How to use Machine learning for efficient xVA calculations: The case of Credit Valuation Adjustment (CVA)

Study carried out by the Data Science Practice
Special thanks to Pascal PIERROT

coperneec

FEVRIER 2022

Summary

Introduction

Nowadays, with regulatory requirements, the valuation of any financial product should take into account the possibility of default of any agents involved in the transaction. Moreover, the trading activity should be funded by different source of liquidity instead of a single one with a unique risk-free interest rate. xVA denotes the various value adjustment that are applied to the valuation process to incorporate these stylized fact. xVA calculation is computationally challenging because it involves the evaluation of all the trades in the portfolio under each market and credit scenario. For instance, the Credit Value Adjustment (CVA) requires the simulation of future market moves, then pathwise valuation of each future counterparty portfolio. The Margin Value Adjustment (MVA) requires even more calculation such as pathwise sensitivities. Recent development highlight the usefulness of IT techniques such as of GPU programming but also Machine learning to improve the efficiency of such calculation.

In this note, we will focus on the CVA calculation and show how can Machine Learning be applied as a model of future portfolio values to improve the computational efficiency. In a first part, we briefly present the CVA framework and how it can be numerically estimated. In a second part, we briefly present the state of the art of CVA calculation improvement and then our detailed Machine learning approach with model chosen. In a last part, we test these Machine learning models on interest rate swap and swaption portfolios.

1 The credit valuation adjustment framework

1.1 Definition

The credit valuation adjustment represents the difference between the risk-free valuation of a portfolio or a contract and the valuation which account for the probability of default of each counterparty. Computing the CVA then involves a model of the counterparty default. Let's develop it with some more formalism. We refer the reader to the references for a more detailed framework.

Let's consider a pricing probability basis $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{Q})$. $\mathbb{F} = \mathcal{F}_{t\in[0,T]} \subseteq \mathcal{F}$ is a filtration satisfying the usual assumption such that all processes of interest are $\mathbb{F}$ adapted and τ, a random time representing the default time of the counterparty, is a $\mathbb{F}$-stopping time (we ignore here the default of the bank). We denote $\mathbb{E}_t(.) = \mathbb{E}(.|\mathcal{F}_t)$

Let's fix the trading time horizon T (the latest maturity in the portfolio for instance) and assume τ follows a stochastic intensity process γ. In the case there is no collateral management, which will be the case in this work, the time t_0 CVA can be defined as follows:

$$CVA_0 = (1-R)\mathbb{E}_0^{\mathbb{Q}} \int_0^T D_t V_t^+ \delta_\tau(dt)$$

$$= (1-R)\mathbb{E}_0^{\mathbb{Q}} \int_0^T D_t V_t^+ e^{-\int_0^t \gamma_s ds} \gamma_t dt$$

where R is the recovery rate, $D_t = D(0,t)$ is the discount process, $V_t = V(t, X_t)$ the portfolio value as a deterministic function of t and risk factors X_t and $V_t^+ = max(V_t, 0)$

In the case where the default and the portfolio value (expressed in numeraire unit) are independent (no Wrong Way risk) the expression can be simplified as :

$$CVA_0 = (1-R)\int_0^T \mathbb{E}_0^{\mathbb{Q}}[D_t V_t^+] p(t) dt \quad (1)$$

where $p(t)$ is the probability density function of τ. The expectation in the integral is the positive expected exposure. We will make this assumption afterwards.

1.2 Numerical estimation

The precedent formula can be estimated using the Monte-Carlo method: simulating M paths along which each risk factor is sampled and generating a time grid $t_1 < ... < t_N = T$, we have the following approximation :

$$CVA_0 \approx \frac{1}{M}(1-R)\Delta t \sum_{j=0}^{M} \sum_{i=1}^{N} D_{t_i}^{(j)} V(t_i, X_{t_i}^{(j)})^+ \Delta p_i \quad (2)$$

where $\Delta p_i = \mathbb{P}(t_{i-1} < \tau \leq t_i)$

The main challenge in computing the above quantity is the portfolio value $V(t_i, X_{t_i})^+$ which in many cases cannot be evaluated analytically. For instance, in the case of a portfolio involving high dimensional multi-callable products, a full reevaluation of such portfolio would lead in a computationally intensive nested Monte-Carlo simulation.

An usual approach to simplify the calculation is to use a Monte-Carlo regression (LSMC) algorithm instead (see ref [3]). In this method the portfolio future values (here the conditional expectation values filtered on information available at the corresponding time) are approximated by the means of regression functions fitted by backward induction on simulated paths. This method is however sensitive to the accuracy of the regressions on all simulated paths and has little error control, especially for high dimensional underlying models. Some improvements exist on this approach as in reference [4] and [9] for instance.

Another approach we can found in literature is to consider the BSDE formulation of the portfolio dynamic (prefered from the PDE one in high-dimension) which can be expressed in numeraire unit as (Ref 5):

$$d\bar{V}_t = \sum_{ij} \frac{\partial \bar{V}}{\partial X_i}(t, X_t)\sigma_{ij}(t, X_t) dW_t^j \quad (3)$$

with terminal conditions $\bar{V}(T^+)$, σ being the volatility terms of the risk factors dynamic following a multi-dimensional Brownian motion. This equation is then numerically solved by Euler discretization and backward or forward induction. The positive expected exposures are computed afterward. This approach is used by authors in [5].

2 The Machine learning approach

2.1 Introduction

The use of Machine learning in quantitative finance and especially in risk management has recently seen increasing inter-

est from both academics and practitioners. Also the literature that treats the application of Machine learning to xVA calculation has grown. For example, the authors in [1] improve the Monte Carlo estimation of the CVA by approximating the portfolio future values (the mark to market cube) by means of Gaussian process regressions. In [2], the authors introduce neural networks to approximate MtM values and sensitivities of a Bermudan swaption as part of IM and MVA calculation. The authors of [5] solve the BSDE representation (3) with a parameterization of the control process (the deltas) by a set of neural networks. In [6], the same approach is extended to all xVA dynamics in a generic framework by its authors .

In this note, we consider the analytical CVA formula (1) with its Monte Carlo scheme (2) as a starting point and as in [1] or [2], we train a machine learning algorithm to approximate the portfolio point values (MTM cube) using synthetic data from pricing models. Our attempt is to avoids one level of nested simulation while preserving as much as possible the full quality of the original pricer.

In a previous note, we have shown that machine learning models such as neural networks and Gaussian process regression were suitable for approximating option prices as well as the whole implied volatility surface. While Neural networks have shown a better ability to scale when input dimension increase (both in feature and sample size), Gaussian process regression performed well when the input size is small and provides a way to assess the uncertainty of the prediction. Thus, the idea is to reuse and adapt these models as part of the Monte-Carlo estimation of the CVA.

2.2 Modified Monte-Carlo estimation

The approximation of portfolio values within the Monte-Carlo loop can be done in several ways. Approximating the whole portfolio value using a single model, apply several models to approximate each contract separately or use a single multi-output prediction model that approximates the vector value of all contracts. Then compute the aggregated portfolio. This last approach has both the advantage to account for portfolio where contracts share common risk factors with correlations between them, which is often the case in practice and preserve the flexibility to dynamically rebalance the portfolio without the need to retrain the model.

In this approach, the calculation of the CVA can be summarized in the following steps :

- Generate a supervised training set of underlying risk factors and trade properties as features, and the corresponding d-vector valued contract prices of interest as labels $\mathcal{D} = \{(X_i, v(X_i)) \in \mathbb{R}^m \times \mathbb{R}^d,\ i = 1, .., n\}$.
- Train the machine model f to approximate the vector-valued contract prices $\widehat{v}(X_i) = f(X_i, \theta_{\mathcal{D}})$. Denoting $\theta_{\mathcal{D}}$ the model parameters optimized on $\mathcal{D}$.
- Compute the CVA using the Monte-Carlo method :

$$CVA_0 \approx \frac{(1-R)}{M} \Delta t \sum_{j=0}^{M} \sum_{i=1}^{N} D_{t_i}^{(j)} w^T f(X_{t_i}^{(j)}, \theta_{\mathcal{D}})^+ \Delta p_i$$

where w is the vector of portfolio weights

In the case of a high dimensional portfolio sharing multiple asset classes, the question of considering subportfolios arises. Training one ML model per asset class or netting set should be more adapted and efficient in practice.

The first step is done by the means of a sampling methods and pricing models. This is generally a computing intensive task for both feature generation (particularly when the dimension of the data is large) as well as for generating contract values. Yet this step is executed once in an outer simulation and the trained ML model is then reusable. Another point to note is that we assume the underlying stochastic models are already calibrated. In a production-like environment, machine learning models might need to be retrained at each recalibration phase. Yet, a possibility to avoid retraining is to enlarge the training set with the underlying model parameters sampled on a sufficiently wide domain and use online learning if it is applicable.

2.3 Neural networks

The detailed principle of Neural networks was developed in a previous note, but we recall the main lines :

Neural network such as Multi-Layer perceptrons (MLP) are a composition of multiple connected layers, each one representing a linear transformation plus an activation function. The outputs of the layer l can be expressed as :

$$a^l = \sigma_l(W_l a^{l-1} + b_l)$$

where $W_l \in \mathbb{R}^{n_{l-1}} \times \mathbb{R}^{n_l}$ is the weights matrix and $b_l \in \mathbb{R}^{n_l}$ is the bias vector.

The training consists in finding the parameters $\Theta = (W_1, .., W_L, b_1, .., b_L)$ which minimize a given loss function $\mathcal{L}$:

$$\underset{\Theta}{\operatorname{argmax}}\ \mathcal{L}(Y, \mathcal{NN}(X, \Theta))$$

For doing so, we use a mini-batch gradient descent algorithm were weight and bias are iteratelly updated by the calculation of the loss function on batches of training data and backpropagation principle. Hyperparameters such as the learning rate controls the learning process speed.

The use of neural networks for approximating derivative prices is justified by the Universal approximation theorem which indicates that any continuous function on a compact subset of $\mathbb{R}$ can be approximated by a single hidden layer feed forward neural network. Moreover, despite its lack of interpretation NN have the advantage of providing a hidden representation of its inputs.

2.4 Multi-response Gaussian process regression

In the previous note on option pricing , we have detailed the principle of Gaussian Process regression in the case of a single output. We develop here the case of multi-output GPR more adapted for our approach.

Multi-output GPR relies on multi-response Gaussian process.

Definition 2.1 (Multivariate Gaussian process). A function f is a d-variate Gaussian process on χ with vector-valued mean function $\mu : \chi \to \mathbb{R}^d$, kernel function $k : \chi \times \chi \to \mathbb{R}$ and positive semi-definite parameter matrix $\Omega \in \mathbb{R}^{d\times d}$ denoted $f \sim \mathcal{MGP}(\mu, k, \Omega)$ if any finite collection of vector-valued variables $f(x_1), ..., f(x_n)$ has a joint multivariate Gaussian distribution :

$$[f(x_1), .., f(x_n)] \sim \mathcal{N}(M, \Sigma \otimes \Omega)$$

with

$$M \in \mathbb{R}^{d\times n},\ M_{k,i} = \mu_k(x_i), \quad \Sigma \in \mathbb{R}^{n\times n},\ \Sigma_{i,j} = k(x_i, x_j),$$

$$\Sigma \otimes \Omega = \begin{pmatrix} \Sigma_{1,1}\Omega & \cdots & \Sigma_{1,n}\Omega \\ \vdots & \ddots & \vdots \\ \Sigma_{m,1}\Omega & \cdots & \Sigma_{m,n}\Omega \end{pmatrix}$$

Given a set of observations $\{(X_i, Y_i) \in \mathbb{R}^m \times \mathbb{R}^d,\ i = 1, .., n\}$, the Multi-response Gaussian process regression can be expressed as

$$Y_i = f(X_i) + \epsilon,\ i \in \{1, .., n\}$$

with:

- $[f(X_1), ..., f(X_n)]^T \sim \mathcal{N}(0, K(X, X) \otimes \Omega)$, $[K(X, X)]_{i,j} = k(X_i, X_j)$ a $n \times n$-matrix
- $Y \sim \mathcal{MGP}(\mu, k', \Omega)$, $k'(X_i, X_j) = k(X_i, X_j) + \delta_{ij}\sigma^2$, σ being the variance of the additive Gaussian i.i.d noise ϵ

kernels

The kernel k can be any positive semidefinite function. Radial basis functions (RBF) which only depend on $||x - x'||$ are a commonly used Kernels as they are universal: two output points close in input space are highly correlated. Other more elaborate Kernels such as Matern Kernel (see ref [1]).

training and inference

Given a prior distribution on $f(X) = f([X_1, ..., X_n])$, the training of the GPR consists in finding the hyperparameters θ (parameters of the kernel function) and the element of the covariance matrix Ω that minimize the marginal log-likelihood of training samples :

$$\mathcal{L}(Y|X, \theta, \Omega) = \frac{nd}{2} ln(2\pi) + \frac{d}{2} ln|K'(X, X)| + \frac{n}{2} ln|\Omega|$$

$$+\frac{1}{2} tr(K'(X, X)^{-1} Y \Omega^{-1} Y^T)$$

For a new set of observed samples X^*, the joint distribution of the training targets $Y = [Y_1, .., Y_n]$ and the predictive targets f^* is :

$$\begin{bmatrix} Y \\ f^* \end{bmatrix} = \mathcal{MN}\left(0, \begin{bmatrix} K'(X, X) & K(X, X^*)^T \\ K(X^*, X) & K(X^*, X^*) \end{bmatrix}, \Omega\right)$$

Then we have the following conditional distribution for f^*:

$$f^*|X^*, X, y \sim \mathcal{N}(\widehat{M}, \widehat{\Sigma} \otimes \widehat{\Omega})$$

$$\widehat{M} = K(X^*, X)^T K'(X, X)^{-1} Y$$
$$\widehat{\Sigma} = K(X^*, X^*) - K(X^*, X)^T K'(X, X)^{-1} K(X^*, X)$$
$$\widehat{\Omega} = \Omega$$

We have the point estimate for the portfolio value $\widehat{V}(X) = w^T \widehat{M}$ and covariance estimator $\widehat{\Sigma}$ which can be used for model uncertainty.

To be noted that if the training computational time scale poorly on the training sample size n with an $O(n^3)$ complexity (this is due to the Cholesky decomposition of the $n \times n$-matrix K), the prediction is performed in $O(n^2)$.

online learning for model update

Online learning is possible for GPR. Basically, the model update relies on the following Bayesian formulas: for a new observation (X', Y') we have

$$p(f^*, X, Y, X', Y', X^*) = \frac{p(X', Y'|f^*)p(f^*|X, Y, X^*)}{\int_Z p(X', Y'|z)p(z|Z, Y, X^*)dz}$$

as the new posterior on the test sample X^* and the previous posterior become the prior in the update ($f^* \in Z$).

As suggested in [1], this can be useful to update the model without whole retraining when underlying models are often recalibrated (in intraday for instance). In this case, underlying model parameters are added to input features, then, at each recalibration phase, a new set of prices is generated and the posterior updated.

3 Numerical experiments

In this section, we apply the machine learning approach for a set of interest rate swap portfolio first then a portfolio of swaptions. More insights about the financial models can be found in Appendix.

In our approach we train one machine learning model per time step of a primarily defined time grid of future scenarios as it was found to be more efficient in practice. In order to have a realistic realization of the market (interest rate curves), the training data is generated using a Monte-Carlo simulation. Only the interest rate value is thus considered as feature (in fact we use rather the state variable x defined in Appendix).

3.1 Case I: Interest rate Swap portfolio

We construct a portfolio of both payer and receiver interest rates swaps on the Euribor 6M with different maturities and nominals (4 to 6 year maturities and 1 to 7.10^5 range nominals. We assume a flat yield term structure for simplicity. We use the [6M, 1Y, .., 10Y] yield curve for discounting. All prices of interest are computed using the Quantlib library functions.

We model the survival probability $p(t) = e^{-\int_0^t \lambda(s)ds}$ with a deterministic piecewise-constant hazard rate function λ. The default curve is then built by interpolation on our time grid.

Machine learning models are implemented in Python using the Torch module. We use a Radial basis function as kernel

of the Gaussian process regression. If the hyperparameter optimization is automatically done during the training process for GPR, it is however tricky for the neural networks due to the high number of models (one per time step) and the pricing map which highly depends on the time to maturity. In a sake of simplicity and limited computer resources, the NN architecture were only tuned for one particular time step and using 500 train samples. For doing so, a grid space of hyperparameters choice is first defined. It includes 2 or 3 hidden layers, and hidden size comprised between 40 and 120 units. 20% of training data were used for validation. The final selected NN have a shape (50,80,70) for the two dimensional portfolio.

The CVA calculation is tested by varying the training size (the number of paths generated for training data) and portfolio size. We generate 5000 future scenarios on a 6 year-long time grid with monthly time step (which also include fixing dates. A Monte-Carlo loop with full revaluation is used as a Benchmark. The graph below shows expected positive exposures for a two swaps portfolio (receiver and payer with 4Y and 5Y maturities) :

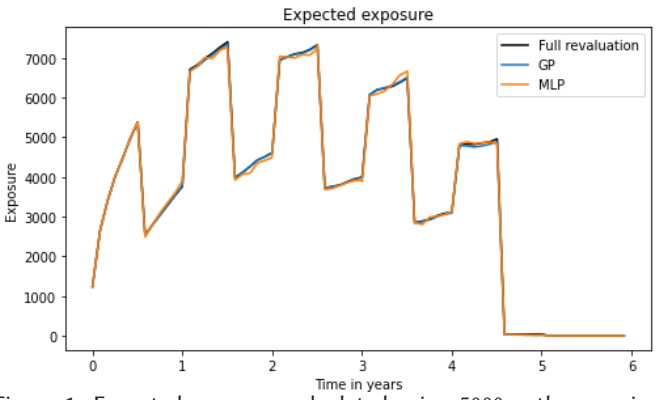

Figure 1: Expected exposure calculated using 5000 path scenarios and 2000 training sample per time step for ML models

The table below summarizes the different tested cases for the CVA calculation compared to the full revaluation case:

CVA_0 values for 5000 scenarios and 77 time step					
portfolio size	training size	model	CVA	CVA reval	diff%
d=2	N=500	GPR	609.17	613.2	0.657
		MLP	611.485	613.2	0.28
	N=1000	GPR	610.053	613.2	0.513
		MLP	608.80	613.2	0.717
	N=2000	GPR	610.96	613.2	0.366
		MLP	610.458	613.2	0.447
d=6	N=500	GPR	712.742	713.756	0.142
		MLP	714.63	713.756	0.123

The train size has not a great impact on final precision and 500 training data point is sufficient to have a correct accuracy. In contrast, GPR predictions are more stable than the NN. The time execution of the MtM cube generation is divided by 6 with the GPR and by 55 with the NN.

3.2 Case II: Swaptions portfolio

We now test the method for a portfolio of swaptions, we detail here the case of a single Bermudan swaption portfolio with 5 annual exercise dates on an underlying payer swap with 5Y maturity and 10^6 nominal. In this case, we also compare the Machine learning approximation with the least square Monte-Carlo method detailed in appendix. As Bermudan swaption are callable, we need to explicitly simulate the exercise decision. Thus, the pathwise values of the underlying swap need to be estimated too. We train at each time step two ML models, one to approximate the swaption NPV and the second to approximate the Swap NPVs. The training sets are generated again by Monte-Carlo simulations and PDE pricing for the swaption. Again, We also compare the results against the case of the PDE full revaluation. We generate 1500 future scenarios of the short rate on the same time grid as previously.

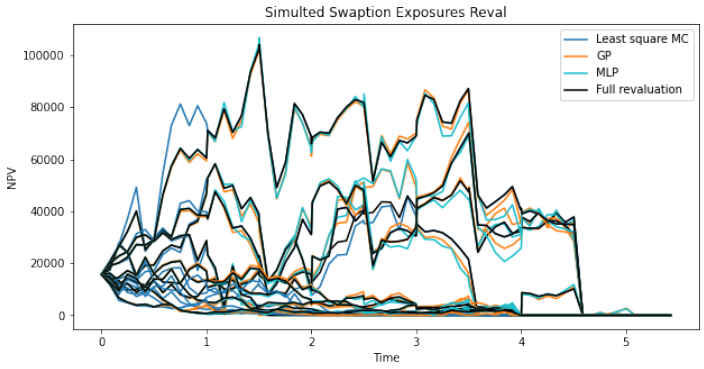

Figure 2: Positive future exposures for 10 paths

One of the difficulties for the case of such multi-callable product is the sensitivity to the accuracy of the exercise decision function approximation. Indeed, error in the estimated stopping time $1(\hat{S}(t) > \hat{V}(t))$ (denoting V and S the value of the swaption and the exercise)leads to a large discrepancy in the future exposure estimation (Ref[2]) as the portfolio value can jump. An alternative would be to also learn the optimal decision $\hat{1}(S(t) > V(t))$ by a third model at every call time point in the future. In our work, the use of a classification NN for the decision function improved the accuracy of the CVA calculation.

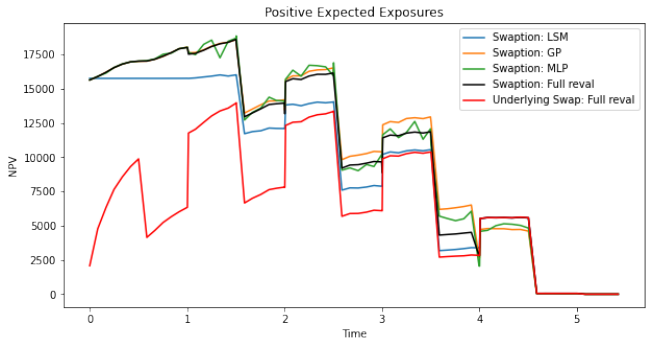

Figure 3: Positive expected exposures

CVA_0 **for 1500 scenarios**			
model	CVA	diff%	MtM exec. time
Full revaluation	1452.95		35min
LSM	1306.0	10.12	0.27s
GPR	1501.43	3.33	6.5s
MLP	1465.67	0.87	0.17s

Conclusion

In this note, we have discussed the application of Machine learning as an efficient way to approximate the MtM cube for the Monte-Carlo estimation of the CVA. Both Neural networks and Gaussian process regression were introduced as selected Machine models to address the problem as they have some particular advantages: hidden representation and dimensional scalability for neural networks, Bayesian model uncertainty for GPR. We have applied these methods for some simple portfolios of common IR products in a basic setup (no collateral or Wrong Way risk, single curve environment and single factor model). The results are promising in term of precision and scalability in the case of swap portfolios. The optimization of hyperparameters for NN is tricky, but some improvements can be made in this direction. Besides, our models were executed on CPU while a GPU execution (possible for both models) would be even more fast.

Other interesting features of the proposed Machine learning models could be exploited such as fast gradient computation for CVA pathwise sensitivities. For instance, GPR provides an analytical way of computing the gradient with respects to its input while the gradient can also be extracted from NN with the backpropagation principle.

4 Appendix

In this section, we briefly present the financial modelling aspect and pricing used for the numerical applications. More details can be found in [8] and [9].

The one factor Gaussian model (G++)

The short rate dynamic under the risk neutral measure in the one factor Gaussian model is:

$$dr(t) = \kappa(t)(\theta(t) - r(t))dt + \sigma(t)dW^{\mathbb{Q}}(t)$$

σ and κ are considered piecewise constant.

For practical purpose, we usually consider rather the dynamic of the quantity $x(t) := r(t) - f(0,t)$, ($f(t,T)$, $t < T$ being the instantaneous forward rate), which is:

$$dx(t) = (y(t) - \kappa(t)x(t))dt + \sigma(t)dW^{\mathbb{Q}}(t)$$

with $y(t) = \int_0^t e^{-2\int_s^t \kappa(u)du}\sigma^2(s)ds$. In the T-forward measure, the dynamics becomes:

$$dx(t) = (y(t) - \sigma^2(t)G(t,T) - \kappa(t)x(t))dt + \sigma(t)dW^{\mathbb{Q}^{\mathbb{T}}}(t)$$

with $G(t,t') = \int_t^{t'} e^{\int_t^u \kappa(s)ds}dt$

The price of a zero coupon bond can then be expressed as :

$$P(t,T) = \frac{P(0,T)}{P(0,t)}e^{-x(t)G(t,T) - \frac{1}{2}y(t)G^2(t,T)}$$

Zero-coupon prices can thus be computed for scenarios yield curve construction and pricing purpose.

Swap price

A vanilla interest rate swap consists of an exchange between a floating and a fixed rate. Let's denote $T_1, .., T_N$ the settlement dates of the swap. If we place in $t \in [T_{n-1}, T_n[$, the cash flow of a payer swap with remaining dates $T_n, .., T_N$ (tenor $T_N - T_n$), where the party receive the floating rate L and pay the fixed rate R, is given by the difference between the floating leg and fixed leg cashflow values :

$$S(t, X_t) = \sum_{k=n+1}^{N} \Delta_k E_t^{\mathbb{Q}^{T_k}}[L(T_{k-1}, T_k)]P(t,T_k) - R\Delta_k P(t,T_k)$$

The expectation in the formula is the simply-compounded forward rate which can be expressed in term of zero coupon prices and accrual period $\Delta_k := T_{k+1} - T_k$:

$$E_t^{\mathbb{Q}^{T_k}}[L(T_{k-1}, T_k)] = \frac{1}{\Delta_k}\Big(\frac{P(t,T_{k-1})}{P(t,T_k)} - 1\Big)$$

These formulas enable to calculate the pathwise price of the swap. The function **DiscountingSwapEngine** of Quantlib implement the pricer and can be linked to a provided term structure.

Swaption price

A Swaption is an option which enables the owner the possibility to enter an underlying swap at the expiry time (European case) or at multiple exercise dates (Bermudan case). In the single factor model, the price $V(t, x(t))$ of the swaption verifies the Feynman-Kac pricing PDE :

$$\frac{\partial V}{\partial t} + (y(t) - \kappa x(t))\frac{\partial V}{\partial x} + \frac{1}{2}\sigma(t)\frac{\partial^2 V}{\partial x^2} = (x(t) + f(0,t))V$$

subjects to the terminal condition $V(T, X_T) = \max(0, S(T, X_T))$

The Quantib function **Gaussian1dSwaptionEngine** provides a PDE sovler using Finite-differences.

American Monte-Carlo for Bermuda swaption exposures

In the case of Bermudan, assuming the exercise dates coincide with the payment dates of the underlying swap, the pricing problem can be expressed as an optimal stopping problem on exercise dates $\mathcal{T} = \{T_1, ..., T_{N-1}\}$:

$$V(t, X_t) = \max_{\tau \in \mathcal{T}} N_t E_t^{\mathbb{Q}^N}[N_\tau^{-1} S(\tau, X_\tau)]$$

denoting N the numeraire. If $\{t_0 < ... < t_M = T_{N-1}\}$ are the dates at which we want to compute the exposure (which include call dates), we have if $t_m = T_n < T_N$ is a call date:

$$V(t_m, x) = max\{S(t_m, x)^+, \underbrace{N_{t_m}\mathbb{E}^{\mathbb{Q}^N}(N_{t_{m+1}}^{-1}V(t_{m+1}, X_{t_{m+1}})|X_{t_m} = x)}_{C(t_m,x)}\}$$

where C is the continuation value.
At any time $T_n < t_m < T_{n+1}$ we have $V(t_m, x) = C(t_m, x)$ and at the expiry: $V(T_{N-1}, x) = max\{S(T_{N-1}, x), 0\}$.

The Least square Monte carlo (LSM) consists in approximating the continuation value by first simulating risk neutral market realization paths $(X_{t_0}^{(j)}, ..., X_{t_M}^{(j)})_{j=1}^{\mathcal{J}}$. Then, proceed backward-in-time from the known terminal conditions, fitting at each step t_m a regression function f_m in order to minimize the following square error :

$$\sum_{j=1,\, ITM}^{\mathcal{J}} \Big[\frac{N_{t_m}^{(j)}}{N_{t_m+1}^{(j)}} V(t_{m+1}, X_{t_{m+1}}^{(j)}) - f_m(X_{t_m}^{(j)})\Big]^2$$

where path future values $V(t_{m+1}, X_{t_{m+1}}^{(j)})$ are known from previous step. Originally, the regression functions are linear regressions on a set of polynomial basis functions ϕ_k. That is we have $f_m(x) = \sum_{k=1}^{R} \beta_k(x)\phi_k(x)$. Then, the path continuation value is approximated as $C(t_m, X_{t_m}^{(j)}) \approx \mathbb{E}^{\mathbb{Q}^N}[f(X_{t_m})|X_{t_m}^{(j)}] = f(X_{t_m}^{(j)})$ and the present values $V(t_m, X_{t_m}^{(j)})$ can be computed according to previous formulas.
The t_0 value is $V(0, X_{t_0}) = \frac{N_{t_0}}{\mathcal{J}} \sum_j C(t_0, X_{t_0}^{(j)})$

Finally, the exposure profile of the swaption is computed along each path forward-in-time as:

$$E(t_m, X_{t_m}^{(j)}) = \Big[1_{t_m < \tau_j^*} V(t_m, X_{t_m}^{(j)}) + 1_{t_m \geq \tau_j^*} S(t_m, X_{t_m}^{(j)})\Big]^+$$

where $\tau_j^* = \inf\{\tau \in \mathcal{T} | S(\tau, X_\tau^{(j)}) \geq V(\tau, X_\tau^{(j)})\}$.

An alternative and improvement of the LSM method is the Stochastic grid bundling (SGB) where the future value V is approximated given time t_{m+1} state variables instead and by several local regressions done on bundles of paths. We refer the reader to reference [9] for details.

References

[1] Matthew F. Dixon Stéphane Crépey. Gaussian process regression for derivative portfolio modeling and application to cva computations. 2019.

[2] Allan Cowan Anup Aryan. Deep mva: Deep learning for margin valuation adjustment of callable products. *IHS Markit, Financial Risk Analytics*, 2020.

[3] Serguei Issakov Alexandre Antonov and Serguei Mechkov. Backward induction for future values. *Risk Magazine*, 2015.

[4] Oh Kang Kwon Mark S. Joshi. Least squares monte carlo credit value adjustment with small and unidirectional bias. 2016.

[5] Dan Grecu Jian-Huang She. Neural network for cva: Learning future values. 2018.

[6] Christoph Reisinger Alessandro Gnoatto, Athena Picarelli. Deep xva solver – a neural network based counterparty credit risk management framework. 2020.

[7] Bo Wang Zexun Chen and Alexander N. Gorban. Multivariate gaussian and student−t process regression for multi-output prediction. 2019.

[8] Peter Caspers. One factor gaussian short rate model implementation. 2013.

[9] Patrik Karlsson Drona Kandhai Qian Feng, Shashi Jain and Cornelis W. Oosterlee. Efficient computation of exposure profiles on real-world and risk-neutral scenarios for bermudan swaptions. 2016.

[10] Matthias Groncki. Jupyter notebooks – a swiss army knife for quants.

A propos de Coperneec

« From Revolution to Performance »

Coperneec est un cabinet de conseil cross-sectoriel spécialiste de la valorisation de la Data et du développement IT.

Nos méthodes et techniques scientifiques éprouvées permettent de résoudre des problématiques dans tous les secteurs de l'industrie.

Notre vocation : extraire la connaissance à partir des données et pérenniser les avancées technologiques qui en découlent. La R&D est au cœur de notre ADN et les expertises de nos consultants sont en permanence challengées afin d'accompagner au plus près les révolutions technologiques et scientifiques.

Contactez-nous

Aymeric LISBONNE
Partner
alisbonne@coperneec.com
06 88 69 67 75

coperneec

est une marque de

canopee group