Projection of the risk free curve and recalibration inside OSEM

The purpose of this workbook is to showcase the calibration methodology inside OSEM. The key to the calibration is the Smith Wilson algorithm commonly used in insurance. This algorithm allows a continious approximation of arbitrary maturities based on a subset of available maturities. This example is split onto 3 parts.

- The first part shows how the necessary parameters and curves are imported from input files.
- Second part shows how the curves are generated and the Smith-Wilson calibrations derived.
- The last part shows how the saved calibration can be used to derive an the discount rates for any maturity.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
```

In [2]:
```
from ImportData import import_SWEiopa
from CurvesClass import Curves
```

Importing files

The parameters and the current risk free curve are provided as input:

- Parameters.csv; Parameters related to the run
- EIOPA_param_file.csv; Assumed yield curve at modelling date and relevant maturities
- EIOPA_curves_file.csv; Parameters related to the EIOPA time 0 calibration

The exact location of the EIOPA files is encoded in the Parameters input file.

In [3]:
```
paramfile = pd.read_csv("Input\Parameters.csv", index_col="Parameter")
```

The location of the two EIOPA files are:

In [4]:
```
selected_param_file = paramfile["Value"].loc["EIOPA_param_file"]
selected_curves_file = paramfile["Value"].loc["EIOPA_curves_file"]
```

The risk free curve belongs to the following country:

In [5]:
```
country = paramfile["Value"].loc["country"]
```

Import all necessary parameters:

In [6]:
```
[maturities_country, curve_country, extra_param, Qb] = import_SWEiopa(selected_param_
```

Preparation and treatment of the curves

The OSEM does multiple steps in order to make it easier for OSEM to access discount curves at any point in time and for arbitrary maturities. All steps are performed and saved within the Curves class.

The curves class

The curve class object contains all the data necessary to run the model.

In [7]:
```
# ultimate forward rate
ufr = extra_param["UFR"]/100

# Numeric precision of the optimisation
precision = float(paramfile["Value"].loc["Precision"])

# Targeted distance between the extrapolated curve and the ultimate forward rate at t
tau = float(paramfile["Value"].loc["Tau"])# 1 basis point

modelling_date = paramfile.loc["Modelling_Date"]

# Number of projection years
n_years = int(paramfile.loc["n_proj_years"][0])
```

In [8]:
```
curves = Curves(ufr, precision, tau, modelling_date, country)
```

After the Curves class is initiated, there are a series if calibrations that needs to be performed.

- Saving of the input term structure
- Calculation of the forward rates
- Projection of the forward spot rates
- Calculation of the Smith-Wilson calibrations for each period

Save the input term structure

SetObservedTermStructure sets the liquid maturities and the coresponding yields to the m_obs and r_obs property of the Curves class.

In [9]:
```
curves.SetObservedTermStructure(maturity_vec=curve_country.index.tolist(), yield_vec=
```

Calculate 1 year forward rates

Using the time 0 spot curve provided as input, the yearly forward rates can be calculated from the annually reported yield curve

$$fw_{EIOPA,1} = y_1$$

otherwise:

$$fw_{EIOPA,t} = \frac{(1+y_t)^{-t}}{(1+y_{t-1})^{-(t-1)}}$$

The forward rates will be used to calculate forward spot curves

In [10]:
```
curves.CalcFwdRates()
```

Forward yield curve

The forward yield curve can be calculated by using the 1-year forward rates from the suitable moment onwards. The spot yield curve is calculated for each projection period. For the projection period i, the yield curve can be calculated as:

$$y^i_{t-i} = \prod_i^t \left(1 + fw_{EIOPA}(t)\right)^{\frac{1}{t-i}}$$

In [11]:
```
curves.ProjectForwardRate(n_years)
```

Calibrate Smith-Wilson on each yield curve

Each forward spot curve is used to calibrate the SW algorithm. This calibration can then be used to generate yield corves of arbitrary maturities. The calibration is split on two parts. First one calibrates the curve at time 0

Calibration at time 0

In [12]:
```
NameOfYear = "Yield_year_0"
NameOfYear_2 = "Maturities_year_0"
NameOfYear_3 = "Calibration_year_0"
NameOfYear_4 = "Alpha_year_0"

r_obs = np.transpose(np.array(curves.r_obs[NameOfYear])) # Obtain the yield curve
m_obs = np.transpose(np.array(curves.m_obs[NameOfYear_2])) # Obtain the maturities cu

# Calculate the calibration parameter alpha
alphaoptimized = [curves.BisectionAlpha(0.05, 0.5, m_obs, r_obs, curves.ufr, curves.t

# Save the calibration parameter alpha
curves.alpha[NameOfYear_4] = alphaoptimized

b_calibrated = curves.SWCalibrate(r_obs, m_obs, curves.ufr, curves.alpha[NameOfYear_4
curves.b[NameOfYear_3] = b_calibrated
```

Calibration at all future times

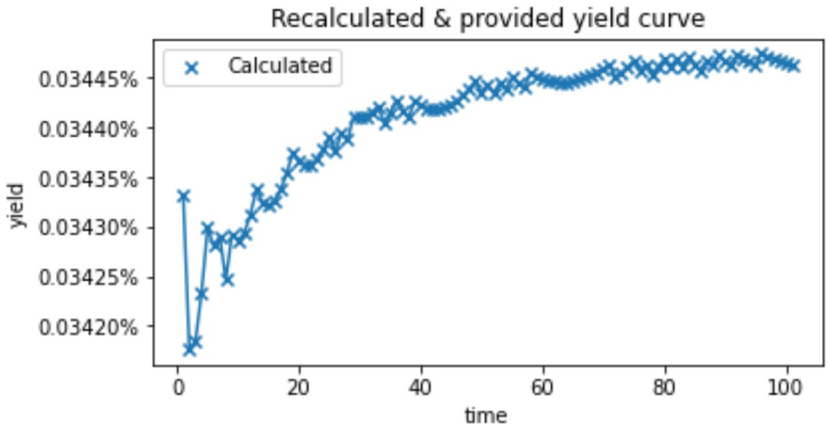
In [13]:
```
for iYear in range(1, n_years):
    ProjYear = iYear
    NameOfYear = "Yield_year_" + str(ProjYear)
    NameOfYear_2 = "Maturities_year_" + str(ProjYear)
    NameOfYear_3 = "Calibration_year_" + str(ProjYear)
    NameOfYear_4 = "Alpha_year_"+str(ProjYear)

    r_obs = np.transpose(np.array(curves.r_obs[NameOfYear]))[:-ProjYear] # Obtain the
    m_obs = np.transpose(np.array(curves.m_obs[NameOfYear_2]))[:-ProjYear]
    alphaoptimized = [curves.BisectionAlpha(0.05, 0.5, m_obs, r_obs, curves.ufr, curv
    if NameOfYear_4 in curves.alpha.columns:
        curves.alpha[NameOfYear_4] = alphaoptimized
    else:
        curves.alpha = curves.alpha.join(pd.Series(data=None, index=None, name = Name
        curves.alpha[NameOfYear_4] = alphaoptimized

    b_calibrated = curves.SWCalibrate(r_obs, m_obs, curves.ufr, curves.alpha[NameOfYe
    b_calibrated = np.append(b_calibrated, np.repeat(np.nan, ProjYear))

    if NameOfYear_3 in curves.b.columns:
        curves.b[NameOfYear_3] = b_calibrated
    else:
        curves.b = curves.b.join(pd.Series(data= None,index=None, name=NameOfYear_3,
        curves.b[NameOfYear_3] = b_calibrated
```

Example 1

Is the yield curve equal to the curve used for the calibration?

In [14]:
```
r_obs_est = curves.SWExtrapolate(m_obs, m_obs, curves.b[NameOfYear_3][:-(ProjYear)],
```

In [15]:
```
fig, ax1 = plt.subplots(1, 1)
ax1.scatter(m_obs, r_obs, label="Calculated", marker="x")
ax1.plot(m_obs, r_obs_est)
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()
```



Example 2

The future yield curve for arbitrary maturities.

Assuming the algorithm is processing the year 3. A hypothetical asset has a cash flow at year fractions: $\{0.7, 1.2, 2.1, 3.543\}$

To make it possible to calculate the present value, the coresponding dicount rates are calculated:

In [16]:
```
ModellingYear = 3
maturity_name = "Maturities_year_" + str(ModellingYear)
calibration_name = "Calibration_year_" + str(ModellingYear)
alpha_name = "Alpha_year_" + str(ModellingYear)

# The maturities for which we are looking the discount yields
desired_mat = np.array([0.7, 1.2, 2.1, 3.543])
```

In [17]:
```
calib_b = curves.b[calibration_name][:-ModellingYear].values
```

In [18]:
```
calib_maturities = curves.m_obs[maturity_name][:-ModellingYear].values
```

In [19]:
```
calib_alpha = curves.alpha[alpha_name][0]
```

In [20]:
```
result = curves.SWExtrapolate(desired_mat, calib_maturities, calib_b, curves.ufr, ca
```

The required yields are:

In [21]:
```
display(result)
```

```
array([0.02618662, 0.026155  , 0.02616982, 0.02642073])
```