:
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

---

### Maturity vector

Vector of maturities used in the calibration

In [74]:
```
maturities_country.head(15)
```

Out[74]:
```
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 [75]:
```
Qb = param_country_raw.iloc[relevant_positions]
```

---

### Calibration vector

Vector **Qb** provided as input

In [76]:
```
Qb
```

Out[76]:
```
Country
1    -5.328489
2    -5.639309
3     3.222237
4     0.227246
5    -0.419723
6     0.751781
7    -0.415237
8     0.034069
9     1.866519
10   -0.947211
11   -0.006315
12    0.091852
13   -0.009938
14   -0.009606
15   -0.232935
16    0.000123
```

```
17    0.000119
18    0.000115
19    0.000111
20    0.120593
21   -0.004708
22   -0.004551
23   -0.004400
24   -0.004253
25   -0.004111
26   -0.003974
27   -0.003841
28   -0.003713
29   -0.003589
30   -0.275425
31    0.007125
32    0.006887
33    0.006657
34    0.006435
35    0.006221
36    0.006013
37    0.005813
38    0.005619
39    0.005431
40    0.005250
41    0.005075
42    0.004906
43    0.004742
44    0.004584
45    0.004431
46    0.004284
47    0.004141
48    0.004003
49    0.003869
50    0.104605
Name: United Kingdom Values, dtype: float64
```

---

In [77]:
```
curve_raw = pd.read_csv(selected_curves_file, sep=',',index_col=0)
```

In [78]:
```
curve_country = curve_raw.loc[:,country]
```

Back to the top

# Calibration parameters and calibration vector provided by EIOPA

In [79]:

```
# 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 [80]:

```
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 [81]:

```
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 [82]:
```
r_Target.head(15)
```

Out[82]:

| | Recalculated rates |
|---|---|
| **0** | 0.057535 |
| **1** | 0.055002 |
| **2** | 0.051974 |
| **3** | 0.049468 |
| **4** | 0.047457 |
| **5** | 0.045832 |
| **6** | 0.044564 |
| **7** | 0.043594 |
| **8** | 0.042905 |
| **9** | 0.042463 |
| **10** | 0.042145 |
| **11** | 0.041900 |
| **12** | 0.041701 |
| **13** | 0.041526 |
| **14** | 0.041356 |

---

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_20230831_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 [83]:
```
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 [84]:
```
target_curve = pd.DataFrame(target_curve,columns=['Given rates'])
```

---

**EIOPA curve provided**

Yield curve provided by EIOPA

In [85]:
```
target_curve.head()
```

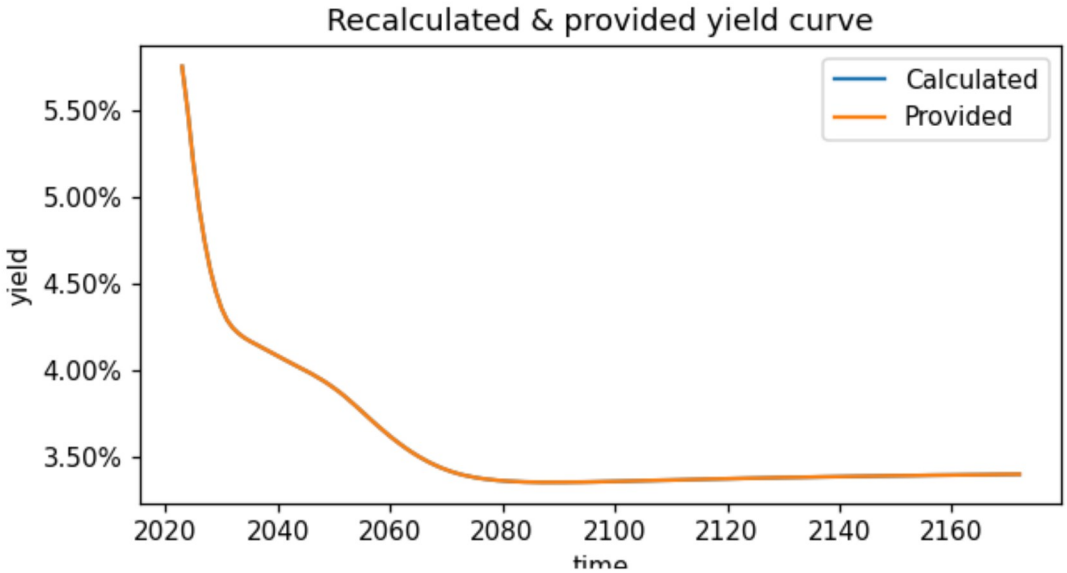
Out[85]:

| | Given rates |
|---|---|
| **0** | 0.05754 |
| **1** | 0.05500 |
| **2** | 0.05197 |
| **3** | 0.04947 |
| **4** | 0.04746 |

In [86]:
```
x_data_label = range(2023,2023+r_Target.shape[0],1)
```

In [87]:
```
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 [88]:
```
test_statistics_bdp = pd.DataFrame(abs(r_Target.values-target_curve.values)*10000, c
```

**EIOPA curve comparison**

Absolute difference in bps

In [89]:
```
test_statistics_bdp.head()
```

Out[89]:

| | Abs diff in bps |
|---|---|
| **0** | 0.050002 |
| **1** | 0.022908 |
| **2** | 0.042952 |
| **3** | 0.016622 |

**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.

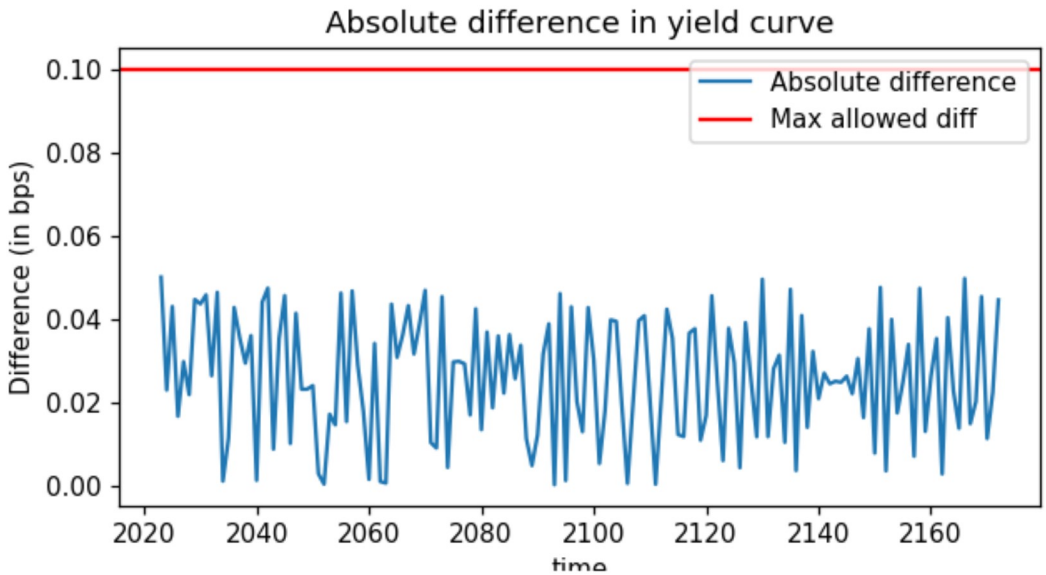
In [90]:
```
result1 = SuccessTest(test_statistics_bdp.values, test_statistics_max_diff_in_bps, te
```

```
Test passed
Test passed
```

In [91]:
```
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 August 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_20230831_Qb_SW.xlsx* and the parameters displayed in the file *EIOPA_RFR_20230831_Term_Structures.xlsx*.

In [92]:
```
pd.DataFrame(data = [result1], columns = ["Mean test","Max test"],  \
             index= ["Provided vs calculated"])
```

Out[92]:

| | Mean test | Max test |
|---|---|---|
| **Provided vs calculated** | True | True |

---

**Final yield curve**

Full yield curve provided by EIOPA in %

In [93]:
```
(curve_country*100).head(150)
```

Out[93]:
```
Country
1      5.754
2      5.500
3      5.197
4      4.947
5      4.746
6      4.583
7      4.456
8      4.359
9      4.291
10     4.246
11     4.215
12     4.190
13     4.170
14     4.153
15     4.136
16     4.118
17     4.101
18     4.083
19     4.066
20     4.049
21     4.032
22     4.015
23     3.998
24     3.981
25     3.962
26     3.943
27     3.922
28     3.899
```

| | |
|---|---|
| 29 | 3.875 |
| 30 | 3.849 |
| 31 | 3.821 |
| 32 | 3.792 |
| 33 | 3.763 |
| 34 | 3.733 |
| 35 | 3.703 |
| 36 | 3.675 |
| 37 | 3.647 |
| 38 | 3.620 |
| 39 | 3.594 |
| 40 | 3.570 |
| 41 | 3.547 |
| 42 | 3.525 |
| 43 | 3.505 |
| 44 | 3.487 |
| 45 | 3.469 |
| 46 | 3.454 |
| 47 | 3.439 |
| 48 | 3.427 |
| 49 | 3.415 |
| 50 | 3.405 |
| 51 | 3.396 |
| 52 | 3.389 |
| 53 | 3.383 |
| 54 | 3.377 |
| 55 | 3.373 |
| 56 | 3.369 |
| 57 | 3.366 |
| 58 | 3.363 |
| 59 | 3.361 |
| 60 | 3.359 |
| 61 | 3.357 |
| 62 | 3.356 |
| 63 | 3.355 |
| 64 | 3.355 |
| 65 | 3.354 |
| 66 | 3.354 |
| 67 | 3.354 |
| 68 | 3.354 |
| 69 | 3.354 |
| 70 | 3.355 |
| 71 | 3.355 |
| 72 | 3.355 |
| 73 | 3.356 |
| 74 | 3.357 |
| 75 | 3.357 |
| 76 | 3.358 |
| 77 | 3.359 |
| 78 | 3.359 |
| 79 | 3.360 |
| 80 | 3.361 |
| 81 | 3.362 |
| 82 | 3.362 |
| 83 | 3.363 |
| 84 | 3.364 |
| 85 | 3.365 |
| 86 | 3.366 |
| 87 | 3.366 |

| | |
|---|---|
| 88 | 3.367 |
| 89 | 3.368 |
| 90 | 3.369 |
| 91 | 3.370 |
| 92 | 3.370 |
| 93 | 3.371 |
| 94 | 3.372 |
| 95 | 3.373 |
| 96 | 3.373 |
| 97 | 3.374 |
| 98 | 3.375 |
| 99 | 3.376 |
| 100 | 3.376 |
| 101 | 3.377 |
| 102 | 3.378 |
| 103 | 3.378 |
| 104 | 3.379 |
| 105 | 3.380 |
| 106 | 3.380 |
| 107 | 3.381 |
| 108 | 3.382 |
| 109 | 3.382 |
| 110 | 3.383 |
| 111 | 3.383 |
| 112 | 3.384 |
| 113 | 3.384 |
| 114 | 3.385 |
| 115 | 3.386 |
| 116 | 3.386 |
| 117 | 3.387 |
| 118 | 3.387 |
| 119 | 3.388 |
| 120 | 3.388 |
| 121 | 3.389 |
| 122 | 3.389 |
| 123 | 3.390 |
| 124 | 3.390 |
| 125 | 3.391 |
| 126 | 3.391 |
| 127 | 3.392 |
| 128 | 3.392 |
| 129 | 3.393 |
| 130 | 3.393 |
| 131 | 3.393 |
| 132 | 3.394 |
| 133 | 3.394 |
| 134 | 3.395 |
| 135 | 3.395 |
| 136 | 3.395 |
| 137 | 3.396 |
| 138 | 3.396 |
| 139 | 3.397 |
| 140 | 3.397 |
| 141 | 3.397 |
| 142 | 3.398 |
| 143 | 3.398 |
| 144 | 3.398 |
| 145 | 3.399 |
| 146 | 3.399 |

```
147    3.400
148    3.400
149    3.400
150    3.401
```