PROTOTYPE BOND PRICING_v2

The purpose of this script is to showcase different functionalities of corporate bonds inside the OSEM model

In [ ]:
```
import os
import datetime as dt
import numpy as np
import pandas as pd
```

In [3]:
```
from CurvesClass import Curves
from ImportData import import_SWEiopa, get_corporate_bonds
from BondClasses import *
from ConfigurationClass import Configuration
from ImportData import get_configuration, get_settings
from MainLoop import create_cashflow_dataframe
```

Input files

There are multiple input files needed to calibrate the fixed income portfolio. They are located in the "Input" folder.

Parameters.csv

Parameters file holds information about the type of run and the modelling date.

- EIOPA_param_file ...the relative location of the EIOPA parameter file that will be used as the RFR Ex. "Input/Param_no_VA.csv"
- EIOPA_curves_file ... the relative location of the EIOPA yield curve that will be used as the RFR Ex. "Input/Curves_no_VA.csv"
- country ... the name of the country that will be used as the base for this run Ex. "Slovenia"
- n_proj_years ... length of a run in years starting from the Modelling date Ex. 50
- Precision ... precision parameter specifying the acceptable tollerance between the calibrated bond price and the market value Ex. 0.00000001
- Tau ... the acceptable size of the gap between the extrapolated yield rate and the ulitmate forward rate Ex. 0.0001
- compounding ... the way that the interest rates are compounded in the run Ex. -1
- Modelling_Date ... the starting date of the run specified as a date string Ex."29/04/2023"

EIOPA RFR files

There are two types of files derived from the monthly EIOPA RFR submision that are used in this model. The "Curves_XX.csv" containing the yearly yield curves for all countries in scope and the

"Param_XX.csv" with the paameters used to derive the curves. These files are used to derive the risk free term structure at the modelling date and to efficiently project the evolution of the term structure.

Portfolio description

The modelled portfolio is split by asset classes. The fixed income portfolio is located in the file "Bond_Portfolio.csv". Each security needs the following fields:

- Asset ID ... unique id such as an ISIN, SEDOL or CUSIP code Ex. IT1234567891
- Asset_Type ... asset type string Ex. "Corporate_Bond"
- NACE ... NACE asset classification code (nomenclature statistique des activités économiques dans la Communauté européenne) Ex. A1.4.5
- Issue_Date ... the string date specifying the issue date of the bond Ex. 3/12/2021
- Maturity_Date ... the string date specifying the maturity date of the bond Ex. 3/12/2021
- Notional_amount ... the notional amount of the bond Ex. 100
- Coupon_Rate ... percentage of the notional amount paid in dividends every period (specified by Frequency) Ex. 0.0014
- Frequency ... number of times per a year that dividends are paid Ex. 1 (once per a year)
- Recovery_Rate ... percentage of the notional amound that can be recovered in case of a default Ex. 0.80
- Default_Probability ... percentage probability of default per year Ex. 0.012
- Units ... number of each bond held in the portfolio Ex. 230
- Market_Price ... market price of the bond at the modelling date Ex. 96

Sector spread

The list of NACE sector codes and the sector specific spread over the risk free rate

- NACE ... NACE code of the issuer Ex. "A1.1"
- NACE code text ... description of the NACE code for this issuer Ex. "Growing of non-perennial crops"
- sSpread ... NACE sector specific spread over the risk free rate Ex. 0.01

In the POC, the spread is displayed directly in the bond input file

Set up the base folder

In [4]:
```
base_folder = os.getcwd()  # Get current working directory
```

Most of the run settings are saved in the configuration file:

In [5]:
```
conf: Configuration
conf = get_configuration(os.path.join(base_folder, "ALM.ini"), os)
```

These lines of code just extract the absolute location of different files:

In [6]:
```
parameters_file = conf.input_parameters
cash_portfolio_file = conf.input_cash_portfolio
bond_portfolio_file = conf.input_bond_portfolio
```

In [7]:
```
paramfile = pd.read_csv("Input/Parameters.csv")
paramfile.index = paramfile["Parameter"]
```

The parameter file is:

In [8]:
```
display(paramfile)
```

Parameter	Parameter	Value
EIOPA_param_file	EIOPA_param_file	Input/Param_no_VA.csv
EIOPA_curves_file	EIOPA_curves_file	Input/Curves_no_VA.csv
country	country	Slovenia
run_type	run_type	Risk Neutral
n_proj_years	n_proj_years	50
Precision	Precision	1E-10
Tau	Tau	0.0001
compounding	compounding	-1
Modelling_Date	Modelling_Date	29/04/2023

In [9]:
```
del paramfile["Parameter"]
```

The settings object holds data about file locations, information about the run settings and model parameters such as modelling date.

In [10]:
```
settings = get_settings(parameters_file)
```

The CorpBond object contains information about each equity position. This includes:

- asset_id
- nace
- issuer
- issue_date
- maturity_date
- coupon_rate
- bond specific spread
- notional_amount

- frequency
- recovery_rate
- default_probability
- units
- market_price

A Python generator reads the bond portfolio file and encodes it into a dictionary based on the asset id. Each asset id contains a CorpBond object describing a single fixed income position.

In [11]:
```
bond_input_generator = get_corporate_bonds(bond_portfolio_file)
bond_input = {corp_bond.asset_id: corp_bond for corp_bond in bond_input_generator}
```

The dictionary containing the bond portfolio is:

In [12]:
```
display(bond_input)
```

```
{1234: CorpBond(asset_id=1234, nace='A1.4.5', issuer=None, issue_date=datetime.date(2
021, 12, 3), maturity_date=datetime.date(2026, 12, 12), coupon_rate=0.03, notional_am
ount=100.0, zspread=0.01, frequency=1, recovery_rate=0.4, default_probability=0.03, u
nits=1.0, market_price=94.0),
 2889: CorpBond(asset_id=2889, nace='B5.2.0', issuer=None, issue_date=datetime.date(2
021, 12, 3), maturity_date=datetime.date(2028, 12, 12), coupon_rate=0.05, notional_am
ount=100.0, zspread=0.01, frequency=2, recovery_rate=0.4, default_probability=0.03, u
nits=2.0, market_price=92.0),
 31: CorpBond(asset_id=31, nace='B8.9.3', issuer=None, issue_date=datetime.date(2019,
12, 3), maturity_date=datetime.date(2025, 12, 3), coupon_rate=0.04, notional_amount=1
00.0, zspread=0.01, frequency=12, recovery_rate=0.4, default_probability=0.03, units=
3.0, market_price=96.0)}
```

CorpBondPortfolio class contains all CorpBond objects in a dictionary:

In [13]:
```
bond_portfolio = CorpBondPortfolio(bond_input)
```

In [14]:
```
bond_portfolio.corporate_bonds
```

Out[14]:
```
{1234: CorpBond(asset_id=1234, nace='A1.4.5', issuer=None, issue_date=datetime.date(2
021, 12, 3), maturity_date=datetime.date(2026, 12, 12), coupon_rate=0.03, notional_am
ount=100.0, zspread=0.01, frequency=1, recovery_rate=0.4, default_probability=0.03, u
nits=1.0, market_price=94.0),
 2889: CorpBond(asset_id=2889, nace='B5.2.0', issuer=None, issue_date=datetime.date(2
021, 12, 3), maturity_date=datetime.date(2028, 12, 12), coupon_rate=0.05, notional_am
ount=100.0, zspread=0.01, frequency=2, recovery_rate=0.4, default_probability=0.03, u
nits=2.0, market_price=92.0),
 31: CorpBond(asset_id=31, nace='B8.9.3', issuer=None, issue_date=datetime.date(2019,
12, 3), maturity_date=datetime.date(2025, 12, 3), coupon_rate=0.04, notional_amount=1
00.0, zspread=0.01, frequency=12, recovery_rate=0.4, default_probability=0.03, units=
3.0, market_price=96.0)}
```

Importing the information about the economic environment

import_SWEiopa() reads the necessary data about the current yield curve. One of these parameters (the ufr or ultimate forward rate) is necessary in the equity example as ufr is used in the Gordon growth formula to calculate the terminal value of the equity position. Inside OSEM, the parameters related to the yield curve are saved in the Curves object.

In [15]:
```
[maturities_country, curve_country, extra_param, Qb] = import_SWEiopa(settings.EIOPA_
                                                                      settings.EI
# Curves object with information about term structure
curves = Curves(extra_param["UFR"] / 100, settings.precision, settings.tau, settings.
                settings.country)
```

In [16]:
```
ufr = extra_param["UFR"]/100 # ultimate forward rate
precision = float(settings.precision) # Numeric precision of the optimisation
# Targeted distance between the extrapolated curve and the ufr at the convergence po
tau = float(settings.tau) # 1 basis point
```

In [17]:
```
curves.SetObservedTermStructure(maturity_vec=curve_country.index.tolist(), yield_vec=
curves.CalcFwdRates()
curves.ProjectForwardRate(settings.n_proj_years)
curves.CalibrateProjected(settings.n_proj_years, 0.05, 0.5, 1000)
```

In [18]:
```
spreadfile = pd.read_csv("Input/Sector_Spread.csv")
spreadfile.index = spreadfile["NACE"]
del spreadfile["NACE"]
```

Save the calibration parameters of the selected curve into the Curves instance:

Cash flow projection of a bond portfolio

The basis of OSEM is cash flow simulation. The cash flows for the coupon payment and the return of the notional are simulated separately.

A list of dictionaries containing all the dates and amounts of coupon payments are produced by calling the create_coupon_flows function:

In [19]:
```
dividend_flows = bond_portfolio.create_coupon_flows(settings.modelling_date, settings
```

The list of dictionaries containing the return of the notional amount is produced by calling the function create_maturity_flows:

In [20]:
```
terminal_flows = bond_portfolio.create_maturity_flows(terminal_date=settings.end_date
```

All cash flows can be represented in a matrix with all possible cash flow dates as columns and all equities as rows. The non-zero entries then represent the value of the cash flow at that date. The first step is to calculate the unique dates for the entire portfolio of bonds. This is done by calling the unique_dates_profiles() function over the dates related to coupons or notional amount

payments.

Both can then conveniently be represented as DataFrames.

Note that a vector of bond specific spreads is also provided as output.

In [21]:
```
unique_list = bond_portfolio.unique_dates_profile(dividend_flows)
```

In [22]:
```
unique_terminal_list = bond_portfolio.unique_dates_profile(terminal_flows)
```

Using the sorted list of unique dates as column headers, the dataframes containing the information related to the cash flows can be produced.

The first dataframe contains the market price of each position. Additionaly, the dataframe of zspreads is returned that helps to price the bonds using a discounted cash flow method. The last output is a dataframe containing the amount (units) of each bond in the portfolio is created.

In [23]:
```
[market_price_df, zspread_df, units_df] = bond_portfolio.init_bond_portfolio_to_data
```

A dataframe of cash flows and notional amount payments is created:

In [24]:
```
# Dataframe with bond  coupon cash flows
cash_flows = create_cashflow_dataframe(dividend_flows, unique_list)
# Dataframe with bond notional cash flows
notional_cash_flows = create_cashflow_dataframe(terminal_flows, unique_terminal_list)
```

Cash flow dataframe with coupon amounts and dates:

In [25]:
```
display(cash_flows)
```

	2023-05-03	2023-06-03	2023-07-03	2023-08-03	2023-09-03	2023-10-03	2023-11-03	2023-
1234	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2889	0.0	5.0	0.0	0.0	0.0	0.0	0.0	
31	4.0	4.0	4.0	4.0	4.0	4.0	4.0	

3 rows × 38 columns

Cash flow dataframe with notional amount payments and dates:

In [26]:
```
display(notional_cash_flows)
```

	2025-12-03	2026-12-12	2028-12-12
1234	0.0	100.0	0.0
2889	0.0	0.0	100.0

	2025-12-03	2026-12-12	2028-12-12
31	100.0	0.0	0.0

The extra spread due to the extra riskines of the bond compared to a risk free instrument:

In [27]:
```
display(zspread_df)
```

	2023-04-29
1234	0.01
2889	0.01
31	0.01

Calculation of present value of each instrument

The cashflows can be used to price the current market value of the bond, implied by the assumed economic parameters.

This pricing is done using the risk free rate as the discounting factor. In practice, the price of risk for an equity share is positive.

A calibration method needs to be used to calculate the spread implied by the market. This example will show the pricing using the risk free rate assumptions and the calibration that returns the spread such that the observed market price is preserved.

For simplicity, this example does the pricing at the modelling date by setting the projection year equal to 0.

In [28]:
```
proj_period = 0
```

The present value of the bond implied by the current yield strucute is:

In [29]:
```
market_price_df = bond_portfolio.price_bond_portfolio(cash_flows, notional_cash_flows
```

In [30]:
```
market_price_df
```

Out[30]:

	2023-04-29
1234	96.375799
2889	131.437251
31	210.297223

Calibrate the spread to match market price

To calibrate the spread implied by the market, OSEM uses a bisection method to obtain the spread such that when added on top of the risk free term structure, the discounted cashflows equal to the current market price.

In [31]:
```
calibrated_spread = bond_portfolio.corporate_bonds[1234].bisection_spread(x_start=-0
                                  , x_end=0.2
                                  , modelling_date=settings. modelling_date
                                  , end_date=settings.end_date
                                  , proj_period=proj_period
                                  , curves=curves
                                  , precision= 0.00000001
                                  , max_iter=100000)
```

The market value calculated using the discounted cash flow method using the calibrated zspread is:

In [32]:
```
bond_portfolio.corporate_bonds[1234].price_bond(cash_flows.loc[1234],notional_cash_f
```

Out[32]:
```
array([94.00000004])
```

In [33]:
```
zspread_df
```

Out[33]:

	2023-04-29
1234	0.01
2889	0.01
31	0.01

The function to calibrate the entire portfolio:

In [34]:
```
zspread_df=bond_portfolio.calibrate_bond_portfolio(zspread_df, settings, proj_period,
```

In [35]:
```
zspread_df
```

Out[35]:

	2023-04-29
1234	0.017604
2889	0.100214
31	0.200000