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 [29]:

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

In [30]:

```
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 [31]:

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

The location of the two EIOPA files are:

In [32]:

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

In [33]:

```
selected_param_file
```

Out[33]:

```
'Input/Param_no_VA.csv'
```

In [34]:

```
selected_curves_file
```

Out[34]:

```
'Input/Curves_no_VA.csv'
```

The risk free curve belongs to the following country:

In [35]:
```
country = paramfile["Value"].loc["country"]
```

In [36]:
```
country
```

Out[36]:
```
'Slovenia'
```

Import all necessary parameters:

In [37]:
```
[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 [38]:
```
# 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 
tau = float(paramfile["Value"].loc["Tau"])# 1 basis point

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

In [39]:
```
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 [40]:
```
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 [41]:
```
curves.CalcFwdRates()
```

In [42]:
```
display(curves.fwd_rates)
```

	Forward
0	NaN
1	1.031582
2	1.027909

	Forward
3	1.026170
4	1.026146
...	...
145	1.034191
146	1.034211
147	1.034231
148	1.034251
149	1.034271

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:

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

Calculate term structure calibration for the modelling date

In [43]:
```
ProjYear = 0
# Number of projection years
N = int(paramfile.loc["n_proj_years"][0])
```

In [44]:
```
curves.ProjectForwardRate(N)
```

In [45]:
```
display(curves.r_obs)
```

	Yield	Yield year1	Yield year2	Yield year3	Yield year4	Yield year5	Yield year6	Yield year7	Yield year8	Yielc year!
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	Nal
146	0.03275	0.032749	0.032768	0.032811	NaN	NaN	NaN	NaN	NaN	Nal

	Yield	Yield year1	Yield year2	Yield year3	Yield year4	Yield year5	Yield year6	Yield year7	Yield year8	Yield year9
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

Calculate the alpha parameter

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

In [46]:

```
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
```

Calibrate first calibration vector b

In [47]:

```
bCalibrated = curves.SWCalibrate(curves.r_obs["Yield"], curves.m_obs["Maturity"], cur
bCalibrated = np.append(bCalibrated,np.repeat(np.nan, ProjYear))
```

In [48]:

```
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

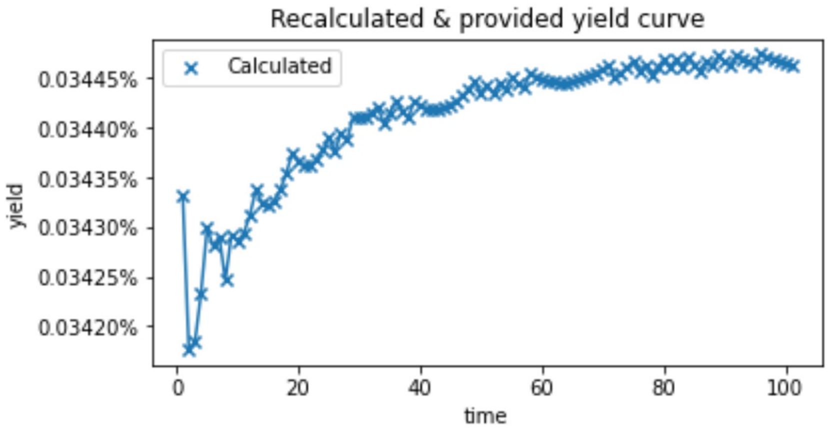
Repeat for all

In [49]:

```
for iYear in range(1,N):
    ProjYear = iYear
    NameOfYear = "Yield year"+str(ProjYear)
    r_Obs = np.transpose(np.array(curves.r_obs[NameOfYear]))[:-ProjYear] # Obtain the
    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=NameOf
        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 [50]:

```
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 [51]:

```
#curves.b.to_csv("Intermediate/b.csv")
```

In [52]:

```
#curves.alpha.to_csv("Intermediate/alpha.csv")
```

In [53]:
```
curves.r_obs
```

Out[53]:

	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 [54]:
```
#curves.m_obs.to_csv("Intermediate/M_Obs.csv")
```

In [55]:
```
#curves.fwd_rates.to_csv("Intermediate/FwdRates.csv")
```

In [56]:
```
#Qb.to_csv("Intermediate/Qb_0.csv")
```