_PROTOTYPE EQUITY PRICING_v3

November 29, 2024

1 PROTOTYPE EQUITY PRICING_v3

One of the basic functions of OSEM is creating and manipulating cash flows from different classes of financial instruments. This page shows the processing of company shares held within a portfolio.

An ownership interest or a share represents ownership of a small piece of a legal entity. The most common of them being listed companies. In OSEM these shares provide two kinds of benefits. A periotic dividend and its intrinsic value (The share can be sold to another entity).

The enterprise and with it the equity share is assumed to grow at a constant rate every year. At periodic periods, the dividend is paid out. The size of the dividend determined by the value of the share. A % of the market value referred here as dividend yield.

This page has 4 sections.

1) The first section shows how to import the necessary information about equity positions, the overall economic environment as well as other parameters.
2) The next section imports the term structure which is needed later on.
3) The third section shows how to generate the cash flows for a set of equities.
4) The last section shows how a hypothetical equity can be calibrated if the user wishes to preform a risk-neutral run.

1.1 Importing and handling of equity data

The packages and imports needed in this script are the following:

[187]:
```
import datetime
import os
import pandas as pd

from ConfigurationClass import Configuration
from CurvesClass import Curves
from EquityClasses import *
from ImportData import get_EquityShare, get_settings, import_SWEiopa, \
    get_configuration
from MainLoop import create_cashflow_dataframe
```

Set up the base folder

[189]:
```
base_folder = os.getcwd()  # Get current working directory
```

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

[191]:
```
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:

[193]:
```
parameters_file = conf.input_parameters
cash_portfolio_file = conf.input_cash_portfolio
equity_portfolio_file = conf.input_equity_portfolio
```

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

[195]:
```
settings = get_settings(parameters_file)
```

The EquityShare object contains information about each equity position. This includes: * Asset_ID * NACE * Issuer * Issue_Date * Dividend_Yield * Frequency * Units * Market_Price * Growth_Rate

[197]:
```
equity_input_generator = get_EquityShare(equity_portfolio_file)
equity_input = {equity_share.asset_id: equity_share for equity_share in␣
 ↪equity_input_generator}
```

EquitySharePortfolio object contains all EquityShare objects in a dictionary.

[199]:
```
equity_portfolio = EquitySharePortfolio(equity_input)
```

1.2 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.

[202]:
```
[maturities_country, curve_country, extra_param, Qb] = import_SWEiopa(settings.
 ↪EIOPA_param_file,
                                                                        ␣
 ↪settings.EIOPA_curves_file, settings.country)
# Curves object with information about term structure
curves = Curves(extra_param["UFR"] / 100, settings.precision, settings.tau,␣
 ↪settings.modelling_date,
                settings.country)
```

[203]:
```
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 point
tau = float(settings.tau) # 1 basis point
```

[204]:
```
curves.SetObservedTermStructure(maturity_vec=curve_country.index.tolist(),␣
↪yield_vec=curve_country.values)
curves.CalcFwdRates()
```

[205]:
```
curves.ProjectForwardRate(settings.n_proj_years)
```

[206]:
```
curves.CalibrateProjected(settings.n_proj_years, 0.05, 0.5, 1000)
```

1.3 Projection of cash flows for an equity portfolio

The first computation step inside the OSEM equity preparation process is the identification of all the unique dates and dividend size amounts. The representation of assets in terms of individual cash flows on the time-line is one of the core principles of OSEM. This is done by two functions. One for dividend dates and another for terminal rates.

Both functions generate a list of dictionaries containing the date of a cash flow and the amount. Same is also true for the terminal amount calculation.

Calculation of the dividend amount: The dividend size is calculated using the dividend yield provided as input for each equity position. However the market value changes as time moves forward. To account for this, the growth rate and the time fraction between the modelling date and the date of the cash flow is used to calculate a future market value.

ToDo Formulas

The same logic is applied to the calculation of terminal rates.

[209]:
```
dividend_flows = equity_portfolio.create_dividend_flows(settings.
↪modelling_date, settings.end_date)
terminal_flows = equity_portfolio.create_terminal_flows(modelling_date=settings.
↪modelling_date,
                                                        ␣
↪terminal_date=settings.end_date,
                                                        ␣
↪terminal_rate=curves.ufr)
```

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 equities. This is done by the unique_dates_profiles() function.

The same logic can be applied to terminal dates.

Both can then conveniently be represented as DataFrames.

Note that a vector of growth rates is also provided as output. This makes it simpler to increase the market value of the portfolio as OSEM moves from one modelling period to the next one.

[211]:
```
unique_list = equity_portfolio.unique_dates_profile(dividend_flows)
unique_terminal_list = equity_portfolio.unique_dates_profile(terminal_flows)
```

[212]:
```
[market_price_df, growth_rate_df, units_df] = equity_portfolio.
↪init_equity_portfolio_to_dataframe(settings.modelling_date)
```

The create_cashflow_dataframe() function converts the list of dictionaries of cashflows and dates, into a single DataFrame:

[214]:
```
cash_flows = create_cashflow_dataframe(dividend_flows, unique_list)
# Dataframe with terminal cash flows
terminal_cash_flows = create_cashflow_dataframe(terminal_flows,␣
↪unique_terminal_list)
```

[215]:
```
display(cash_flows)
```

```
      2023-12-03  2024-12-03  2025-12-03  2026-12-03  2027-12-03  2028-12-03  \
1125    2.836786    2.865193    2.893805    2.922704    2.951890    2.981450
2123    4.654653    4.747875    4.842701    4.939422    5.038074    5.138974
3232    3.930888    4.088343    4.251648    4.421477    4.598089    4.782269

      2029-12-03  2030-12-03  2031-12-03  2032-12-03  …  2063-12-03  \
1125    3.011223    3.041294    3.071665    3.102424  …    4.222438
2123    5.241612    5.346299    5.453077    5.562290  …   10.272092
3232    4.973293    5.171948    5.378537    5.593979  …   18.852033

      2064-12-03  2065-12-03  2066-12-03  2067-12-03  2068-12-03  2069-12-03  \
1125    4.264720    4.307309    4.350323    4.393766    4.437764    4.482081
2123   10.477818   10.687085   10.900531   11.118241   11.340913   11.567418
3232   19.607166   20.390359   21.204835   22.051846   22.935150   23.851276

      2070-12-03  2071-12-03  2072-12-03
1125    4.526840    4.572046    4.617830
2123   11.798446   12.034089   12.275104
3232   24.803996   25.794772   26.828002

[3 rows x 50 columns]
```

[216]:
```
display(terminal_cash_flows)
```

```
      2073-04-29
1125  154.544895
2123  247.465156
3232  681.363655
```

Risk spreads In OSEM, the cash flows of equity shares increase with the assumed growth rate. However in real world applications, when calculating the present value of these cash flows, the "riskiness" of the issuer should be accounted for. This is done using discounting spreads. These spreads represent the fact that certain geographies and certain sectors are more prone to failure and the required risk premium is therefore higher.

Each equity position has 3 kinds of spreads are available:

1) Country specific spread: Country specific spread represents the extra riskiness related to the country of operation (spread_country)
2) Sector specific spread: Spread specific to the sector in which the issuer opperates (spread_sector)
3) Assumed shock scenario specific spread: Spread specific to the shock scenario (spread_stress)

[ ]:

[ ]:

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.

Note that this pricing is done using the risk free rate as the discounting factor with a risk spread equal to the sum of the 3 components. This combined spread is added to the yield in a parralel shift. To demonstrate, the discount factors are calculated as:

$$df_i(t) = \frac{1}{(y(t) + spreadCountry + spreadSector + spreadShock)^t}$$

Where: - t is the time period - $df_i(t)$ is the discount factor at time t for the equity i - $y(t)$ is the yield implied by the risk-free interest rate at time t - $spreadCountry$, $spreadSector$, $spreadShock$ are the country, sector and shock spreads.

This example will show pricing at the modelling date.

[219]:
```
proj_period = 0
```

[220]:
```
asset_id_tmp = cash_flows.index[0]
```

[221]:
```
equity_portfolio.equity_share[asset_id_tmp]
```

[221]:
```
EquityShare(asset_id=1125, nace='A1.4.5', issuer=None,
issue_date=datetime.date(2021, 12, 3), dividend_yield=0.03, frequency=1,
units=1.0, market_price=94.0, growth_rate=0.01, spread_country=0.0,
spread_sector=0.0, spread_stress=0.0)
```

[222]:
```
asset_id_tmp = cash_flows.index[0]

temp_dividend = cash_flows.loc[asset_id_tmp].to_dict()

temp_terminal = terminal_cash_flows.loc[asset_id_tmp].to_dict()

price = equity_portfolio.equity_share[asset_id_tmp].price_share(temp_dividend,␣
 ↪temp_terminal, settings.modelling_date, proj_period, curves)
print(price)
```

```
[110.30416346]
```

Additivity of spreads The 3 kinds of spread work in the same way and increasing any of them has the same effect.

We will demonstrate this by fixing them one at a time.

[224]:
```
asset_id_tmp = cash_flows.index[0]

temp_dividend = cash_flows.loc[asset_id_tmp].to_dict()

temp_terminal = terminal_cash_flows.loc[asset_id_tmp].to_dict()
```

[225]:
```
equity_portfolio.equity_share[asset_id_tmp].spread_country = 0.01
equity_portfolio.equity_share[asset_id_tmp].spread_sector = 0.01
equity_portfolio.equity_share[asset_id_tmp].spread_stress = 0.00
```

[226]:
```
price = equity_portfolio.equity_share[asset_id_tmp].price_share(temp_dividend,␣
 ↪temp_terminal, settings.modelling_date, proj_period, curves)
print(price)
```

```
[68.45708747]
```

[227]:
```
equity_portfolio.equity_share[asset_id_tmp].spread_country = 0.01
equity_portfolio.equity_share[asset_id_tmp].spread_sector = 0
equity_portfolio.equity_share[asset_id_tmp].spread_stress = 0.01
```

[228]:
```
price2 = equity_portfolio.equity_share[asset_id_tmp].price_share(temp_dividend,␣
 ↪temp_terminal, settings.modelling_date, proj_period, curves)
print(price2)
```

```
[68.45708747]
```

[229]:
```
equity_portfolio.equity_share[asset_id_tmp].spread_country = 0
equity_portfolio.equity_share[asset_id_tmp].spread_sector = 0
equity_portfolio.equity_share[asset_id_tmp].spread_stress = 0.02
```

[230]:
```
price3 = equity_portfolio.equity_share[asset_id_tmp].price_share(temp_dividend,␣
 ↪temp_terminal, settings.modelling_date, proj_period, curves)
print(price3)
```

```
[68.45708747]
```

1.4 Calibration of an equity share

In a real world run, the growth rate of each equity is brough in as a modelling assumption and must be calculated externaly. If the user is interested in a risk-neutral run, the performance of each asset must equal to that of the risk free rate. In OSEM this is done using the growth rate as the calibrating parameter. To do this, a bisection algorithm is used to calibrate the growth rate such that the discounted cash flows of each equity equal to the current market price.

For this demonstration, a single equity position is created, the growth rate is calibrated and then the present value of the calibrated equity is compared to the assumed current market value.

Note that the projection period is selected as 0 meaning the initial modelling date. But the calibration works also for other projection periods. The only change needed would be to change the market price.

[233]:
```
proj_period = 0
```

[325]:
```
spread_country = 0
spread_sector = 0.02
spread_stress = 0
```

[327]:
```
test_share_1 = EquityShare(asset_id=1, nace='A1.4.5', issuer=None,␣
↪issue_date=datetime.date(2021, 12, 3), dividend_yield=0.03, frequency=1,␣
↪units=1,market_price=94.0, growth_rate=0.
↪01,spread_country=spread_country,spread_sector=spread_sector,spread_stress=spread_stress␣
↪)
```

[329]:
```
opt_growth = test_share_1.bisection_growth(-1, 1, settings.modelling_date,␣
↪settings.end_date, proj_period, curves, 0.00000001,100000)
```

[331]:
```
print(opt_growth)
```

```
0.023442290723323822
```

[333]:
```
test_dividends = test_share_1.create_single_cash_flows(settings.modelling_date,␣
↪settings.end_date, opt_growth)
test_terminal = test_share_1.create_single_terminal(settings.modelling_date,␣
↪settings.end_date, curves.ufr, opt_growth)
```

If the calibration was performed correctly, the present value calculated by discounting future cash flows with the assumed risk free rate should be equal to the initial observed market price.

[335]:
```
print(test_share_1.market_price)
```

```
94.0
```

[337]:
```
print(test_share_1.price_share(test_dividends, test_terminal, settings.
↪modelling_date, proj_period, curves))
```

```
[94.00001498]
```