ORE Credit Migration Model

Acadia Inc.

4 May 2023

Document History

Date	Author	Comment
4 May 2023	Acadia	initial release

Contents

1 Introduction

This paper summarizes the methodology of the Credit Migation Model provided in ORE, its parametrization and implementation details.

2 Methodology

2.1 Credit-worthiness model and transition matrices

We model the credit-worthiness X_i of an "entity" i. Here, an entity represents either a *counterparty* in a (derivative) trade or an *issuer* referenced in a funded position exposed to credit risk (e.g. a Bond, Loan) or credit derivative trade (e.g. CDS). The credit-worthiness X_i is modelled as

$$dX_i = dY_i + dZ_i, \qquad dY_i = \sum_{j=1}^{n} \beta_{ij}\, dG_j, \qquad dZ_i = \sigma_i dW_i \tag{1}$$

Here:

- Y_i is the systematic part of the credit risk driven by global credit factors G_j (shared between entities), which are standard Brownian motions with correlation $dG_k\, dG_l = \rho_{kl}\, dt$. The β_{ij} are entity-specifc loadings.
- Z_i is the idiosyncratic part of the credit risk of entity i, driven by standard Brownian motions W_i which are independent of all global factors G_j
- β_{ij} and σ_i are chosen such that X_i is a standard brownian motion, i.e. it generates variance t up to time t

The systemic/global factors G_i are evolved in ORE's standard exposure simulation. Conditional on the systematic factors the credit worthiness indices X_i are independent.

Input to the credit migration model is an *unconditional* annual transition matrix M_i that can be entity dependent. For n rating states including the default state M_i is an $n \times n$ matrix. The kth row and jth column of M_i contains the probability p_{kj} to migrate from rating state k to state j, where the states are ordered from highest to lowest quality. A typical matrix is given in 3.4.

To derive (unconditional) transition matrices $M_i(t)$ for arbitrary time horizons $t \in (0, \infty)$ we use the algorithm QOG in [2].

Writing

$$P_{kf} = \sum_{j=1}^{f} p_{kj} \tag{2}$$

for the probability to migrate from initial state k to some final state better or equal in quality than f, it is straightforward to construct the migration matrix $M_i(t) \,|\, Y_i$ conditional on the realisation of the systematic part Y_i as

$$P_{kf} \,|\, Y_i := \Phi\left(\frac{\Phi^{-1}(P_{kf}) - Y_i(t)/\sqrt{t}}{\sigma_i}\right) \tag{3}$$

2.2 PnL Distributions

We generate a PnL distribution for a subset of the exposure simulation time steps. For a fixed time step the total PnL is given by the sum of

- the market risk PnL (if active in the credit simulation config, see 3.4)
- the credit migration risk PnL (if active, again see 3.4)

2.2.1 Market Risk PnL

The market risk PnL is a single number per exposure simulation path and computed as

$$-\nu_0 + \sum_{l=1}^{L} c(l) + \nu_L \tag{4}$$

where ν_0 is the NPV of the simulated portfolio at the T0 valuation date, $c(l)$ is the NPV of the cashflows between simulation time step $l-1$ (first future simulation time step has index 0) and l and ν_L is the NPV of the simulated portfolio at the simulation time step for which the PnL distribution is generated.

The market risk PnL is computed for the full simulated portfolio.

2.2.2 Credit Migration PnL

The credit migration PnL is itself computed as a distribution conditional on a single exposure simulation path. The path-wise distributions are averaged over all exposure simulation paths to get the final credit migration PnL distribution.

The credit migration PnL splits into

- credit migration PnL from counterparty risk for trades that are in a specified netting set (NettingSetIds in 3.4)
- credit migration PnL from issuer risk for trades that are exposed to issuer risk (Bonds, CDS etc.[1]) *and* the issuer is specified under Entities in 3.4

The former part of the credit migration PnL is determined as the default risk of the netted exposure (after collateralisation) from the usual exposure post-processing. No migration risk is considered here.

The latter part of the credit migration PnL is generated in different ways depending on the evaluation mode (see 3.4) as follows:

Analytic: In this mode we do not simulate the idiosyncratic factors. We exploit the fact that the credit-worthiness indicators X_i are conditionally independent given the systematic state Y_i. We use a multi-state Hull-White bucketing algorithm to generate the distribution. Each entity contributes a probability / PnL vector of size n where n

[1] current coverage: trade types CreditDefaultSwap, Bond

is the number of rating states to the bucketing algorithm. The PnL vector is based on the conditional transition matrix.

TerminalSimulation: Based on the conditional transition matrix for an outer exposure simulation path we run an inner Monte Carlo simulation to generate realisations of the idiosyncratic risk for each entity at the terminal/horizon date t_n. The number of paths for this inner simulation is specified in 3.4. For each inner path we get a final migration state for an entity and from that a PnL contribution.

3 Parametrisation

3.1 General settings in ore.xml

There is a credit-model specific entry for Analytic type *simulation* that determines that multiple market values are stored in the NPV cube corresponding to the rating states at a simulation point. The number given should be equal to the number of rating states including the default state that are considered in the credit-model run. If omitted, no credit-state dependent market values are stored in the NPV cube, which will generally let the post-processing step for the credit-model fail. The following setting is suitable for e.g. rating states Aaa, Aa, A, Baa, Ba, B, C, Default.

```
<Analytic type="simulation">
  <Parameter name="storeCreditStateNPVs">8</Parameter>
  ...
```

There are several parameters in the Analytic type *XVA* relevant for the post-processing step of the credit-model:

```
<Analytic type="xva">
  <Parameter name="creditMigration">Y</Parameter>
  <Parameter name="creditMigrationDistributionGrid">-2E7,1E7,300</Parameter>
  <Parameter name="creditMigrationTimeSteps">3,11</Parameter>
  <Parameter name="creditMigrationConfig">creditsimulation.xml</Parameter>
  <Parameter name="creditMigrationOutputFiles">credit_migration_bond1</Parameter>
  ...
```

The meaning of the parameters are as follows:

- creditMigration: Y to activate the credit-model post-processing step, N to deactivate this step
- creditMigrationDistributionGrid: parameter triple min, max, steps that defines the binning of the PnL distribution
- creditMigrationTimeSteps: simulation time steps to evaluate, 0 refers to the first future simulation time
- creditMigrationConfig: credit simulation config, described in 3.4
- creditMigrationOutputFiles: prefix for output PnL distribution files, the time step and file extension csv will be appended to that name

3.2 Simulation settings in simulation.xml

The cross asset model section should contain the number of systematic credit factors Z_i to evolve in the cross asset model.

```
<CrossAssetModel>
  <CreditStates>
    <NumberOfFactors>1</NumberOfFactors>
  </CreditStates>
  ...
```

The market section should contain the same entry [2]

```
<Market>
  <CreditStates>
    <NumberOfFactors>1</NumberOfFactors>
  </CreditStates>
  ...
```

The market section should also include a similar entry for the aggregation scenario data generation (since the systematic credit states are stored in the aggregation scenario data)

```
<Market>
  <AggregationScenarioDataCreditStates>
    <NumberOfFactors>1</NumberOfFactors>
  </AggregationScenarioDataCreditStates>
  ...
```

and entries that specify credit curve ids that are used in the postprocessing. For each credit curve id the survival probability up to the simulation point on each path are stored. The following stores this information for 7 credit curves labelled "Bond" plus rating (excluding default in this case):

```
<Market>
  <AggregationScenarioDataSurvivalWeights>
    <Name>BOND_AAA</Name>
    <Name>BOND_AA</Name>
    <Name>BOND_A</Name>
    <Name>BOND_BBB</Name>
    <Name>BOND_BB</Name>
    <Name>BOND_B</Name>
    <Name>BOND_C</Name>
  </AggregationScenarioDataSurvivalWeights>
  ...
```

[2] This is redundant, but technically more transparent: The former entry determines the number of credit factors used to evolve the cross asset model while the latter entry determines the number of credit factors to include in a scenario in the simulation market.

3.3 Pricing engine config in pricingengine.xml

If multiple credit state NPVs should be stored during the simulation run (see 3.1), suitable pricing engines have to be used that allow for the computation of several rating state NPVs. By convention, those pricing engines store the state NPVs in an additional result with label "stateNPVs". If this additional result is not provided for a trade, the multi-state valuation calculator will populate all state npvs in the NPV cube with the same value as the usual npv valuation calculator does. The "stateNPVs" result contains a default value as well, i.e. if there are e.g. 8 rating states in total including a default state, the result will be a vector of length 8.

Currently there are multi-state pricing engines for bond and cds. Example config for bond:

```
<Product type="Bond">
  <Model>DiscountedCashflows</Model>
  <ModelParameters/>
  <Engine>DiscountingRiskyBondEngineMultiState</Engine>
  <EngineParameters>
    <Parameter name="TimestepPeriod">6M</Parameter>
    <Parameter name="Rule_0">(_AAA$|_AA$|_A$|_BBB$|_BB$|_B$|_C$),_AAA</Parameter>
    <Parameter name="Rule_1">(_AAA$|_AA$|_A$|_BBB$|_BB$|_B$|_C$),_AA</Parameter>
    <Parameter name="Rule_2">(_AAA$|_AA$|_A$|_BBB$|_BB$|_B$|_C$),_A</Parameter>
    <Parameter name="Rule_3">(_AAA$|_AA$|_A$|_BBB$|_BB$|_B$|_C$),_BBB</Parameter>
    <Parameter name="Rule_4">(_AAA$|_AA$|_A$|_BBB$|_BB$|_B$|_C$),_BB</Parameter>
    <Parameter name="Rule_5">(_AAA$|_AA$|_A$|_BBB$|_BB$|_B$|_C$),_B</Parameter>
    <Parameter name="Rule_6">(_AAA$|_AA$|_A$|_BBB$|_BB$|_B$|_C$),_C</Parameter>
  </EngineParameters>
</Product>
```

The parameters Rule_0, ..., Rule_6 are rules how to substitute a regex within a credit curve id of a bond (left parameter) with a new string (right parameter). If the bond contains a credit curve id containing one of the rating states like e.g.

```
  <BondData>
    <IssuerId>CPTY_C</IssuerId>
    <CreditCurveId>BOND_BB</CreditCurveId>
    ...
```

this ensures that the bond will be priced using the credit curves with id BOND_AAA, ..., BOND_C.

3.4 Credit simulation config in creditsimulation.xml

This is the configuration of the credit-model postprocessing step.

The central object of the credit simulation are "entities" that stand for either counterparties of (derivative) trades or issuers referenced from bonds or credit default swaps.

The config contains one or several definition of (annual) transition matrices that define the overall migration probabilities for the entities. Example:

```
<TransitionMatrices>
  <TransitionMatrix>
    <Name>TransitionMatrix_1</Name>
    <Data t0="0.0" t1="1.0">
  <!--   Aaa     Aa      A       Baa     Ba      B       C    Default  -->
      0.8588, 0.0976, 0.0048, 0.0000, 0.0003, 0.0000, 0.0000, 0.0000,
      0.0092, 0.8487, 0.0964, 0.0036, 0.0015, 0.0002, 0.0000, 0.0004,
      0.0008, 0.0224, 0.8624, 0.0609, 0.0077, 0.0021, 0.0000, 0.0002,
      0.0008, 0.0037, 0.0602, 0.7916, 0.0648, 0.0130, 0.0011, 0.0019,
      0.0003, 0.0008, 0.0046, 0.0402, 0.7676, 0.0788, 0.0047, 0.0140,
      0.0001, 0.0004, 0.0016, 0.0053, 0.0586, 0.7607, 0.0274, 0.0660,
      0.0000, 0.0000, 0.0000, 0.0100, 0.0279, 0.0538, 0.5674, 0.2535,
      0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 1.0000
    </Data>
  </TransitionMatrix>
</TransitionMatrices>
```

The parameters t0, t1 can be used to define a term structure of transition matrices (not supported at the moment). The transition matrices are references from the entities definitions that follow next. Here, the entity identified by `CPTY_A` uses the migration matrix `TransitionMatrix_1`. The is a single factor loading β defined here, i.e. the assumption is that one systemic credit factor Z_i is evolved. Several factor loading for multifactor models are specified as a comma separated list. The initial state define the T0 rating state. This should be consistent with the credit curve setup in the trades / reference data, i.e. bonds or cds referencing `CPTY_A` should use a default curve `CPTY_A_Baa`, since Baa is the 4th credit state and numbering of the initial states starts at 0.

```
<Entities>
  <Entity>
    <Name>CPTY_A</Name>
    <FactorLoadings>0.4898979485566356</FactorLoadings>
    <TransitionMatrix>TransitionMatrix_1</TransitionMatrix>
    <InitialState>3</InitialState>
  </Entity>
  ...
```

The netting set id entry determines which netting sets are considered in the calculation of default risk for derivative exposure. This uses the netted exposure (i.e. after collateral) with zero recovery (hardcoded assumption at the moment).

```
<NettingSetIds>
  CPTY_B,CPTY_C
</NettingSetIds>
```

Finally the risk section defines how and which risks are considered in the postprocessing step:

```
<Risk>
  <Market>N</Market>
  <Credit>Y</Credit>
  <ZeroMarketPnl>N</ZeroMarketPnl>
```

```
    <Evaluation>ForwardSimulationB</Evaluation>
    <CreditMode>Migration</CreditMode>
    <LoanExposureMode>Value</LoanExposureMode>
    <DoubleDefault>Y</DoubleDefault>
    <Paths>1000</Paths>
    <Seed>42</Seed>
  </Risk>
```

The meanings of the parameters are:

- Market: Include market risk pnl, this is derived from the simulated market values (in the initial state only) and cashflows in the NPV cube
- Credit: Include credit risk pnl, this is derived from
  - migration / default risk for counterparty derivative exposure (specified via NettingSetIds)
  - migration / default risk for issuer exposure (bond, cds)
- ZeroMarketPnl: If Y the market pnl of a credit-risk relevant trade (typically a CDS) is weighted with the survival probability until the simulation point.
- Evaluation: the method to compute the credit risk pnl:
  - Analytic: Simulation of systematic credit factors only. Exploit conditional independency of entity defaults and Hull-White bucketing to generate credit risk pnl.
  - TerminalSimulation: Simulation of systematic credit risk factors using a large step from $t = 0$ to the current analyzed time step and conditioning on the systematic credit risk factors at the terminal time step
- CreditMode: Migration or Default, indicating whether to consider all rating migrations or only the migration to default in the credit pnl
- LoanExposureMode: Notional or Value, indicating whether to consider the market value or the notional as a substitute for the market value in the credit pnl
- DoubleDefault: Consider double default risk for CDS (simultaneous default of both issuer and counterparty), this is only supported for Evaluation = Analytic.
- Paths: Number of "inner" simulation paths for ForwardSimulationA, ForwardSimulationB, TerminalSimulation. For each "outer" exposure simulation path, this number of inner paths are simulated to get the credit migration pnl distribution for the outer path
- Seed: Seed used to generate the inner simulation paths. A Mersenne Twister RNG is used for inner path generation.

4 Implementation Details

4.1 Matrix Utilities

4.1.1 Exponential and Logarithm of a matrix

The exponential and logarithm of a matrix is provided in `matrixfunctions.hpp` as `Logm()` and `Expm()`. The implementation depends on the availabilty of the Eigen library [1] during the build of ORE:

- Eigen is available: The Eigen implementation of the matrix exponential and logarithm are used. See https://eigen.tuxfamily.org/dox/unsupported/group__MatrixFunctions__Module.html#matrixbase_exp and https://eigen.tuxfamily.org/dox/unsupported/group__MatrixFunctions__Module.html#matrixbase_log.
- Eigen is not available: The QuantLib implementation of the matrix exponential is used. The matrix logarithm is not implemented and will throw an error when called. The credit model will in general not work in this case, i.e. the Eigen installation is required to ensure full functionality.

4.1.2 Transition matrix utilities

`transitionmatrix.hpp` contains some utilities related to transition matrices:

- `checkTransitionMatrix()` checks whether a given (quadratic) matrix is a transition matrix, i.e. all entries are in $[0, 1]$ and row sums are 1.
- `checkGeneratorMatrix()` checks whether a given (quadratic) matrix is a generator matrix, i.e. all entries are non-negative and row sums are 0
- `sanitiseTransitionMatrix()` modifies a matrix such that the result is a valid transition matrix. The function simply caps resp. floors the matrix entries at 1 resp. 0. If after that operation a row sum s excluding the diagonal element is ≤ 1 the diagonal element for that row is overwritten with $1 - s$. If the row sum s excluding the diagonal element is > 1, all elements in the row are divided by s. In both cases the resulting row entries will sum up to 1.
- `generator()` computes a generator for a given transition matrix and time horizon T following the algorithm QOG in [2].

4.2 CreditMigrationHelper

This class produces the pnl distributions. The main functions are:

- `build()`: given a set of trades (usually the whole simulation portfolio), identify
  - trades that reference a credit name which is also a configured entity, those contribute to the *issuer risk* within the credit migration pnl (current coverage: trade types bond, cds)
  - trades that fall into a netting set configured under NettingSetIds, those contribute to the *counterparty risk* within the credit migration pnl

- `pnlDistribution()`: build the aggregated market and credit migration pnl for a given time step. This breaks down as follows
  - Scale unconditional transition matrices to the given time step (using QOG in [2], i.e. a regularized generator)
  - For each (outer) exposure simulation path, compute the market risk pnl
  - Compute the credit migration pnl as described below.

The computation of the credit migration pnl in the last step is done as follows:

- Evaluation = Analytic: generate the credit-risk pnl using `generateConditionalMigrationPnl()`. This includes
  - computing the conditional transition matrices
  - generating the issuer-risk pnl at the simulation horizon based on the conditional matrices and the NPVs for all credit states computed with multi-state pricing engines
  - generating the counterparty-risk pnl at the simulation horizon based on the conditional migration probabilty to default and the usual NPV for relevant derivative trades
- Evaluation $\neq$ Analytic: generate the credit-risk pnl using calls to
  - `initEntityStateSimulation()`: Compute conditional transition matrices for large step (TerminalSimulation)
  - for each "inner" simulation path: simulate the entity state using `simulateEntityStates()`
  - for each "inner" simulation path: generate credit-risk pnl using `generateMigrationPnl()` based on the simulated entity state at the terminal time step and for issuer and counterparty risk (this is donw similar to Evaluation = Analytic described above)

References

[1] Eigen library https://eigen.tuxfamily.org

[2] Alexander Kreinin and Marina Sidelnikova, *Regularization Algorithms for Transition Matrices*, Algo Research Quaterly, Vol. 4, Nos. 1/2 March/June 2001