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.

In [20]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
```

In [21]:

```
from ImportData import import_SWEiopa
from CurvesClass import Curves
```

Importing external 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

Read the necessary input files

In [22]:

```
paramfile = pd.read_csv("Input\Parameters.csv", index_col="Parameter")
```

The location of the two EIOPA files are:

In [23]:

```
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 [24]:

```
country = paramfile["Value"].loc["country"]
```

Import all necessary parameters:

In [25]:

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

The curve class

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

In [26]:
```
# 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 [27]:
```
curves = Curves(ufr, precision, tau, modelling_date, country)
```

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

In [28]:
```
curves.SetObservedTermStructure(maturity_vec=curve_country.index.tolist(), yield_vec=
```

Calculate 1 year forward rates

The forward rates will be used to calculate forward spot curves

In [29]:
```
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 formula used is:

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

In [30]:
```
curves.ProjectForwardRate(n_years)
```

Calibrate every yield curve

Repeat for all

In [31]:

```
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]))
alphaoptimized = [curves.BisectionAlpha(0.05, 0.5, M_Obs, r_Obs, curves.ufr, curves.t
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 = NameOfYe
    curves.alpha[NameOfYear_4] = alphaoptimized

bCalibrated = curves.SWCalibrate(r_Obs, M_Obs, curves.ufr, curves.alpha[NameOfYear_4]
curves.b[NameOfYear_3] = bCalibrated

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

    bCalibrated = curves.SWCalibrate(r_Obs, M_Obs, curves.ufr, curves.alpha[NameOfYea
    bCalibrated = np.append(bCalibrated, np.repeat(np.nan, ProjYear))

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

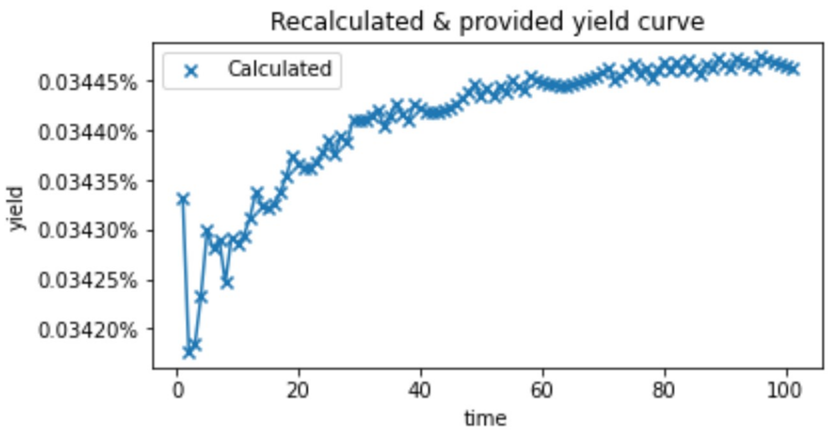
Examples

In [32]:

```
r_Obs_Est = curves.SWExtrapolate(M_Obs, M_Obs, curves.b[NameOfYear_3][:-(ProjYear)],
```

In [33]:

```
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

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 [34]:

```
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 [35]:

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

In [36]:

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

In [37]:

```
calib_alpha = curves.alpha[alpha_name][0]
```

In [38]:

```
result = curves.SWExtrapolate(desired_mat, calib_maturities, calib_b, curves.ufr, cal
```

The required yields are:

In [39]:

```
display(result)
```

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