# EIOPA RISK-FREE CURVE MONTHLY YIELD CALCULATION

January 8, 2026

## 1 Summary

The European Insurance and Occupational Pensions Authority (EIOPA) provide every month a yield curve for each relevant country. The curve is provided as a series of annual yield rates starting with 1 year and going all the way to 150. However, many use cases require a different granularity. This workbook uses the same Smith-Wilson algorithm together with EIOPA's own calibration vectors to generate the yield curve for any granularity. This avoids the need to implement a separate interpolation/extrapolation method.

## 2 Table of Contents



## 3 Note on Smith & Wilson algorithm

To replicate the calculations, this example uses a modified Smith&Wilson implementation (The original implementation is available on GitHub at `https://github.com/open-source-modelling`):

- Python
- Matlab
- JavaScript

## 4 Limitations of the implementation

- This demonstration assumes each month is exactly $\frac{1}{12}$ of a year
- Requires from the user to manually insert the calibration vector and other parameters

- Coupon frequency is assumed to be 1

## 5 Data requirements

This script contains the EIOPA risk-free rate publication for December 2025. 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_20251231_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_20251231_Term_Structures.xlsx*, sheet *RFR_spot_no_VA*.

## 6 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)

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
%matplotlib notebook
%matplotlib inline
pd.options.display.max_rows = 150
```

## 7 Importing data

Besides the annual yield curve, EIOPA also provides the calibration vectors and other parameters that were used in the generation of the yield curve. The calibration vector, the maturities used and the extra parameters. These values together with the Smith-Wilson algorithm uniquely define a yield curve for all maturities. In this section each of these values is imported. For convenience, it is defined within the Jupyter Notebook.

### 7.1 Importing calibration vector Qb and maturities

The calibration vectors are in the Excel file "EIOPA_RFR_20251231_Qb_SW.xlsx.

The Qb vector is saved as a dictionary with keys equal to maturities and values equal to the corresponding value in the vector *Qb*.

[3]:
```
Qb_dict = {1: -0.08092804958528270,
 2: 8.8549954718833000,
 3: -5.38088673718341000,
4:  1.64325748798146000,
5: -3.48269039442415000,
6: 7.75561965822494000,
7: -7.12934593220290000,
8: -1.38222739592224000,
```

```
9: 11.02344154124140000,
10:-10.73840154775390000,
11: -2.80633661206670000,
12:18.14824939350060000,
13:-14.77350180170510000,
14:0.07403954303455470,
15:2.53881584166336000,
16:-0.00329218388790419,
17:-0.00318701247618992,
18:-0.00308520084819934,
19:-0.00298664167299065,
20:-0.09517437973634960,
}
```

This vector contains two elements necessary to calibrate the SW algorithm. The maturities and the values. Each is saved in a separate numpy array.

[4]:
```
Qb = np.array(list(Qb_dict.values()))
```

---

Calibration vector
Vector Qb provided as input

[5]:
```
display(Qb)
```

```
array([-8.09280496e-02,  8.85499455e+00, -5.38088674e+00,  1.64325749e+00,
       -3.48269039e+00,  7.75561966e+00, -7.12934593e+00, -1.38222740e+00,
        1.10234415e+01, -1.07384015e+01, -2.80633661e+00,  1.81482494e+01,
       -1.47735018e+01,  7.40395430e-02,  2.53881584e+00, -3.29218389e-03,
       -3.18701248e-03, -3.08520085e-03, -2.98664167e-03, -9.51743797e-02])
```

[6]:
```
M_Obs = np.transpose(np.array(list(Qb_dict.keys())))
```

---

Maturities used in the calibration
Vector M_Obs provided as input

[7]:
```
display(M_Obs)
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20])
```

### 7.2 Importing extra parameters

The ekstra parameters used in the calibration of the SW algorithm are porvided in the header of the Excel file “EIOPA_RFR_20251231_Term_Structures.xlsx. For example in sheet”RFR_spot_no_VA”.

### 7.2.1 Ultimate forward rate

[8]:
```
ufr = 3.3/100 # Ultimate forward rate ufr represents the rate to which the rate␣
 ↪curve will converge as time increases:
```

---

Ultimate forward rate
Parameter ufr provided as input

[9]:
```
print(ufr)
```

```
0.033
```

### 7.2.2 Speed of convergence $\alpha$

[10]:
```
# Convergence speed parameter alpha controls the speed at which the curve␣
 ↪converges towards the ufr from the last liquid point:
alpha = 0.073632
```

---

Speed of convergence
Parameter $\alpha$ provided as input

[11]:
```
print(alpha)
```

```
0.073632
```

## 8 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`.

[12]:
```
def SWExtrapolate(M_Target, M_Obs, Qb, ufr, alpha):
# SWEXTRAPOLATE Interpolate or/and extrapolate rates for targeted maturities␣
 ↪using a Smith-Wilson algorithm.
# out = SWExtrapolate(M_Target, M_Obs, Qb, ufr, alpha) calculates the rates for␣
 ↪maturities specified in M_Target using the calibration vector b.
#
# Arguments:
#    M_Target = k x 1 ndarray. Each element represents a bond maturity of␣
 ↪interest. Ex. M_Target = [[1], [2], [3], [5]]
#    M_Obs =    n x 1 ndarray. Observed bond maturities used for calibrating␣
 ↪the calibration vector b. Ex. M_Obs = [[1], [3]]
#    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 alpha. Ex. alpha = 0.05
#
#
# Returns:
#    k x 1 ndarray. Represents the targeted rates for a zero-coupon bond. Each␣
 ↪rate belongs to a targeted zero-coupon bond with a maturity from T_Target.␣
 ↪Ex. r = [0.0024; 0.0029; 0.0034; 0.0039]
#
# For more information see https://www.eiopa.europa.eu/sites/default/files/
 ↪risk_free_interest_rate/12092019-technical_documentation.pdf

    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 alpha. Ex. alpha = 0.05
    #
    # Returns:
    #    n_1 x n_2 matrix representing the Heart of the Wilson function for␣
 ↪selected maturities and parameter alpha. H is calculated as in the paragraph␣
 ↪132 of the EIOPA documentation.
    #
    # For more information see https://www.eiopa.europa.eu/sites/default/files/
 ↪risk_free_interest_rate/12092019-technical_documentation.pdf

        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)) - alpha * np.absolute(u_Mat-v_Mat) - np.exp(-alpha * np.
 ↪absolute(u_Mat-v_Mat))); # Heart of the Wilson function from paragraph 132

    H = SWHeart(M_Target, M_Obs, alpha) # Heart of the Wilson function from␣
 ↪paragraph 132
    p = np.exp(-np.log(1+ufr)* M_Target) + np.diag(np.exp(-np.log(1+ufr) *␣
 ↪M_Target)) @ H @ Qb # Discount pricing function for targeted maturities from␣
 ↪paragraph 147
    return p ** (-1/ M_Target) -1 # Convert obtained prices to rates and return␣
 ↪prices
```

## 9 Maturities of interest

Vector `M_Target` contains the maturities of interest expressed as fractions of a year. So for example, for a monthly granularity, the maturities are $\frac{1}{12}, \frac{2}{12}, \ldots, \frac{12}{12}, \frac{13}{12}, \ldots$

[13]:
```
# For which maturities do we want the SW algorithm to calculate the rates. In this case, for every year up to 150:
M_Target = np.transpose(np.arange(1,601)/12)
```

---

Relevant maturities
Vector of maturities as fractions of a year

[14]:
```
display(M_Target)
```

```
array([ 0.08333333,  0.16666667,  0.25      ,  0.33333333,  0.41666667,
        0.5       ,  0.58333333,  0.66666667,  0.75      ,  0.83333333,
        0.91666667,  1.        ,  1.08333333,  1.16666667,  1.25      ,
        1.33333333,  1.41666667,  1.5       ,  1.58333333,  1.66666667,
        1.75      ,  1.83333333,  1.91666667,  2.        ,  2.08333333,
        2.16666667,  2.25      ,  2.33333333,  2.41666667,  2.5       ,
        2.58333333,  2.66666667,  2.75      ,  2.83333333,  2.91666667,
        3.        ,  3.08333333,  3.16666667,  3.25      ,  3.33333333,
        3.41666667,  3.5       ,  3.58333333,  3.66666667,  3.75      ,
        3.83333333,  3.91666667,  4.        ,  4.08333333,  4.16666667,
        4.25      ,  4.33333333,  4.41666667,  4.5       ,  4.58333333,
        4.66666667,  4.75      ,  4.83333333,  4.91666667,  5.        ,
        5.08333333,  5.16666667,  5.25      ,  5.33333333,  5.41666667,
        5.5       ,  5.58333333,  5.66666667,  5.75      ,  5.83333333,
        5.91666667,  6.        ,  6.08333333,  6.16666667,  6.25      ,
        6.33333333,  6.41666667,  6.5       ,  6.58333333,  6.66666667,
        6.75      ,  6.83333333,  6.91666667,  7.        ,  7.08333333,
        7.16666667,  7.25      ,  7.33333333,  7.41666667,  7.5       ,
        7.58333333,  7.66666667,  7.75      ,  7.83333333,  7.91666667,
        8.        ,  8.08333333,  8.16666667,  8.25      ,  8.33333333,
        8.41666667,  8.5       ,  8.58333333,  8.66666667,  8.75      ,
        8.83333333,  8.91666667,  9.        ,  9.08333333,  9.16666667,
        9.25      ,  9.33333333,  9.41666667,  9.5       ,  9.58333333,
        9.66666667,  9.75      ,  9.83333333,  9.91666667, 10.        ,
       10.08333333, 10.16666667, 10.25      , 10.33333333, 10.41666667,
       10.5       , 10.58333333, 10.66666667, 10.75      , 10.83333333,
       10.91666667, 11.        , 11.08333333, 11.16666667, 11.25      ,
       11.33333333, 11.41666667, 11.5       , 11.58333333, 11.66666667,
       11.75      , 11.83333333, 11.91666667, 12.        , 12.08333333,
       12.16666667, 12.25      , 12.33333333, 12.41666667, 12.5       ,
       12.58333333, 12.66666667, 12.75      , 12.83333333, 12.91666667,
       13.        , 13.08333333, 13.16666667, 13.25      , 13.33333333,
       13.41666667, 13.5       , 13.58333333, 13.66666667, 13.75      ,
```

```
13.83333333, 13.91666667, 14.        , 14.08333333, 14.16666667,
14.25      , 14.33333333, 14.41666667, 14.5       , 14.58333333,
14.66666667, 14.75      , 14.83333333, 14.91666667, 15.        ,
15.08333333, 15.16666667, 15.25      , 15.33333333, 15.41666667,
15.5       , 15.58333333, 15.66666667, 15.75      , 15.83333333,
15.91666667, 16.        , 16.08333333, 16.16666667, 16.25      ,
16.33333333, 16.41666667, 16.5       , 16.58333333, 16.66666667,
16.75      , 16.83333333, 16.91666667, 17.        , 17.08333333,
17.16666667, 17.25      , 17.33333333, 17.41666667, 17.5       ,
17.58333333, 17.66666667, 17.75      , 17.83333333, 17.91666667,
18.        , 18.08333333, 18.16666667, 18.25      , 18.33333333,
18.41666667, 18.5       , 18.58333333, 18.66666667, 18.75      ,
18.83333333, 18.91666667, 19.        , 19.08333333, 19.16666667,
19.25      , 19.33333333, 19.41666667, 19.5       , 19.58333333,
19.66666667, 19.75      , 19.83333333, 19.91666667, 20.        ,
20.08333333, 20.16666667, 20.25      , 20.33333333, 20.41666667,
20.5       , 20.58333333, 20.66666667, 20.75      , 20.83333333,
20.91666667, 21.        , 21.08333333, 21.16666667, 21.25      ,
21.33333333, 21.41666667, 21.5       , 21.58333333, 21.66666667,
21.75      , 21.83333333, 21.91666667, 22.        , 22.08333333,
22.16666667, 22.25      , 22.33333333, 22.41666667, 22.5       ,
22.58333333, 22.66666667, 22.75      , 22.83333333, 22.91666667,
23.        , 23.08333333, 23.16666667, 23.25      , 23.33333333,
23.41666667, 23.5       , 23.58333333, 23.66666667, 23.75      ,
23.83333333, 23.91666667, 24.        , 24.08333333, 24.16666667,
24.25      , 24.33333333, 24.41666667, 24.5       , 24.58333333,
24.66666667, 24.75      , 24.83333333, 24.91666667, 25.        ,
25.08333333, 25.16666667, 25.25      , 25.33333333, 25.41666667,
25.5       , 25.58333333, 25.66666667, 25.75      , 25.83333333,
25.91666667, 26.        , 26.08333333, 26.16666667, 26.25      ,
26.33333333, 26.41666667, 26.5       , 26.58333333, 26.66666667,
26.75      , 26.83333333, 26.91666667, 27.        , 27.08333333,
27.16666667, 27.25      , 27.33333333, 27.41666667, 27.5       ,
27.58333333, 27.66666667, 27.75      , 27.83333333, 27.91666667,
28.        , 28.08333333, 28.16666667, 28.25      , 28.33333333,
28.41666667, 28.5       , 28.58333333, 28.66666667, 28.75      ,
28.83333333, 28.91666667, 29.        , 29.08333333, 29.16666667,
29.25      , 29.33333333, 29.41666667, 29.5       , 29.58333333,
29.66666667, 29.75      , 29.83333333, 29.91666667, 30.        ,
30.08333333, 30.16666667, 30.25      , 30.33333333, 30.41666667,
30.5       , 30.58333333, 30.66666667, 30.75      , 30.83333333,
30.91666667, 31.        , 31.08333333, 31.16666667, 31.25      ,
31.33333333, 31.41666667, 31.5       , 31.58333333, 31.66666667,
31.75      , 31.83333333, 31.91666667, 32.        , 32.08333333,
32.16666667, 32.25      , 32.33333333, 32.41666667, 32.5       ,
32.58333333, 32.66666667, 32.75      , 32.83333333, 32.91666667,
33.        , 33.08333333, 33.16666667, 33.25      , 33.33333333,
33.41666667, 33.5       , 33.58333333, 33.66666667, 33.75      ,
```

```
33.83333333, 33.91666667, 34.        , 34.08333333, 34.16666667,
34.25      , 34.33333333, 34.41666667, 34.5       , 34.58333333,
34.66666667, 34.75      , 34.83333333, 34.91666667, 35.        ,
35.08333333, 35.16666667, 35.25      , 35.33333333, 35.41666667,
35.5       , 35.58333333, 35.66666667, 35.75      , 35.83333333,
35.91666667, 36.        , 36.08333333, 36.16666667, 36.25      ,
36.33333333, 36.41666667, 36.5       , 36.58333333, 36.66666667,
36.75      , 36.83333333, 36.91666667, 37.        , 37.08333333,
37.16666667, 37.25      , 37.33333333, 37.41666667, 37.5       ,
37.58333333, 37.66666667, 37.75      , 37.83333333, 37.91666667,
38.        , 38.08333333, 38.16666667, 38.25      , 38.33333333,
38.41666667, 38.5       , 38.58333333, 38.66666667, 38.75      ,
38.83333333, 38.91666667, 39.        , 39.08333333, 39.16666667,
39.25      , 39.33333333, 39.41666667, 39.5       , 39.58333333,
39.66666667, 39.75      , 39.83333333, 39.91666667, 40.        ,
40.08333333, 40.16666667, 40.25      , 40.33333333, 40.41666667,
40.5       , 40.58333333, 40.66666667, 40.75      , 40.83333333,
40.91666667, 41.        , 41.08333333, 41.16666667, 41.25      ,
41.33333333, 41.41666667, 41.5       , 41.58333333, 41.66666667,
41.75      , 41.83333333, 41.91666667, 42.        , 42.08333333,
42.16666667, 42.25      , 42.33333333, 42.41666667, 42.5       ,
42.58333333, 42.66666667, 42.75      , 42.83333333, 42.91666667,
43.        , 43.08333333, 43.16666667, 43.25      , 43.33333333,
43.41666667, 43.5       , 43.58333333, 43.66666667, 43.75      ,
43.83333333, 43.91666667, 44.        , 44.08333333, 44.16666667,
44.25      , 44.33333333, 44.41666667, 44.5       , 44.58333333,
44.66666667, 44.75      , 44.83333333, 44.91666667, 45.        ,
45.08333333, 45.16666667, 45.25      , 45.33333333, 45.41666667,
45.5       , 45.58333333, 45.66666667, 45.75      , 45.83333333,
45.91666667, 46.        , 46.08333333, 46.16666667, 46.25      ,
46.33333333, 46.41666667, 46.5       , 46.58333333, 46.66666667,
46.75      , 46.83333333, 46.91666667, 47.        , 47.08333333,
47.16666667, 47.25      , 47.33333333, 47.41666667, 47.5       ,
47.58333333, 47.66666667, 47.75      , 47.83333333, 47.91666667,
48.        , 48.08333333, 48.16666667, 48.25      , 48.33333333,
48.41666667, 48.5       , 48.58333333, 48.66666667, 48.75      ,
48.83333333, 48.91666667, 49.        , 49.08333333, 49.16666667,
49.25      , 49.33333333, 49.41666667, 49.5       , 49.58333333,
49.66666667, 49.75      , 49.83333333, 49.91666667, 50.        ])
```

## 10 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.

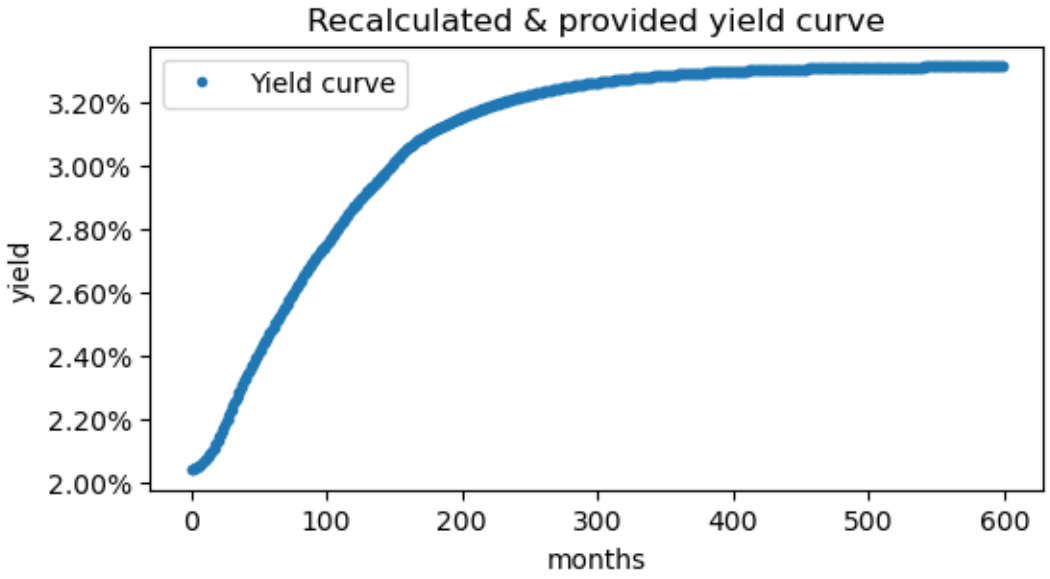
[15]:

```
r_Target = SWExtrapolate(M_Target,M_Obs, Qb, ufr, alpha)
r_Target = pd.DataFrame(r_Target,columns=['Recalculated rates'], index =␣
↪M_Target)
```

[16]:
```
x_data_label = range(0,r_Target.shape[0],1)
```

[17]:
```
fig, ax1 = plt.subplots(1,1)
ax1.plot(x_data_label, r_Target.values*100, color='tab:blue',label="Yield␣
↪curve", marker='.', linestyle='')

ax1.set_ylabel("yield")
ax1.set_title('Recalculated & provided yield curve')
ax1.set_xlabel("months")
ax1.legend()
ax1.yaxis.set_major_formatter(mtick.PercentFormatter())
fig.set_figwidth(6)
fig.set_figheight(3)
plt.show()
```




---

## 11 Conclusion

This notebook calculated the Italian monthly yield curve using the EIOPA RFR calibration.

Note: As a check, the user can compare every 12-th value with the yield reported in the Excel file "EIOPA_RFR_20251231_Term_Structures.xlsx" or similar checks.

Final monthly yield curve
Full yield curve provided by EIOPA but in monthly granularity

[18]:
```
display(r_Target)
```

```
           Recalculated rates
0.083333             0.020424
0.166667             0.020436
0.250000             0.020452
0.333333             0.020472
0.416667             0.020495
…                       …
49.666667            0.033104
49.750000            0.033104
49.833333            0.033104
49.916667            0.033104
50.000000            0.033104

[600 rows x 1 columns]
```