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 [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 external files

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

- Parameters.csv; TBC
- EIOPA_param_file.csv; TBC
- EIOPA_curves_file.csv; TBC

Import Param file

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

Read base EIOPA curve from file

In [4]:
```
selected_param_file = paramfile.loc["EIOPA_param_file"][0]
selected_curves_file = paramfile.loc["EIOPA_curves_file"][0]
country = paramfile.loc["country"][0]

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

Initiate Curve object

In [5]:
```
# ultimate forward rate
ufr = extra_param["UFR"]/100
# Numeric precision of the optimisation
Precision = float(paramfile.loc["Precision"][0])

# Targeted distance between the extrapolated curve and the ultimate forward rate at t
Tau = float(paramfile.loc["Tau"][0])# 1 basis point

MD = paramfile.loc["Modelling_Date"]
```

In [6]:
```
Curves = Curves(ufr, Precision, Tau, MD, country)
```

In [8]:
```
Curves.SetObservedTermStructure(maturity_vec=curve_country.index.tolist(), yield_vec=
```

In [9]:
```
ProjYear = 0
# Number of projection years
N = int(paramfile.loc["n_proj_years"][0])
```

Calculate 1 year forward rates

The forward rates will be used to calculate forward spot curves

In [11]:
```
Curves.CalcFwdRates()
```

In [12]:
```
display(Curves.fwd_rates)
```

	Forward
0	NaN
1	1.031582
2	1.027909
3	1.026170
4	1.026146
...	...
145	1.034191
146	1.034211
147	1.034231
148	1.034251
149	1.034271

150 rows × 1 columns

Forward spot rates

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

In [13]:
```
Curves.ProjectForwardRate(N)
```

In [15]:
```
display(Curves.r_obs)
```

	Yield	Yield year1	Yield year2	Yield year3	Yield year4	Yield year5	Yield year6	Yield year7	Yield year8	Yield year9
0	0.03472	0.031582	0.027909	0.026170	0.026146	0.026663	0.027041	0.027560	0.028290	0.02895(
1	0.03315	0.029745	0.027039	0.026158	0.026404	0.026852	0.027301	0.027925	0.028620	0.02801(
2	0.03140	0.028552	0.026741	0.026326	0.026617	0.027088	0.027631	0.028267	0.028104	0.02945:
3	0.03009	0.027951	0.026722	0.026505	0.026853	0.027389	0.027960	0.027968	0.029161	0.02988:
4	0.02930	0.027693	0.026786	0.026716	0.027140	0.027701	0.027782	0.028841	0.029563	0.02954:
...	...	...	...	...	...	...	...	...	...	...
145	0.03274	0.032739	0.032757	0.032801	0.032856	NaN	NaN	NaN	NaN	NaN
146	0.03275	0.032749	0.032768	0.032811	NaN	NaN	NaN	NaN	NaN	NaN
147	0.03276	0.032760	0.032778	NaN	NaN	NaN	NaN	NaN	NaN	NaN
148	0.03277	0.032770	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
149	0.03278	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

150 rows × 50 columns

Calculate the alpha parameter

Calibration of the alpha parameter for the base curve using the bisection algorithm.

In [16]:
```
alphaoptimized = [Curves.BisectionAlpha(0.05, 0.5, Curves.m_obs["Maturity"], Curves.r
if "Yield year" in Curves.alpha.columns:
    Curves.alpha["Yield year"] = alphaoptimized
else:
    Curves.alpha = Curves.alpha.join(pd.Series(data=None, index=None, name="Yield yea
    Curves.alpha["Yield year"] = alphaoptimized
```

In [17]:
```
Curves.alpha
```

Out[17]:

	Yield year
0	0.5

Calibrate first calibration vector b

In [18]:
```
bCalibrated = Curves.SWCalibrate(Curves.r_obs["Yield"], Curves.m_obs["Maturity"], Cur
bCalibrated = np.append(bCalibrated,np.repeat(np.nan, ProjYear))
```

In [19]:
```
if "Yield year" in Curves.b.columns:
    Curves.b["Yield year"] = bCalibrated
else:
    Curves.b = Curves.b.join(pd.Series(data= None,index=None, name="Yield year",dtype
    Curves.b["Yield year"] = bCalibrated
```

Calibrate every yield curve

Parameters that stay the same

Calibrate time 0 curve

Calculated implied observations for the first year of projection

In [20]:
```
ProjYear = 1
NameOfYear = "Yield year"+str(ProjYear)
```

In [21]:
```
r_obs = np.transpose(np.array(Curves.r_obs[NameOfYear]))[:-ProjYear]
```

Calculated implied maturities for the first year of projection

In [22]:
```
m_obs = np.transpose(np.array(range(1,r_obs.size+1)))
```

Calibrate

In [23]:
```
alphaoptimized = [Curves.BisectionAlpha(0.05, 0.5, m_obs, r_obs, Curves.ufr, Curves.t
```

In [24]:
```
if NameOfYear in Curves.alpha.columns:
    Curves.alpha[NameOfYear] = alphaoptimized
else:
    Curves.alpha = Curves.alpha.join(pd.Series(data=None, index=None, name=NameOfYear
    Curves.alpha[NameOfYear] = alphaoptimized
```

In [25]:
```
bCalibrated = Curves.SWCalibrate(r_obs, m_obs, Curves.ufr, Curves.alpha[NameOfYear][0
bCalibrated = np.append(bCalibrated,np.repeat(np.nan,ProjYear))
```

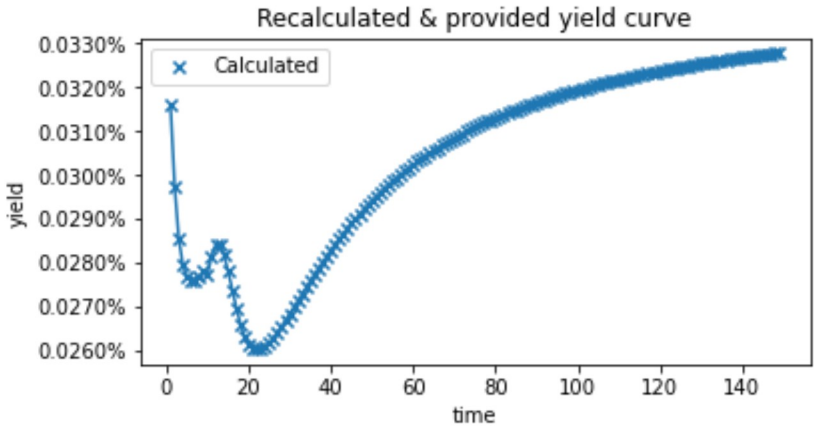
In [26]:
```
if NameOfYear in Curves.b.columns:
    Curves.b[NameOfYear] = bCalibrated
else:
    Curves.b = Curves.b.join(pd.Series(data= None,index=None, name=NameOfYear,dtype=
    Curves.b[NameOfYear] = bCalibrated
```

In [27]:
```
Curves.b[NameOfYear][:-1]
```

Out[27]:
```
0     -0.080601
1      0.015151
2      0.007984
3      0.010251
4     -0.007252
         ...
144    6.988021
145   -1.911537
146    0.574049
147   -0.314029
148    0.266142
Name: Yield year1, Length: 149, dtype: float64
```

In [28]:
```
r_Obs_Est = Curves.SWExtrapolate(m_obs, m_obs, Curves.b[NameOfYear][:-1], Curves.ufr
```

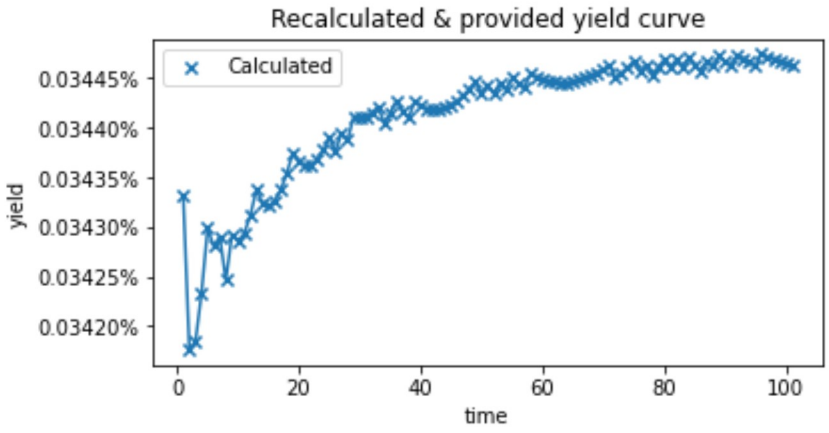
In [29]:
```
fig, ax1 = plt.subplots(1,1)
ax1.scatter(m_obs, r_obs, label="Calculated",marker="x")
ax1.plot(m_obs, r_Obs_Est)
ax1.set_ylabel("yield")
ax1.set_title('Recalculated & provided yield curve')
ax1.set_xlabel("time")
ax1.legend()
ax1.yaxis.set_major_formatter(mtick.PercentFormatter())
fig.set_figwidth(6)
fig.set_figheight(3)
plt.show()
```



Repeat for all

In [30]:
```
for iYear in range(2,50):
    ProjYear = iYear
    NameOfYear = "Yield year"+str(ProjYear)
    r_Obs = np.transpose(np.array(Curves.r_obs[NameOfYear]))[:-ProjYear]
    M_Obs = np.transpose(np.array(range(1,r_Obs.size+1)))
    alphaoptimized = [Curves.BisectionAlpha(0.05, 0.5, M_Obs, r_Obs, Curves.ufr, Curv
    if NameOfYear in Curves.alpha.columns:
        Curves.alpha[NameOfYear] = alphaoptimized
    else:
        Curves.alpha = Curves.alpha.join(pd.Series(data=None, index=None, name=NameO
        Curves.alpha[NameOfYear] = alphaoptimized
    bCalibrated = Curves.SWCalibrate(r_Obs, M_Obs, Curves.ufr, Curves.alpha[NameOfYea
    bCalibrated = np.append(bCalibrated,np.repeat(np.nan,ProjYear))
    if NameOfYear in Curves.b.columns:
        Curves.b[NameOfYear] = bCalibrated
    else:
        Curves.b = Curves.b.join(pd.Series(data= None,index=None, name=NameOfYear,dty
        Curves.b[NameOfYear] = bCalibrated
    r_Obs_Est = Curves.SWExtrapolate(M_Obs, M_Obs, Curves.b[NameOfYear][:-(ProjYear)]
```

In [31]:
```
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()
```



Saving calibrated results

In [32]:
```
Curves.b.to_csv("Intermediate/b.csv")
```

In [33]:
```
Curves.alpha.to_csv("Intermediate/alpha.csv")
```

In [34]:
```
Curves.r_obs
```

Out[34]:

	Yield	Yield year1	Yield year2	Yield year3	Yield year4	Yield year5	Yield year6	Yield year7	Yield year8	Yield year9
0	0.03472	0.031582	0.027909	0.026170	0.026146	0.026663	0.027041	0.027560	0.028290	0.02895(
1	0.03315	0.029745	0.027039	0.026158	0.026404	0.026852	0.027301	0.027925	0.028620	0.02801(
2	0.03140	0.028552	0.026741	0.026326	0.026617	0.027088	0.027631	0.028267	0.028104	0.02945:
3	0.03009	0.027951	0.026722	0.026505	0.026853	0.027389	0.027960	0.027968	0.029161	0.02988:
4	0.02930	0.027693	0.026786	0.026716	0.027140	0.027701	0.027782	0.028841	0.029563	0.02954:
...	...	...	...	...	...	...	...	...	...	...
145	0.03274	0.032739	0.032757	0.032801	0.032856	NaN	NaN	NaN	NaN	NaN
146	0.03275	0.032749	0.032768	0.032811	NaN	NaN	NaN	NaN	NaN	NaN
147	0.03276	0.032760	0.032778	NaN	NaN	NaN	NaN	NaN	NaN	NaN
148	0.03277	0.032770	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
149	0.03278	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

150 rows × 50 columns

In [35]:
```
Curves.m_obs.to_csv("Intermediate/M_Obs.csv")
```

In [36]:
```
Curves.fwd_rates.to_csv("Intermediate/FwdRates.csv")
```

In [37]:
```
Qb.to_csv("Intermediate/Qb_0.csv")
```