ORE+ AMC Module 1.8.4.0

Quaternion Risk Management

24 June 2024

Document Change History

ORE+ Release	Date	Author	Comment
na	25 February 2019	Peter Caspers	initial version
na	23 October 2023	Peter Caspers	add parameter RegressorModel
na	4 April 2024	Peter Caspers	add parameter RegressionVarianceCutoff
na	18 June 2024	Henning Segger	add bond engine
na	19 September 2024	Peter Caspers	add parameters RecalibrateOnStickyCloseOutDates, ReevaluateExerciseInStickyRun

Contents

1 Summary

This document describes the American Monte Carlo (AMC) module in ORE+.

2 Overview

The exposure analysis implemented in ORE[1] is divided into two independent steps:

1. in a first step a list of NPVs (or a "NPV cube") is computed. The list is indexed by the trade ID, the simulation time step and the scenario sample number. Each entry of the cube is computed using the same pricers as for the T0 NPV calculation by shifting the evaluation date to the relevenat time step of the simulation and updating the market termstructures to the relevant scenario market data. The market data scenarios are generated using a *risk factor evolution model* which can be a cross asset model, but also be based on e.g. historical simulation.

2. in a scecond step the generated NPV cube is passed to a post processor that aggregates the results to XVA figures of different kinds.

The AMC module allows to replace the first step by a different approach which works faster in particular for exotic deals. The second step remains the same. The risk factor evolution model coincides with the pricing models for the single trades in this approach and is always a cross asset model operated in a pricing measure.
For AMC the entries of the NPV cube are now viewed as conditional NPVs at the simulation time given the information that is generated by the cross asset model's driving stochastic process up to the simulation time. The conditional expectations are then computed using a regression analysis of some type. In our current implementation this is chosen to be a parametric regression analysis.
The regression models are calibrated per trade during a traning phase and later on evaluated in the simulation phase. The set of paths in the two phases is in general different w.r.t. their number, time step structure, and generation method (Sobol, Mersenne Twister) and seed. Typically the regressand is the (deflated) dirty *path* NPV of the trade in question, or also its underlying NPV or an option continuation value (to take exercise decisions or represent the physical underlying for physical exercise rights). The regressor is typically the model state. Certain exotic features that introduce path-dependency (e.g. a TaRN structure) may require an augmentation of the regressor though (e.g. by the already accumulated amount in case of the TaRN).
The path NPVs are generated at their *natural event dates*, like the fixing date for floating rate coupons or the payment date for fixed cashflows. This reduces the requirements for the cross asset model to provide closed form expressions for the numeraire and conditional zero bonds only.
Since the evaluation of the regression functions is computationally cheap the overall timings of the NPV cube generation are generally smaller compared to the classic approach, in particular for exotic deals like Bermudan swaptions.
From a methodology point of view an important difference between the classic and the AMC exposure analysis lies in the model consistency: While the conditional NPVs computed with AMC are by construction consistent with the risk factor evolution model

[1] also referred to as the *classic* exposure analysis in what follows

driving the XVA simulation, the scenario NPVs in the classic approach are in general not consistent in this sense unless the market scenarios are fully implied by the cross asset model. Here "fully implied" means that not only rate curves, but also market volatility and correlation term structures like FX volatility surfaces, swaption volatilties or CMS correlation term structures as well as other parmeters used by the single trade pricers have to be deduced from the cross asset model, e.g. the mean reversion of the Hull White 1F model and a sutaible model volatility feeding into a Bermudan swaption pricer.
We note that the generation of such implied term structures can be computationally expensive even for simple versions of a cross asset model like one composed from LGM IR and Black-Scholes FX components etc., and even more so for more exotic component flavours like Cheyette IR components, Heston FX components etc.
In the current implementation only a subset of all trade types can be simulated using AMC while all other trade types are still simulated using the classic engine. The separation of the trades and the join of the resulting classic and AMC cubes is automatic. The post processing step is run on the joint cube from the classic and AMC simulations as before.

3 Configuration

3.1 Application

To use the AMC engine for an XVA simulation the following parameters can be added to the simulation setup:

```
<Analytic type="simulation">
  ...
  <Parameter name="amc">Y</Parameter>
  <Parameter name="amcPathDataInput">amcpathdata.dat</Parameter>
  <Parameter name="amcPathDataOutput">amcpathdata.dat</Parameter>
  <Parameter name="amcIndividualTrainingOutput">Y</Parameter>
  <Parameter name="amcIndividualTrainingInput">N</Parameter>
  <Parameter name="amcTradeTypes">Swap,Swaption,CapFloor,ForwardRateAgreement,FxOption,BermudanSwapt
  <Parameter name="amcPricingEnginesFile">pricingengines_amc.xml</Parameter>
</Analytic>
```

The trades which have a trade type matching one of the types in the `amcTradeTypes` list, will be built against the pricing engine config provided and processed in the AMC engine. As a naming convention, pricing engines with engine type AMC provide the required functionality to be processed by the AMC engine, for technical details cf. **??**.
All other trades are processed by the classic simulation engine in ORE. The resulting cubes from the classic and AMC simulation are joined and passed to the post processor in the usual way.
Note that since sometimes the AMC pricing engines have a different base ccy than the risk factor evolution model (see below), a horizon shift parameter in the simulation set up should be set for all currencies, so that the shift also applies to these reduced models.
The `amcPathDataInput` and `amcPathDataOutput` parameters are optional. If the output path is given, the generated path data is serialized to disk. The path should be either absolute or relative to the working directory. If the input path is given, path generation is suppressed and paths are deserialized from the specified file. This can be used to speed up calculations using the same simulation setup.

The `amcIndividualTrainingOutput` and `amcIndividualTrainingInput` parameters are optional. If the output is requested, each trade's AMC calculator is serialized to disk. If `amcIndividualTrainingInput` is set to Y and the binary files have been previously generated, AMC calculator generation is suppressed and the AMC calculator is deserialized from the appropriate file.

3.2 Pricing Engine Configuration

The pricing engine configuration is similar for all AMC enabled products, e.g. for Bermudan swaptions:

```
<Product type="BermudanSwaption">
  <Model>LGM</Model>
  <ModelParameters/>
  <Engine>AMC</Engine>
  <EngineParameters>
    <Parameter name="Training.Sequence">MersenneTwisterAntithetic</Parameter>
    <Parameter name="Training.Seed">42</Parameter>
    <Parameter name="Training.Samples">10000</Parameter>
    <Parameter name="Training.BasisFunction">Monomial</Parameter>
    <Parameter name="Training.BasisFunctionOrder">6</Parameter>
    <Parameter name="Pricing.Sequence">SobolBrownianBridge</Parameter>
    <Parameter name="Pricing.Seed">17</Parameter>
    <Parameter name="Pricing.Samples">0</Parameter>
    <Parameter name="BrownianBridgeOrdering">Steps</Parameter>
    <Parameter name="SobolDirectionIntegers">JoeKuoD7</Parameter>
    <Parameter name="MinObsDate">true</Parameter>
    <Parameter name="RegressorModel">Simple</Parameter>
    <Parameter name="RegressionVarianceCutoff">1E-5</Parameter>
    <Parameter name="RecalibrateOnStickyCloseOutDates">false</Parameter>
    <Parameter name="ReevaluateExerciseInStickyRun">false</Parameter>
  </EngineParameters>
</Product>
```

The `Model` differs by product type, table 1 summarises the supported product types and model and engine types. The engine parameters are the same for all products:

1. `Training.Sequence`: The sequence type for the traning phase, can be `MersenneTwister`, `MersenneTwisterAntithetc`, `Sobol`, `Burley2020Sobol`, `SobolBrownianBridge`, `Burley2020SobolBrownianBridge`
2. `Training.Seed`: The seed for the random number generation in the training phase
3. `Training.Samples`: The number of samples to be used for the training phase
4. `Pricing.Sequence`: The sequence type for the pricing phase, same values allowed as for training
5. `Training.BasisFunction`: The type of basis function system to be used for the regression analysis, can be `Monomial`, `Laguerre`, `Hermite`, `Hyperbolic`, `Legendre`, `Chbyshev`, `Chebyshev2nd`
6. `BasisFunctionOrder`: The order of the basis function system to be used
7. `Pricing.Seed`: The seed for the random number generation in the pricing

8. `Pricing.Samples`: The number of samples to be used for the pricing phase. If this number is zero, no pricing run is performed, instead the (T0) NPV is estimated from the training phase (this result is used to fill the T0 slice of the NPV cube)

9. `BrownianBridgeOrdering`: variate ordering for Brownian bridges, can be `Steps`, `Factors`, `Diagonal`

10. `SobolDirectionIntegers`: direction integers for Sobol generator, can be `Unit`, `Jaeckel`, `SobolLevitan`, `SobolLevitanLemieux`, `JoeKuoD5`, `JoeKuoD6`, `JoeKuoD7`, `Kuo`, `Kuo2`, `Kuo3`

11. `MinObsDate`: if true the conditional expectation of each cashflow is taken from the minimum possible observation date (i.e. the latest exercise or simulation date before the cashflow's event date); recommended setting is `true`

12. `RegressorModel`: Simple, LaggedFX. If not given, it defaults to Simple. Depending on the choice the regressor is built as follows:

    - Simple: For an observation date the full model state observed on this date is included in the regressor. No past states are included though.
    - LaggedFX: For an observation date the full model state observed on this date is included in the regressor. In addition, past FX states that are relevant for future cashflows are included. For example, for a FX resettable cashflow the FX state observed on the FX reset date is included.

13. `RegressionVarianceCutoff`: Optional. If given, a coordinate transform and (possibly) a factor reduction is applied to the regressors, such that $1 - \epsilon$ of the total variance of regressors is kept, where ϵ the given parameter. This helps dealing with collinearity and also reducing the dimnensionality of the regression model.

14. `RecalibrateOnStickyCloseOutDates`: Optional, defaults to false. If true, the amc regression models are retrained on close-out dates in a sticky-date simulation. Otherwise the regression model from the valuation date is reused for the sticky run. If the valuation date is close to today (e.g. today + 1D) and the close-out date is relatively speaking further out (e.g. today + 15D), recalibration on close-out dates might be necessary to ensure stable results.

15. `ReevaluateExerciseInStickyRun`: Optional, defaults to false. If true, the exercise decision is updated for the run on sticky close-out dates, otherwise the exercise indicator from the valuation date is reused on the close-out date.

Product Type	Model	Engine
Swap	CrossAssetModel	AMC
CrossCurrencySwap	CrossAssetModel	AMC
FxOption	CrossAssetModel	AMC
BermudanSwaption	LGM	AMC
MultiLegOption	CrossAssetModel	AMC

Table 1: AMC enabled products with engine and model types

A sample configuration file can be found in ExamplesPlus / ExamplePlus_AMC / Input / pricingengine_amc.xml. Currently it contains the following product configurations:

1. `Swap`
2. `CrossCurrencySwap`
3. `FxOption`
4. `BermudanSwaption`
5. `MultiLegOption`

For technical reasons the following product configurations for coupon pricers are also present, sinced required by trade builders when setting up a leg. We note that they are not used in the AMC pricing engines though, so their detailled configuration does not matter really:

1. `CappedFlooredIborLeg`
2. `CMS`

4 Examples

The folder `ExamplesPlus/ExamplePlus_AMC` contains an example using the AMC valuation engine to produce exposure profiles (EPE, ENE) for example trades:

1. Bermudan swaption
2. Single Currency Swap
3. Cross Currency Swap
4. FX Option

In this section we compare these AMC exposure profiles with those produced by the classic valuation engine. If not stated otherwise the number of training paths and simulation paths is 10k in all cases and the simulation grid has a 3M spacing covering 88 points (22 years).

4.1 Bermudan Swaption

Figure 1 shows the EPE for a Bermudan swaption 10y into 10y in (base ccy) EUR with physical settlement calculated with the AMC and classic valuation engines (this example is taken from the unit tests). The classic run uses the LGM grid engine for valuation. We observe a good consistency between the two runs, the error tolerance for this test is set to 20 basispoints absolute NPV difference per unit notional.

4.2 Single Currency Swap

Figure 2 shows the EPE and ENE for a 20y vanilla Swap in USD (taken from the AMC Example). The currency of the amc calculator is USD in this case, i.e. it is different from the base ccy of the simulation (EUR). The consistency of the classic and amc runs in particular demonstrates the correct application of the currency conversion factor 1. To get a better accuracy for purposes of the plot in this document we increased the number of training paths for this example to 50k and the order of the basis functions to 6.

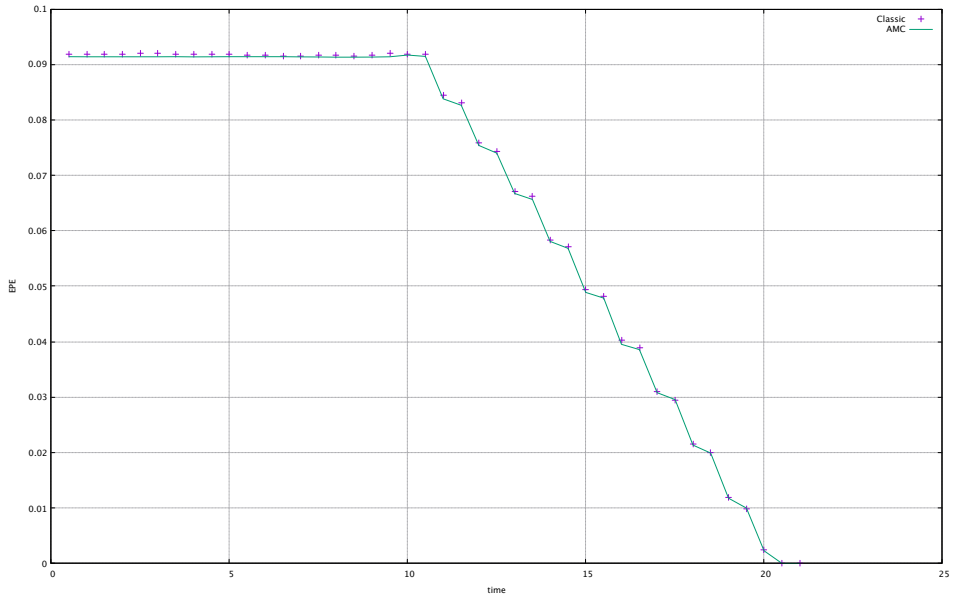

Figure 1: EPE of a EUR Bermudan Swaption computed with the classic and AMC valuation engines

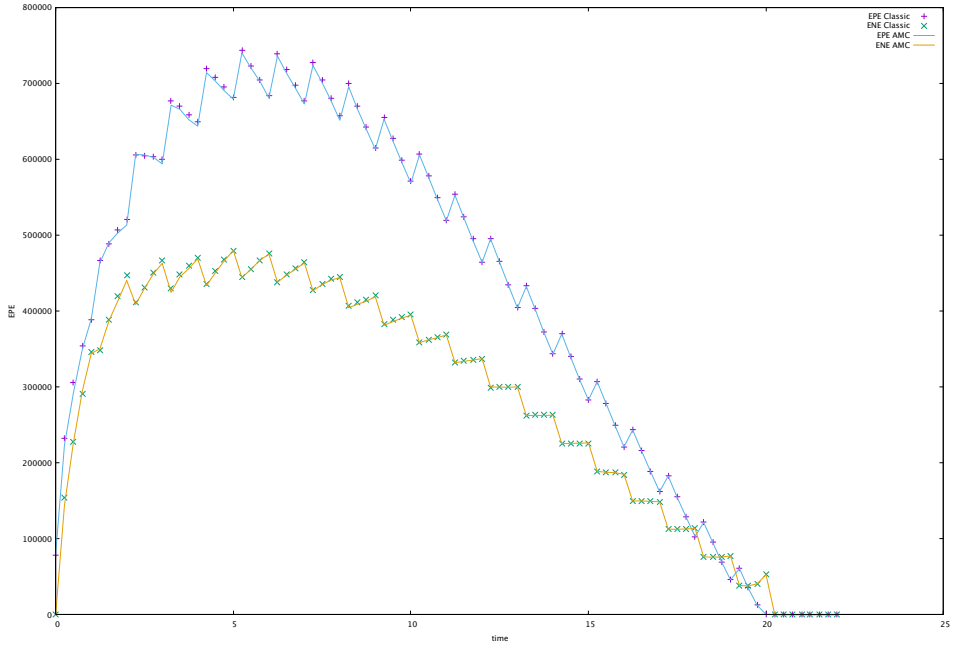

Figure 2: EPE of a USD swap computed with the classic and AMC valuation engines

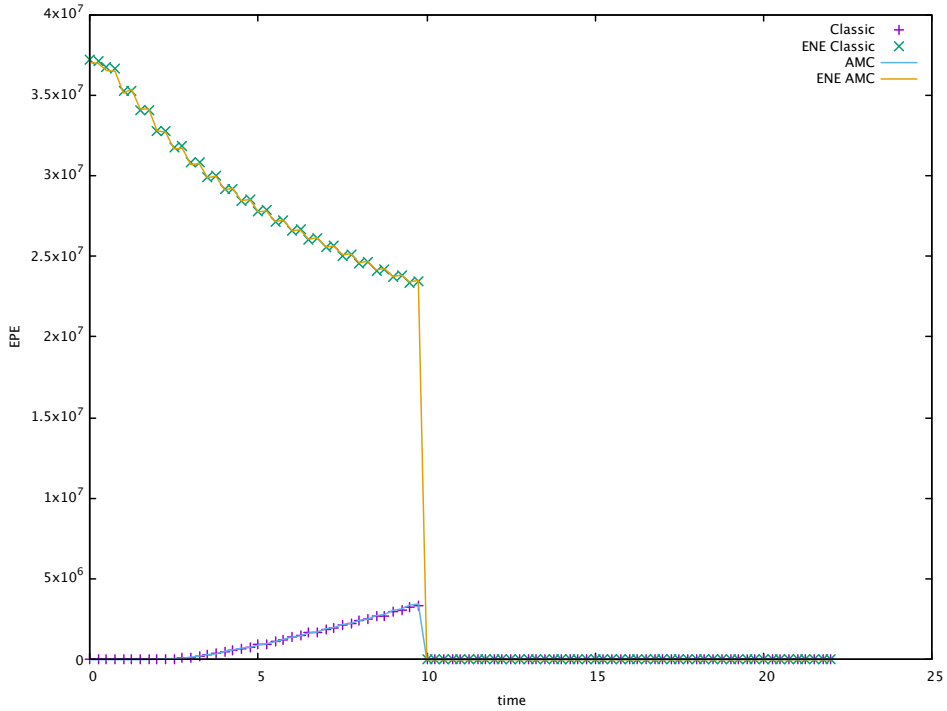

Figure 3: EPE of a EUR-USD cross currency swap computed with the classic and AMC valuation engines

4.3 Cross Currency Swap

Figure 3 shows the EPE and ENE for a 20y cross currency Swap EUR-USD (taken from the AMC Example).

4.4 FX Option

Figure 4 shows the EPE and ENE for a vanilla FX Option EUR-USD with 10y1m expiry (taken from the AMC Example). For the classic run the FX volatility surface is not implied by the cross asset model but kept flat, which yields a slight hump in the profile. The AMC profile is flat on the other hand which demonstrates the consistency of the FX Option pricing with the risk factor evolution model, see the discussion in 2.

5 Additional Features

As a side product the AMC module provides plain MC pricing engines for Bermudan swaptions and a new trade type `MultiLegOption` with a corresponding MC pricing engine.

5.1 MC pricing engine for Bermudan swaptions

The following listing shows a sample configuration for the MC Bermudan swaption engine. The model parameters are identical to the LGM Grid engine configuration. The engine parameters on the other hand are the same as for the AMC engine, see 3.2.

```
<Product type="BermudanSwaption">
  <Model>LGM</Model>
```

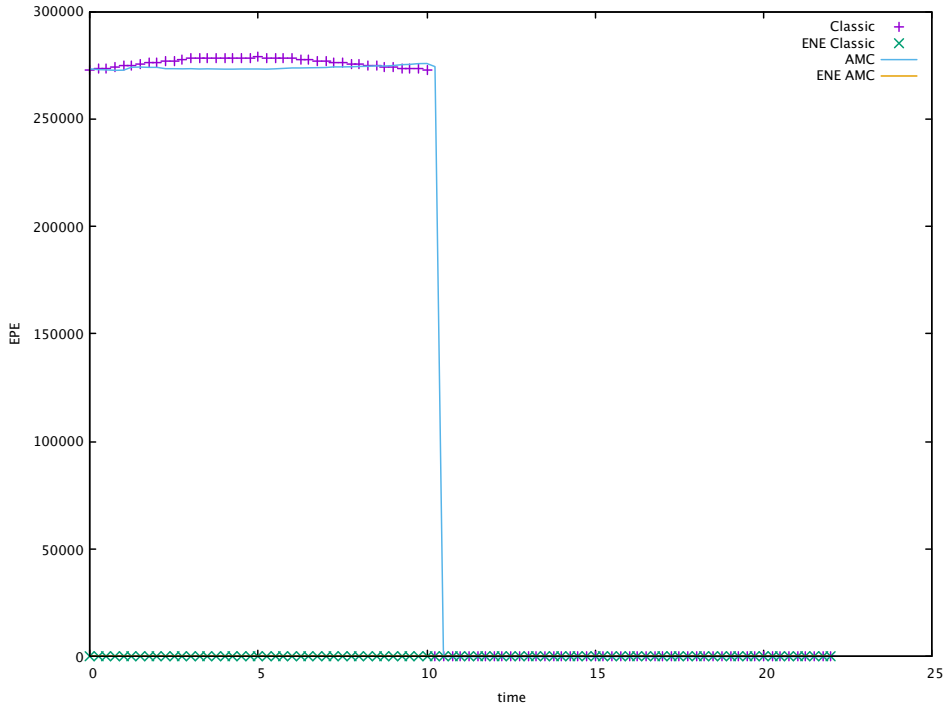

Figure 4: EPE of a EUR-USD FX option computed with the classic and AMC valuation engines

```
  <ModelParameters>
    <Parameter name="Calibration">Bootstrap</Parameter>
    <Parameter name="CalibrationStrategy">CoterminalDealStrike</Parameter>
    <Parameter name="Reversion_EUR">0.0050</Parameter>
    <Parameter name="Reversion_USD">0.0030</Parameter>
    <Parameter name="ReversionType">HullWhite</Parameter>
    <Parameter name="VolatilityType">HullWhite</Parameter>
    <Parameter name="Volatility">0.01</Parameter>
    <Parameter name="ShiftHorizon">0.5</Parameter>
    <Parameter name="Tolerance">1.0</Parameter>
  </ModelParameters>
  <Engine>MC</Engine>
  <EngineParameters>
    <Parameter name="Training.Sequence">MersenneTwisterAntithetic</Parameter>
    <Parameter name="Training.Seed">42</Parameter>
    <Parameter name="Training.Samples">10000</Parameter>
    <Parameter name="Training.BasisFunction">Monomial</Parameter>
    <Parameter name="Training.BasisFunctionOrder">6</Parameter>
    <Parameter name="Pricing.Sequence">SobolBrownianBridge</Parameter>
    <Parameter name="Pricing.Seed">17</Parameter>
    <Parameter name="Pricing.Samples">25000</Parameter>
    <Parameter name="BrownianBridgeOrdering">Steps</Parameter>
    <Parameter name="SobolDirectionIntegers">JoeKuoD7</Parameter>
  </EngineParameters>
</Product>
```

5.2 Multi Leg Options / MC pricing engine

The following listing shows a sample MultiLegOption trade. It consists of

1. an option data block; this is optional, see below

2. a number of legs; in principle all leg types are supported, the number of legs is arbitrary and they can be in different currencies; if the payment currency of a leg is different from a floating index currency, this is interpreted as a quanto payoff

If the option block is given, the trade represents a Bermudan swaption on the underlying legs. If the option block is missing, the legs themselves represent the trade.
See 7.1 and 7.4 for limitations of the multileg option pricing engine.

```
<Trade id="Sample_MultiLegOption">
  <TradeType>MultiLegOption</TradeType>
  <Envelope>...</Envelope>
  <MultiLegOptionData>
    <OptionData>
      <LongShort>Long</LongShort>
      <OptionType>Call</OptionType>
      <Style>Bermudan</Style>
      <Settlement>Physical</Settlement>
      <PayOffAtExpiry>false</PayOffAtExpiry>
      <ExerciseDates>
        <ExerciseDate>2026-02-25</ExerciseDate>
        <ExerciseDate>2027-02-25</ExerciseDate>
        <ExerciseDate>2028-02-25</ExerciseDate>
      </ExerciseDates>
    </OptionData>
    <LegData>
      <LegType>Floating</LegType>
      <Payer>false</Payer>
      <Currency>USD</Currency>
      <Notionals>
        <Notional>100000000</Notional>
      </Notionals>
      ...
    </LegData>
    <LegData>
      <LegType>Floating</LegType>
      <Payer>true</Payer>
      <Currency>EUR</Currency>
      <Notionals>
        <Notional>100000000</Notional>
      </Notionals>
      ...
    </LegData>
  </MultiLegOptionData>
</Trade>
```

The pricing engine configuration is similar to that of the MC Bermudan swaption engine, cf. 5.1, also see the following listing.

```
  <Product type="MultiLegOption">
  <Model>CrossAssetModel</Model>
  <ModelParameters>
    <Parameter name="Tolerance">0.0001</Parameter>
    <!-- IR -->
    <Parameter name="IrCalibration">Bootstrap</Parameter>
    <Parameter name="IrCalibrationStrategy">CoterminalATM</Parameter>
    <Parameter name="ShiftHorizon">1.0</Parameter>
    <Parameter name="IrReversion_EUR">0.0050</Parameter>
```

```
    <Parameter name="IrReversion_GBP">0.0070</Parameter>
    <Parameter name="IrReversion_USD">0.0080</Parameter>
    <Parameter name="IrReversion">0.0030</Parameter>
    <Parameter name="IrReversionType">HullWhite</Parameter>
    <Parameter name="IrVolatilityType">HullWhite</Parameter>
    <Parameter name="IrVolatility">0.0050</Parameter>
    <!-- FX -->
    <Parameter name="FxCalibration">Bootstrap</Parameter>
    <Parameter name="FxVolatility_EURUSD">0.10</Parameter>
    <Parameter name="FxVolatility">0.08</Parameter>
    <Parameter name="ExtrapolateFxVolatility_EURUSD">false</Parameter>
    <Parameter name="ExtrapolateFxVolatility">true</Parameter>
    <!-- Correlations IR-IR, IR-FX, FX-FX -->
    <Parameter name="Corr_IR:EUR_IR:GBP">0.80</Parameter>
    <Parameter name="Corr_IR:EUR_FX:GBPEUR">-0.50</Parameter>
    <Parameter name="Corr_IR:GBP_FX:GBPEUR">-0.15</Parameter>
  </ModelParameters>
  <Engine>MC</Engine>
  <EngineParameters>
    <Parameter name="Training.Sequence">MersenneTwisterAntithetic</Parameter>
    <Parameter name="Training.Seed">42</Parameter>
    <Parameter name="Training.Samples">10000</Parameter>
    <Parameter name="Pricing.Sequence">SobolBrownianBridge</Parameter>
    <Parameter name="Pricing.Seed">17</Parameter>
    <Parameter name="Pricing.Samples">25000</Parameter>
    <Parameter name="Training.BasisFunction">Monomial</Parameter>
    <Parameter name="Training.BasisFunctionOrder">4</Parameter>
    <Parameter name="BrownianBridgeOrdering">Steps</Parameter>
    <Parameter name="SobolDirectionIntegers">JoeKuoD7</Parameter>
  </EngineParameters>
</Product>
```

Model Parameters special to that product are

1. `IrCalibrationStrategy` can be `None`, `CoterminalATM`, `UnderlyingATM`
2. `FXCalibration` can be `None` or `Bootstrap`
3. `ExtrapolateFxVolatility` can be `true` or `false`; if false, no calibration instruments are used that require extrapolation of the market fx volatilty surface in option expiry direction
4. `Corr_Key1_Key2`: These entries describe the cross asset model correlations to be used; the syntax for `Key1` and `Key2` is the same as in the simulation configuration for the cross asset model, see [1].

6 Implementation Details

6.1 AMC valuation engine and AMC pricing engines

The `AMCValuationEngine` is responsible for generating a NPV cube for a portfolio of AMC enabled trades and (optionally) to populate a `AggregationScenarioData` instance with simulation data for post processing, very similar to the classic `ValuationEngine` in ORE.

The AMC valuation engine takes a cross asset model defining the risk factor evolution. This is set up identically to the cross asset model used in the `CrossAssetModelScenarioGenerator`. Similarly the same parameters for the path generation (given as a `ScenarioGeneratorData` instance) are used, so that it is guaranteed that both the AMC engine and the classic engine produce the same paths, hence can be combined to a single cube for post processing. It is checked, that a non-zero seed for the random number generation is used.
The portfolio that the AMC engine consumes is build against an engine factory set up by a pricing engine configuration given in the amc analytics type (see 3.1). This configuration should select special AMC engine builders which (by a pure naming convention) have the engine type "AMC". These engine builders are retrieved from `getAmcEngineBuilders()` in `oreappplus.cpp` and are special in that unlike usual engine builders they take two parameters

1. the cross asset model which serves as a risk factor evolution model in the AMC valuation engine

2. the date grid used within the AMC valuation engine

For technical reasons, the configuration also contains configurations for `CapFlooredIborLeg` and `CMS` because those are used within the trade builders (more precisely the leg builders called from these) to build the trade. The configuration can be the same as for T0 pricing for them, it is actually not used by the AMC pricing engines.
The AMC engine builders build a smaller version of the global cross asset model only containing the model components required to price the specific trade. Note that no deal specific calibration of the model is performed.
The AMC pricing engines perform a T0 pricing and - as a side product - can be used as usual T0 pricing engines if a corresponding engine builder is supplied, see 5.
In addition the AMC pricing engines perform the necessary calculations to yield conditional NPVs on the given global simulation grid. How these calculations are performed is completely the responsibility of the pricing engines, altough some common framework for many trade types is given by a base engine, see 6.2. This way the approximation of conditional NPVs on the simulation grid can be taylored to each product and also each single trade, with regards to

1. the number of traning paths and the required date grid for the training (e.g. containing all relevant coupon and exercise event dates of a trade)

2. the order and type of regressoin basis functions to be used

3. the choice of the regressor (e.g. a TaRN might require a regressor augmented by the accumulated coupon amount)

The AMC pricing engines then provide an additional result labelled `amcCalculator` which is a class implementing the `AmcCalculator` interface which consists of two methods: The method `simulatePath()` takes a `MultiPath` instance representing one simulated path from the global risk factor evolution model and returns an array of conditional, deflated NPVs for this path. The method `npvCurrency()` returns the currency c of the calculated conditional NPVs. This currency can be different from the base currency b of the global risk factor evolution model. In this case the conditional NPVs are converted

to the global base currency within the AMC valuation engine by multiplying them with the conversion factor

$$\frac{N_c(t)X_{c,b}(t)}{N_b(t)} \tag{1}$$

where t is the simulation time, $N_c(t)$ is the numeraire in currency c, $N_b(t)$ is the numeraire in currency b and $X_{c,b}(t)$ is the FX rate at time t converting from c to b.
The technical criterion for a trade to be processed within the AMC valuation is engine is that a) it can be built against the AMC engine factory described above and b) it provides an additional result `amcCalculator`. If a trade does not meet these criteria it is simulated using the classic valuation engine. The logic that does this is located in the overide of the method `OREAppPlus::generateNPVCube()`.
The AMC valuation engine can also populate an aggregation scenario data instance. This is done only if necessary, i.e. only if no classic simulation is performed anyway. The numeraire and fx spot values produced by the AMC valuation engine are identical to the classic engine. Index fixings are close, but not identical, because the AMC engine used the T0 curves for projection while the classic engine uses scenario simulation market curves, which are not exactly matching those of the T0 market. In this sense the AMC valuation engine produces more precise values compared to the classic engine.

6.2 The multileg option AMC base engine and derived engines

Table 1 provides an overview of the implemented AMC engine builders. These builders use the following QuantExt pricing engines

1. `McLgmSwapEngine` for single currency swaps
2. `McCamCurrencySwapEngine` for cross currency swaps
3. `McCamFxOptionEngine` for fx options
4. `McLgmSwaptionEngine` for Bermudan swaptions
5. `McMultiLegOptionEngine` for Multileg option
6. `McLgmBondEngine` for plain vanilla bonds
7. `McLgmFwdBondEngine` for forward bonds

All these engine are based on a common `McMultiLegBaseEngine` which does all the computations. For this each of the engines sets up the following protected member variables (serving as parameters for the base engine) in their `calculate()` method:

1. `leg_`: a vector of `QuantLib::Leg`
2. `currency_`: a vector of `QuantLib::Currency` corresponding to the leg vector
3. `payer_`: a vector of $+1.0$ or -1.0 double values indicating receiver or payer legs
4. `exercise_`: a `QuantLib::Exercise` instance describing the exercise dates (may be `nullptr`, if the underlying represents the deal already)

5. `optionSettlement_`: a `Settlement::Type` value indicating whether the option is settled physically or in cash

Though both bond engines inherit from `McMultiLegBaseEngine` bespoke `calculate()` and `amcCalculator()` methods are in place for the bond flavours.
A call to `McMultiLegBaseEngine::calculate()` will set the result member variables

1. `resultValue_`: T0 NPV in the base currency of the cross asset model passed to the pricing engine
2. `underlyingValue_`: T0 NPV of the underlying (again in base ccy)
3. *`amcCalculator_`: the AMC calculator engine to be used in the AMC valuation engine

The specific engine implementations should convert the `resultValue_` to the npv currency of the trade (as defined by the (ORE) trade builder) so that they can be used as regular pricing engine consistently within ORE. Note that only the additional `amcCalculator` result is used by the AMC valuation engine, not any of the T0 NPVs directly.

7 Limitations and Open Points

This sections lists known limitations of the AMC simulation engine.

7.1 Trade Features

Some trade features are not yet supported by the multileg option engine:

1. legs with fx resetting feature
2. legs with naked option = true
3. coupon types are restricted to Ibor and CMS
4. exercise flows (like a notional exchange common to cross currency swaptions) are not supported

7.2 Flows Generation (for DIM Analysis)

At the current stage the AMC engine does not generate flows which are required for the DIM analysis in the post processor.

7.3 State interpolation for exercise decisions

During the simulation phase exercise times of a specific trade are not necessarily part of the simulated time grid. Therefore the model state required to take the exercise decision has in to be interpolated in general on the simulated path. Currently this is done using a simple linear interpolation while from a pure methodology point of view a Brownian Bridge would be preferable. In our tests we do not see a big impact of this approximation though.

7.4 Missing recalibration of the MCMultiLegOptionEngine

The MC Multi Leg Option Engine builder uses the `CrossAssetModelBuilder` to set up the pricing model. This class does not implement the `ModelBuidler` interface meaning that the model is not recalibrated in a sensitivity analysis run. Therefore the sensitivities calculated by this engine are not valid.

7.5 Basis Function Selection

Currently the basis function system is generated by specifying the type of the functions and the order, see 3.2. The number of indenpendent variables varies by product type and details. Depending on the number of independent variables and the order the number of generated basis functions can get quite big which slows down the computation of regression coefficients. It would be desirable to have the option to filter the full set of basis functions, e.g. by explicitly enumerating them in the configuration, so that a high order can be chosen even for products with a relatively large number of independent variables (like e.g. FX Options or Cross Currency Swaps).

8 Outlook

8.1 Trade Compression

For vanilla trades where the regression is only required to produce the NPV cube entries (and not to take exercise decisions etc.) it is not strictly necessary to do the regression analysis on a single trade level[2]. Although in the current implementation there is no direct way to do the regression analysis on whole (sub-)portfolios instead of single trades, one can represent such a subportfolio as a single technical trade (e.g. as a single swap or multileg option trade) to achieve a similar result. This might lead to better performance than the usual single trade calculation. However one should also try to keep the regressions as low-dimensional as possible (for performance and accuracy reasons) and therefore define the subportfolios by e.g. currency, i.e. as big as possible while at the same time keeping the associated model dimension as small as possible.

References

[1] ORE User Guide, http://www.opensourcerisk.org/documentation/

[2] except single trade exposures are explicitly required of course