Journal of Mathematics in Industry
a SpringerOpen Journal

RESEARCH Open Access

An SGBM-XVA demonstrator: a scalable Python tool for pricing XVA

Ki Wai Chau[1*], Jok Tang[2,3] and Cornelis W. Oosterlee[1,4]

*Correspondence:
kiwaic0108@gmail.com
[1]Delft Institute of Applied Mathematics, Delft University of Technology, Delft, The Netherlands
Full list of author information is available at the end of the article

Abstract
In this work, we developed a Python demonstrator for pricing total valuation adjustment (XVA) based on the stochastic grid bundling method (SGBM). XVA is an advanced risk management concept which became relevant after the recent financial crisis. This work is a follow-up work on Chau and Oosterlee in (Int J Comput Math 96(11):2272–2301, 2019), in which we extended SGBM to numerically solving backward stochastic differential equations (BSDEs). The motivation for this work is basically two-fold. On the application side, by focusing on a particular financial application of BSDEs, we can show the potential of using SGBM on a real-world risk management problem. On the implementation side, we explore the potential of developing a simple yet highly efficient code with SGBM by incorporating CUDA Python into our program.

Keywords: SGBM; XVA; CUDA Python

1 Introduction

Backward stochastic differential equations (BSDEs) have been a popular research subject ever since the work of Pardoux and Peng, [2] and [3]. There are many papers regarding for their applications in mathematical finance and stochastic control. In terms of numerical analysis, there is also significant research on the efficient calculation or approximation for BSDEs. This includes Monte Carlo-based research, like [4], chaos decomposition method [5], cubature methods [6] or Fourier and wavelet-based methods, like in [7] and [8]. However, there are relatively few studies on the practical application of BSDEs. As far as we know, there is not yet an industrial software package for solving general BSDEs. This work is our first step to address this issue.

The goal of this study is basically two-fold. This work can be seen as a follow-up of our theoretical research on the stochastic grid bundling method (SGBM) for BSDEs, see [1]. SGBM is a Monte Carlo-based algorithm which was introduced in [9], and extended its application to BSDEs in [1]. Here, we will study the practical side by developing a demonstrator in Python, where we shall also make use of computing on a Graphics Processing Unit (GPU) in order to improve the scalability, and make use of the CUDA Python package. CUDA Python is a recently open to public programming tool, which was developed by Anaconda. It has become freely available at the end of 2017, being previously commercial software. This programming tool carries the promise of combining fast development time for Python coding with the high efficiency of GPU computing.

The second focus of this work is the application of BSDEs in financial risk management, where we would like to demonstrate the practical opportunities for efficient BSDE solving software. We choose the modeling of the variation margin and the close-out value within risk management, in the form of BSDEs, as the main test problem. In this work, we work under a complete market assumption which includes counterparty risk and margin requirements, and develop a numerical solver for XVA.

A Python demonstrator for solving XVA pricing problems with the SGBM algorithm with GPU computing is the main result of this study. The strength of this package is its scalability with respect to the dimensionality of the underlying stock price processes. While the demonstrator is designed for a specific problem setting, because of the general framework with BSDEs and SGBM, the package could easily be transformed into a general solver for BSDEs and used for other financial problems.

This article is organized as follows. In Sect. 2, we introduce the basic mathematical setting of BSDEs, the fundamental properties of the SGBM algorithm and the application of parallel computing with SGBM. Section 3 describes the programming language, the financial setting for our SGBM-XVA demonstrator, and other technical details, while some numerical tests are performed in Sect. 4. Concluding remarks, possible extensions and outlook are given in Sect. 5.

2 Methodology

We begin this section with a brief review of the mathematical background of BSDEs and of the Monte Carlo-based simulation method SGBM. For further details, the readers are referred to our previous work [1]. Furthermore, we will describe a parallel computing version of SGBM, which is a follow-up on [10], in which parallel SGBM was applied to the pricing of multi-dimensional Bermudan options.

2.1 Backward stochastic differential equations

We use a standard filtered complete probability space $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$, where $\mathbb{F} := (\mathcal{F}_t)_{0\leq t\leq T}$ is a filtration satisfying the usual conditions for a fixed terminal time $T > 0$. The process $W := (W_t)_{0\leq t\leq T}$ is a d-dimensional standard Brownian motion, adapted to the filtration $\mathbb{F}$ and the so-called decoupled forward-backward stochastic differential equation (FBSDE) defines a system of equations of the following form:

$$\begin{cases} dS_t = \mu(t,S_t)\,dt + \sigma(t,S_t)\,dW_t; \\ dV_t = -g(t,S_t,V_t,Z_t)\,dt + Z_t\,dW_t, \end{cases} \tag{1}$$

where $0 \leq t \leq T$. The functions $\mu : [0,T] \times \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : [0,T] \times \mathbb{R}^d \to \mathbb{R}^{d\times d}$ are referred to as the drift and the diffusion coefficients of the forward stochastic process S, respectively, and $s_0 \in \mathcal{F}_0$ is the initial condition for S. The function $g : [0,T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^{1\times d}$ is called the driver function of the backward process and the terminal condition V_T is given by $\mathfrak{N}(S_T)$, for a function $\mathfrak{N} : \mathbb{R}^d \to \mathbb{R}$. All stochastic integrals with Wiener process W are of the Itô type.

For both $\mu(t,s)$ and $\sigma(t,s)$, we assume standard conditions such that a unique strong solution for the forward stochastic differential equation exists,

$$S_t = s_0 + \int_0^t \mu(\tau, S_\tau)\,d\tau + \int_0^t \sigma(\tau, S_\tau)\,dW_\tau.$$

This process also satisfies the Markov property, where $\mathbb{E}[S_\tau|\mathcal{F}_t] = \mathbb{E}[S_\tau|S_t]$, for $\tau \geq t$, where $\mathbb{E}[\cdot]$ denotes the expectation operator with respect to probability measure $\mathbb{P}$.

A pair of adapted processes (V,Z) is said to be the solution of the FBSDE, if V is a continuous real-valued adapted process, Z is a real-valued predictable row vector process, such that $\int_0^T \|Z_t\|^2\,dt < \infty$ almost surely in $\mathbb{P}$, where $\|\cdot\|$ denotes the Euclidean norm, and the pair satisfies Equation (1).

Our goal is to find (V_0, Z_0) by solving the problem backward in time. We do this by first discretizing Equation (1) along the time-wise direction, $\pi: 0 = t_0 < t_1 < t_2 < \cdots < t_N = T$. We assume a fixed, uniform time-step, $\Delta t = t_{k+1} - t_k, \forall k$, and let $\Delta W_{k+1,q} := W_{t_{k+1},q} - W_{t_k,q} \sim \mathcal{N}(0,\Delta t)$, a normally distributed process, for $q = 1,\ldots,d$. The vector ΔW_{k+1} is defined as $(\Delta W_{t_{k+1},1},\ldots,\Delta W_{t_{k+1},d})^\top$. The discretized forward process S^π is defined by

$$S^\pi_{t_0} := s_0, \qquad S^\pi_{t_{k+1}} := S^\pi_{t_k} + \mu\bigl(t_k, S^\pi_{t_k}\bigr)\Delta t + \sigma\bigl(t_k, S^\pi_{t_k}\bigr)\Delta W_{k+1}, \quad k = 0,\ldots,N-1.$$

It is defined by the classical Euler–Maruyama discretization.

We define a discrete-time approximation (Y^π, Z^π) for (Y,Z):

$$\begin{aligned} &V^\pi_{t_N} := \mathfrak{N}\bigl(S^\pi_{t_N}\bigr), \qquad Z^\pi_{t_N} = \nabla\mathfrak{N}\bigl(S^\pi_{t_N}\bigr)\sigma\bigl(t_N, S^\pi_{t_N}\bigr), \\ &\quad \text{for } k = N-1,\ldots,0, \text{ and for } q = 1,\ldots,d, \end{aligned} \tag{2a}$$

$$\begin{aligned} Z^\pi_{t_k,q} :={}& -\frac{1-\theta_2}{\theta_2}\mathbb{E}_k\bigl[Z^\pi_{t_{k+1},q}\bigr] + \frac{1}{\theta_2\Delta t}\mathbb{E}_k\bigl[V^\pi_{t_{k+1}}\Delta W_{k+1,q}\bigr] \\ &+ \frac{1-\theta_2}{\theta_2}\mathbb{E}_k\bigl[g\bigl(t_{k+1}, S^\pi_{t_{k+1}}, V^\pi_{t_{k+1}}, Z^\pi_{t_{k+1}}\bigr)\Delta W_{k+1,q}\bigr], \end{aligned} \tag{2b}$$

$$\begin{aligned} V^\pi_{t_k} :={}& \mathbb{E}_k\bigl[V^\pi_{t_{k+1}}\bigr] + \Delta t\theta_1 g\bigl(t_k, S^\pi_{t_k}, V^\pi_{t_k}, Z^\pi_{t_k}\bigr) \\ &+ \Delta t(1-\theta_1)\mathbb{E}_k\bigl[g\bigl(t_{k+1}, S^\pi_{t_{k+1}}, V^\pi_{t_{k+1}}, Z^\pi_{t_{k+1}}\bigr)\bigr]. \end{aligned} \tag{2c}$$

The notation ∇ denotes the gradient of a function. Note that various combinations of θ_1 and θ_2 give different approximation schemes. We have an explicit scheme for V^π if $\theta_1 = 0$, and an implicit scheme otherwise.

2.2 Stochastic grid bundling method (SGBM)

We now introduce the Monte Carlo-based algorithm SGBM for BSDEs. The main difference between SGBM and other commonly known Monte Carlo algorithm, for example the one in [11], include a so-called *regress-later* regression stage and a so-called *equal-sized bundling* localization.

Due to the Markovian setting of $(S^\pi_{t_k}, \mathcal{F}_{t_k})_{t_k\in\pi}$ and the terminal processes $V^\pi_{t_N}$ and $Z^\pi_{t_N}$ being deterministic with respect to $S^\pi_{t_N}$, there exist functions $v_k^{(\theta_1,\theta_2)}(s)$ and $z_k^{(\theta_1,\theta_2)}(s)$ such that

$$V^\pi_{t_k} = v_k^{(\theta_1,\theta_2)}\bigl(S^\pi_{t_k}\bigr), \qquad Z^\pi_{t_k} = z_k^{(\theta_1,\theta_2)}\bigl(S^\pi_{t_k}\bigr),$$

for the given approximation in Equations (2a)–(2c). Our algorithm estimates these functions, $(y_k^{(\theta_1,\theta_2)}(s), z_k^{(\theta_1,\theta_2)}(s))$, recursively, backward in time, by a local least-squares regression technique onto a function space with basis functions $(p_l)_{0\leq l\leq Q}$.

As a Monte Carlo-based algorithm, our algorithm starts with the simulation of M independent samples of $(S_{t_k}^{\pi})_{0\leq k\leq N}$, denoted by $(S_{t_k}^{\pi,m})_{1\leq m\leq M, 0\leq k\leq N}$. Note that in this basic algorithm, the simulation is performed only once. Therefore, this scheme is a *non-nested* Monte Carlo scheme.

The next step is the backward recursion. At initialization, we assign the terminal, $t_N = T$, values to each path for our approximations, i.e.,

$$\begin{aligned}
&v_N^{(\theta_1,\theta_2),R,I}\big(S_{t_N}^{\pi,m}\big) = \mathfrak{N}\big(S_{t_N}^{\pi,m}\big),\\
&z_N^{(\theta_1,\theta_2),R}\big(S_{t_N}^{\pi,m}\big) = \nabla\mathfrak{N}\big(S_{t_N}^{\pi,m}\big)\cdot\sigma\big(t_N, S_{t_N}^{\pi,m}\big), \quad m = 1,\ldots,M.
\end{aligned}$$

The superscript R is used to distinguish the simple discretization and the discretization with SGBM, while I denotes the number of Picard iterations. The following steps are performed recursively, backward in time, at t_k, $k = N-1,\ldots,0$. First of all, we bundle all paths into $\mathcal{B}_{t_k}(1),\ldots,\mathcal{B}_{t_k}(B)$ equal-sized, non-overlapping partitions based on a sorting result of a bundling function defined on $(S_{t_k}^{\pi,m})$. This is the partition step.

Next, we perform the local regress-later approximation separately within each bundle. The regress-later technique we are using combines the least-squares regression with (analytically determined) expectations of the basis functions to calculate the necessary expectations.

Generally speaking, for any target function f and M Monte Carlo paths, a standard *regress-now* algorithm for a dynamic programming problem approximates a function ι within the space spanned by the regression basis such that it minimizes the value $\frac{1}{M}\sum_{i=1}^{M}(f(S_{t+\Delta t}^{\pi,i}) - \iota(S_t^{\pi,i}))^2$ and approximates the expectation $\mathbb{E}_t[f(S_{t+\Delta t}^{\pi})]$ by $\mathbb{E}_t[\iota(S_t^{\pi})] = \iota(S_t^{\pi})$. As a projection from a function of $S_{t+\Delta t}^{\pi}$ to a function of S_t^{π} is performed, it will introduce a statistical bias to the approximation.

Instead, the *regress-later* technique, as we employ in SGBM, finds a function κ which minimizes the functional $\frac{1}{M}\sum_{i=1}^{M}(f(S_{t+\Delta t}^{\pi,i}) - \kappa(S_{t+\Delta t}^{\pi,i}))^2$ and approximates the expectation $\mathbb{E}_t[f(S_{t+\Delta t}^{\pi})]$ by $\mathbb{E}_t[\kappa(S_{t+\Delta}^{\pi})]$. By using functions on the same variable, at $t+\Delta t$, in the regression basis, we can avoid the statistical bias in the regression. However, the expectation of all basis functions must preferably be known in closed-form, in order to apply the regress-later technique efficiently.

In the context of our BSDE SGBM algorithm, we define the bundle-wise regression parameters $\alpha_{k+1}(b)$, $\beta_{k+1}(b)$, $\gamma_{k+1}(b)$ as

$$\begin{aligned}
\alpha_{k+1}(b) &= \arg\min_{\alpha\in\mathbb{R}^Q} \frac{\sum_{m=1}^{M}(p(S_{t_{k+1}}^{\pi,m})\alpha - v_{k+1}^{(\theta_1,\theta_2),R,I}(S_{t_{k+1}}^{\pi,m}))^2\mathbf{1}_{\mathcal{B}_{t_k}(b)}(S_{t_k}^{\pi,m})}{\sum_{m=1}^{M}\mathbf{1}_{\mathcal{B}_{t_k}(b)}(S_{t_k}^{\pi.m})},\\
\beta_{i,k+1}(b) &= \arg\min_{\beta\in\mathbb{R}^Q} \frac{\sum_{m=1}^{M}(p(S_{t_{k+1}}^{\pi,m})\beta - z_{i,k+1}^{(\theta_1,\theta_2),R}(S_{t_{k+1}}^{\pi,m}))^2\mathbf{1}_{\mathcal{B}_{t_k}(b)}(S_{t_k}^{\pi,m})}{\sum_{m=1}^{M}\mathbf{1}_{\mathcal{B}_{t_k}(b)}(S_{t_k}^{\pi.m})},\\
\gamma_{k+1}(b)&\\
&= \arg\min_{\gamma\in\mathbb{R}^Q} \frac{\sum_{m=1}^{M}(p(S_{t_{k+1}}^{\pi,m})\gamma - g(t_{k+1}, v_{k+1}^{(\theta_1,\theta_2),R,I}(S_{t_{k+1}}^{\pi,m}), z_{k+1}^{(\theta_1,\theta_2),R}(S_{t_{k+1}}^{\pi,m})))^2\mathbf{1}_{\mathcal{B}_{t_k}(b)}(S_{t_k}^{\pi,m})}{\sum_{m=1}^{M}\mathbf{1}_{\mathcal{B}_{t_k}(b)}(S_{t_k}^{\pi.m})}.
\end{aligned}$$

Note that as we actually apply the equal partition bundling, $\sum_m^M = 1_{\mathcal{B}_{\mathbf{t_k}}(\mathbf{b})} = N$ for some constant N and for all bundle. Therefore we won't face the problem of dividing by zero.

The approximate functions within the bundle at time k are defined by the above parameters and the expectations $\mathbb{E}^x_{t_k}[p(S^\pi_{t_{k+1}})]$ and $\mathbb{E}^x_{t_k}[p(S^\pi_{t_{k+1}})\frac{\Delta W_{k,q}}{\Delta t}]$ read,

$$\begin{aligned}
z_{k,q}^{(\theta_1,\theta_2),R}(b,s) &= -\theta_2^{-1}(1-\theta_2)\mathbb{E}^s_{t_k}\big[p\big(S^\pi_{t_{k+1}}\big)\big]\beta_{k+1}(b) \\
&\qquad + \theta_2^{-1}\mathbb{E}^x_{t_k}\left[\frac{\Delta W_{k,q}}{\Delta t}p\big(S^\pi_{t_{k+1}}\big)\right]\big(\alpha_{k+1}(b) + (1-\theta_2)\Delta t\gamma_{k+1}(b)\big), \quad q = 1,\dots,d; \\
v_k^{(\theta_1,\theta_2),R,0}(b,s) &= \mathbb{E}^s_{t_k}\big[p\big(S^\pi_{t_{k+1}}\big)\big]\alpha_{k+1}(b), \\
v_k^{(\theta_1,\theta_2),R,i}(b,s) &= \Delta t\theta_1 g\big(t_k, v_k^{\pi,R,i-1}(s), z_k^{\pi,R}(s)\big) + h_k(b,s), \\
h_k(b,x) &= \mathbb{E}^s_{t_k}\big[p\big(S^\pi_{t_{k+1}}\big)\big]\big(\alpha_{k+1}(b) + \Delta t(1-\theta_1)\gamma_{k+1}(b)\big), \quad i = 1,\dots,I.
\end{aligned} \tag{3}$$

As stated before, a Picard iteration is performed at each time step within each bundle if the choice of (θ_1,θ_2) results in an implicit scheme. For further details on the application of the Picard iteration, readers may refer to [12] or [7] and the references therein.

Finally, the full approximations for each time step are defined as:

$$\begin{aligned}
v_k^{(\theta_1,\theta_2),R,I}(s) &:= \sum_{b=1}^{B} \mathbf{1}_{\mathcal{B}_{t_k}(b)}(s) v_k^{(\theta_1,\theta_2),R,I}(b,s), \\
z_{k,q}^{(\theta_1,\theta_2),R}(s) &:= \sum_{b=1}^{B} \mathbf{1}_{\mathcal{B}_{t_k}(b)}(s) z_{k,q}^{(\theta_1,\theta_2),R}(b,s).
\end{aligned}$$

2.3 Parallel SGBM

One way to improve the efficiency of the SGBM algorithm is by means of GPU acceleration, which has been successfully implemented for the SGBM algorithm for early-exercise options in [10]. The framework of parallel computation from [10] can also be used for the SGBM solver of BSDEs. In this framework, we divide the SGBM algorithm in two stages, namely, the forward simulation stage and the backward recursion approximation stage. In this subsection, we briefly describe how parallel GPU computing is used in each stage.

In the forward simulation stage, we simulate independent samples of the stock price models from the initial time t_0 to the expiration date T as defined by the problem. Moreover, we can already pre-compute all values of interest for the backward step that are related to the stock prices and store them in the memory. These values of interest include the sorting parameter, the basis function values, the expectations of the basis functions and the terminal values. Since the generation of each sample is independent of the other samples and a huge amount of samples may be needed for an accurate Monte Carlo simulation, this stage is particularly suitable for parallel computing. Also note that after the calculation of the values of interest, the actual stock paths can be discarded in the GPU, as they are not required anymore in the backward step.

The second stage is the backward approximation stage. From the discretization of the BSDE, we notice that the calculation in time-wise direction should process sequentially, i.e., starting from the terminal time and going backward along the time direction. However, within each time step, there are many independent processes that are well suitable for parallel computing. Within each time step, the data (i.e. the values of interest) is separated

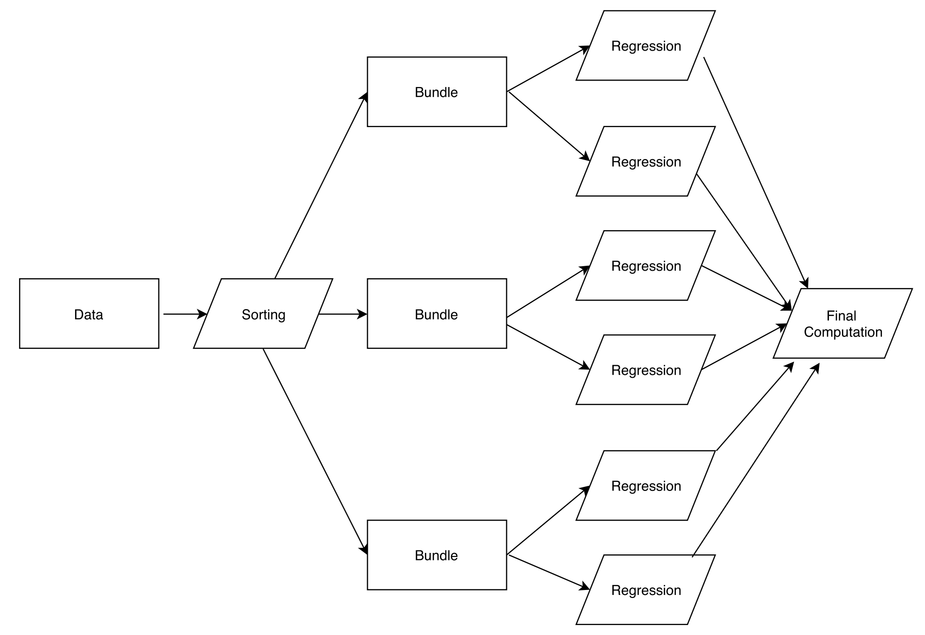

Figure 1 Flowchart for the backward calculation within a time step

into different non-overlapping bundles and the computations in the different bundles are independent of each other. Within a bundle, multiple regression steps on different variables $(V, Z, g, \ldots)$ need to be completed. For a graphical representation of the computation within each time step, we refer to Fig. 1. As each regression within the time step is independent, we can also perform these computation simultaneously. Finally, in order to reduce the overall volume of memory transfers, only the information for the current time point is transferred to the GPU.

3 The SGBM-XVA demonstrator

In order to test and analyze the applicability and practicality of the SGBM algorithm in the BSDE-based financial model framework, we have created an SGBM-XVA demonstrator which computes XVA, i.e. a value of great interest in modern risk management, with the algorithm introduced in the previous section. We make use of the Python programming language and the CUDA Python package for the development of this SGBM-XVA demonstrator. In this section, we address the programming tools that we have used, the basic financial setting for this problem and the design of our code.

3.1 Programming language

Python is the programming language of choice for this project as it is a popular tool in the financial industry nowadays. Being a high-level programming language, Python is easy to develop and particularly useful for scripting because of its easy to write syntax and grammar. These properties are especially useful for the financial industry, as practitioners need to constantly monitor and adapt their models to the changing markets. Moreover, Python has been one of the dominating programming languages in data science with its widely available packages. Therefore, we develop our algorithm under the Python framework. However, Python is an interpreted programming language, so, its performance is not as strong as a compiled language. One of our focusses of study is therefore the balance between rapid development and actual computation performance which can be achieved by Python.

One of the effective techniques for improving the computational efficiency of Python is to pre-compile parts of the code, for example, with the help of the Numba package. This technique has been adopted in our SGBM-XVA demonstrator. In order to further improve the efficiency of our algorithm, we apply parallel GPU computing as stated in the last section. The use of GPUs has been a major development in scientific computing. Along with the computing platform CUDA, the GPU provides a high potential for computational speed-up. With more than hundred threads (the basic operational units within CUDA) in a typical GPU, repetitive function computations can be dedicated to various treads to be run in parallel. It will greatly reduce the computational time.

In this work, we use the CUDA Python packages to incorporate GPU programming into Python. CUDA Python is made out of Python packages from Continuum Analytics, that allow a user to make use of CUDA within the native Python syntax. The tool consists of two main parts: a Numba compiler and the pyculib library. The Numba compiler transforms a native Python code with only supported features into a CUDA kernel or a device function with only a function decorator. This feature enables GPU computing in Python without the need for the programmer to learn a new language. The pyculib library [13] is a collection of several numerical libraries, that provide functions for random number generation, sorting, linear algebra operation and more. This set of tools has been previously included in commercial software, but it has become open source since 2017.

Another benefit of using CUDA Python is in terms of an automatic memory transfer. In a GPU computing framework, it is often necessary to transfer data between the CPU and GPU memory space. In this tool, a programmer can either manage the transfer of memory for better control of GPU memory usage and the bandwidth of memory transfer between CPU and GPU, or let the platform handle it automatically, again simplifying the code development.

The main down-side of this tool is that so far it only supports certain functions from the native Python and Numpy packages. Some of the well-optimized packages in linear algebra are not available for the GPU, and some of the Python code has to be rewritten into a supported version. However, using the package still requires less adaptation effort as compared to the incorporation of other compiled programming languages, like C, into a Python code. As we will discuss later, this tool delivers a great improvement in efficiency in some areas.

3.2 Financial test case: total valuation adjustment (XVA)

Next, we shall introduce the test problem for our SGBM-XVA demonstrator. The main goal here is to show that BSDEs and SGBM can be used to solve financial total valuation adjustment XVA problems in risk management.

In short, we consider a financial market where the bank is selling derivatives to a counterparty, but either the bank or the counterparty may default before expiry. Therefore, a variation margin has to be posted as collateral which will be used to settle the account when one party defaults. In this market, the funding rate for each financial product may be different, as well as the deposit rate for a risk-free account and the funding rate through bonds. The goal of this model is to compare the price of a financial portfolio with and without counterparty risk. The difference between these two prices is called the total value adjustment.

We use a simplified version of the dynamics for our financial market, as in [14]. Within this setting, we take into account the possibility that both parties, in an over-the-counter financial contract, may default, also we include the exchange of collateral, a close-out exchange in the case of a default and the usage of a repurchasing agreement (repo) for position funding. While this model is definitely more realistic than the classical financial derivative pricing theory (where one can borrow and lend freely with negligible cost) by taking into account the counterparty risk, this model still leaves out some parts of the financial deals, like regulatory capital or haircuts that apply to collateral. As already mentioned, we use the standard multi-dimensional Black–Scholes model for the asset dynamics. We select this model as a balance between a relatively realistic model and the tractability for the equations involved. The specific SGBM algorithm can be easily generalized to other models, as described in the outlook section.

Next, we introduce the mathematical model for our asset prices and the default events, as well as the notations that we use. Subsequently, we present the fundamental equations that we are solving in our demonstrator. Detailed financial interpretation as well as the model derivation will be left out and readers are referred to [14] for the description of this XVA model.

3.2.1 The XVA model

In this model, we take on a bank's perspective onto risk management. This perspective focuses on a non-centrally cleared financial derivative on underlying assets S, which are traded between the bank and its client and both parties may default. However, the default will not affect the underlying assets S.

Before we proceed to the BSDE description, which we are going to compute in the SGBM-XVA demonstrator, we introduce the meaning and financial background of the different terms involved. S_i denotes the underlying i-th asset, which follows the standard Black–Scholes model under our assumptions. Its dynamics are defined by the SDE:

$$dS_{t,i} = \bar{\mu}_i S_{t,i}\,dt + \bar{\sigma}_i S_{t,i}\,dB_{t,i}, \quad 1 \leq i \leq d,$$

where B_t is a correlated d-dimensional Wiener process, where

$$dB_{t,i}\,dB_{t,j} = \rho_{ij}\,dt.$$

The constant vector $\bar{\mu} = (\bar{\mu}_1, \ldots, \bar{\mu}_d)^\top$ and $(\bar{\sigma}_1, \ldots, \bar{\sigma}_d)^\top$ represent respectively the drift rate and the standard deviation of the financial assets. The parameters ρ_{ij} form a symmetric non-negative correlation matrix ρ,

$$\rho = \begin{pmatrix} 1 & \rho_{12} & \rho_{13} & \cdots & \rho_{1d} \\ \rho_{21} & 1 & \rho_{23} & \cdots & \rho_{2d} \\ \vdots & \vdots & \vdots & & \vdots \\ \rho_{d1} & \rho_{d2} & \rho_{d3} & \cdots & 1 \end{pmatrix},$$

which is invertible under our assumptions. We can relate the correlated Brownian motion B to a standard, independent d-dimensional Brownian motion W by performing a Cholesky decomposition on ρ. They satisfy the equality

$$B_t = \mathfrak{C} W_t,$$

where $\mathfrak{C}$ is a lower triangular matrix with real and positive diagonal entries, and $\mathfrak{C}\mathfrak{C}^\top = \rho$.

The processes $J^{\mathcal{B}}$ and $J^{\mathcal{C}}$ are used to model the events of default for each party in the transaction. Mathematically, they are defined as counting processes,

$$J_t^{\mathcal{B}} = \mathbf{1}_{\tau^{\mathcal{B}} \le t},$$

and

$$J_t^{\mathcal{C}} = \mathbf{1}_{\tau^{\mathcal{C}} \le t},$$

where $\tau^{\mathcal{B}}$ and $\tau^{\mathcal{C}}$ are stopping times, denoting the random default times of the bank and the counterparty, respectively. The processes are assumed to have stochastic, time-dependent intensities, $\lambda^{\mathcal{B}}$, $\lambda^{\mathcal{C}}$, i.e.

$$\lambda_t^{\mathcal{B}}\, dt = \mathbb{E}\bigl[dJ_t^{\mathcal{B}} | \mathcal{G}_{t-}\bigr]$$

and

$$\lambda_t^{\mathcal{C}}\, dt = \mathbb{E}\bigl[dJ_t^{\mathcal{C}} | \mathcal{G}_{t-}\bigr].$$

Next, we discuss the financial derivatives traded within this model, where we use $\mathfrak{N}$ to denote the terminal payoff for the portfolio. To mitigate counterparty risk, the *variation margins*, X, need to be computed for the two parties. As in [14], the values X are based on the market value of the financial product, and they are computed and re-adjusted frequently. When $X > 0$, the counterparty is posting collateral with the bank.

When one party in the financial contract defaults, the contract position needs to be closed. We denote the portfolio value at default (at time $\tau = \tau^{\mathcal{B}} \wedge \tau^{\mathcal{C}}$) by θ_τ, and it is given by

$$\begin{aligned}\theta_\tau &:= \mathbf{1}_{\tau^{\mathcal{C}} < \tau^{\mathcal{B}}} \theta_\tau^{\mathcal{C}} + \mathbf{1}_{\tau^{\mathcal{B}} < \tau^{\mathcal{C}}} \theta_\tau^{\mathcal{B}} \\ &:= \mathbf{1}_{\tau^{\mathcal{C}} < \tau^{\mathcal{B}}} \bigl(X_\tau + R^{\mathcal{C}} (M_\tau - X_\tau)^+ + (M_\tau - X_\tau)^-\bigr) \\ &\quad + \mathbf{1}_{\tau^{\mathcal{B}} < \tau^{\mathcal{C}}} \bigl(X_\tau + (M_\tau - X_\tau)^+ + R^{\mathcal{B}} (M_\tau - X_\tau)^-\bigr),\end{aligned}$$

where $R^{\mathcal{C}}$, $R^{\mathcal{B}} \in [0, 1]$ are the recovery rates in the case the counterparty and the bank default, respectively. The variable M denotes the close-out value of the portfolio when any party defaults. We will give more details regarding M in a later section.

Next, we introduce the notation for the financial quantities in our model. The adapted stochastic vector processes q^S and γ^S are respectively the repo rate and the dividend yield of the underlying assets. The process r is the stochastic risk-less interest rate. The processes $r^{\mathcal{B}}$ and $r^{\mathcal{C}}$ are the yields of the risky zero coupon bonds of the bank and the counterparty, respectively. The process $q^{\mathcal{C}}$ is the repo rate for the bonds of the counterparty. The interest rate for the variation margin is given by r^X, and, finally, r^F is the cost of external funding. In order to simplify the expression of the demonstrator, we assume all the above processes to be constant.

3.2.2 Fundamental BSDE and reduced BSDE

In [14], a fundamental BSDE is derived through a hedging argument based on a replicating portfolio for the financial derivative. For the hedging portfolio $\hat{V}$, including the counterparty credit risk, the dynamics of posting collateral and holding counterparty bonds for hedging, are given by,

$$\begin{aligned} -d\hat{V}_t &= g\big(t, S_t, \hat{V}_t, \hat{Z}_t, U_t^{\mathcal{B}}, U_t^{\mathcal{C}}\big)\,dt - \hat{Z}_t\,dW_t - U_{t-}^{\mathcal{B}}\,dJ_t^{\mathcal{B}} - U_t^{\mathcal{C}}\,dJ_t^{\mathcal{C}}, \quad t \in [0, \tau \wedge T], \\ \hat{V}_{\tau\wedge T} &= \mathbf{1}_{\tau > T}\mathfrak{N}(S_T) + \mathbf{1}_{\tau \le T}\theta_\tau, \end{aligned} \tag{4}$$

and the driver function is defined as

$$\begin{aligned} g\big(t, s, \hat{v}, \hat{z}, u^{\mathcal{B}}, u^{\mathcal{C}}\big) = {} & -\hat{z}\big(\operatorname{diag}(s)\operatorname{diag}(\bar{\sigma})\mathfrak{C}\big)^{-1}\big(\operatorname{diag}(s)\bar{\mu} \\ & + \operatorname{diag}(s)\big(\gamma^S - q^S\big)\big) + \big(r^{\mathcal{B}} - r\big)u^{\mathcal{B}} + \big(r^{\mathcal{C}} - q^{\mathcal{C}}\big)u^{\mathcal{C}} \\ & + \big(r^X + r\big)X_t - r\hat{v} - \big(r^F - r\big)\big(\hat{v} - X_t + u^{\mathcal{B}}\big)^-. \end{aligned}$$

The notation $\operatorname{diag}(\mathfrak{m})$ denotes a diagonal matrix with the terms of vector $\mathfrak{m}$ on the main diagonal. We compute the price of the derivative by approximating the values of the stochastic processes $(\hat{V}, \hat{Z}, U^{\mathcal{B}}, U^{\mathcal{C}})$ that solve the above BSDE. The price of the contract at time 0 is given by $\hat{V}_0$.

The main difficulty of dealing with this BSDE is that the terminal time is random, as it depends on the time of default. Therefore, this BSDE has to be transformed into a standard BSDE, to reduce the computational complexity, following the techniques used in [15]. The authors in [15] showed that we may recover the solution of (4) by solving the BSDE,

$$\begin{aligned} -d\hat{\mathcal{V}}_t &= g\big(t, S_t, \hat{\mathcal{V}}_t, \hat{\mathcal{Z}}_t, \theta_t^{\mathcal{B}} - \hat{\mathcal{V}}_t, \theta_t^{\mathcal{C}} - \hat{\mathcal{V}}_t\big)\,dt - \hat{\mathcal{Z}}_t\,dW_t, \\ \hat{\mathcal{V}}_T &= \mathfrak{N}(S_T), \quad t \in [0, T], \end{aligned} \tag{5}$$

and using

$$\begin{aligned} \hat{V}_t &= \hat{\mathcal{V}}_t\mathbf{1}_{t<\tau} + \theta_\tau\mathbf{1}_{t\ge\tau}, \\ \hat{Z}_t &= \hat{\mathcal{Z}}_t\mathbf{1}_{t\le\tau}, \\ U_t^{\mathcal{B}} &= \big(\theta_t^{\mathcal{B}} - \hat{\mathcal{V}}_t\big)\mathbf{1}_{t\le\tau}, \\ U_t^{\mathcal{C}} &= \big(\theta_t^{\mathcal{C}} - \hat{\mathcal{V}}_t\big)\mathbf{1}_{t\le\tau}. \end{aligned} \tag{6}$$

Proposition 1 (Theorem A.1 in [14]) *If the pair of adapted stochastic processes $(\hat{\mathcal{V}}, \hat{\mathcal{Z}})$ solves Equation* (5)*, then the solution $(\hat{V}, \hat{Z}, U^{\mathcal{B}}, U^{\mathcal{C}})$ of Equation* (4) *is given by* (6).

Idea of the proof This technique considers the three possibilities of termination, i.e. no default, the bank's default and the counterparty default, and shows that the results in (6) solve (4) under all three situations. □

When we consider the total value adjustment to the financial option values, we compute an alternative price for the same financial contracts, however under the assumption of risk-free dynamics (meaning no counterparty default). The value adjustment is defined as the difference between the two portfolios values (risky minus risk-free option values). The default-free portfolio dynamics, in a repo funding setting, can be expressed as

$$\begin{aligned} -dV_t &= \bigl(-Z_t\bigl(\operatorname{diag}(S_t)\operatorname{diag}(\bar\sigma)\mathfrak{C}\bigr)^{-1}\bigl(\operatorname{diag}(S_t)\bar\mu + \operatorname{diag}(S_t)\bigl(\gamma_t^S - q_t^S\bigr)\bigr) - r_tV_t\bigr)\,dt \\ &\quad - Z_t\,dW_t, \\ V_T &= \mathfrak{N}(S_T). \end{aligned}$$

3.2.3 Discrete system

Here, we explain the setting for the variation margins X and close-out value M. The actual discrete system used in our demonstrator will be introduced as well.

In [16], the variation margin has been mandated to be "the full colleteralised mark-to-market exposure of non-centrally cleared derivatives". Therefore, at any time t, either the mark-to-market risk-free portfolio value V or the counterparty risk adjusted value $\hat V$ appear reasonable choices for the variation margin X. There are also two possible conventions for the portfolio close-out value, namely $M = V$ and $M = \hat V$. In this demonstration, the variation margin and the close-out value are assumed to be the mark-to-market price V. In this case, the valuation problem results in a linear BSDE that contains an exogenous process V. This means that both the variation margin and the close-out value are marked to the market, and are thus not dictated by the bank's internal model. Therefore, there is an additional stochastic process V in the driver for $\hat V$. If the expression of V is not readily available, the numerical simulation of the $\hat V$ system poses extra challenges, because we have to simulate V and $\hat V$ simultaneously.

The driver for the reduced BSDE *with counterparty credit risk* is now given by

$$\begin{aligned} \bar g(t,s,\hat v,\hat z,v) &:= g(t,s,\hat v,\hat z,v-\hat v,v-\hat v) \\ &= -\hat z\bigl(\operatorname{diag}(s)\operatorname{diag}(\bar\sigma)\mathfrak{C}\bigr)^{-1}\bigl(\operatorname{diag}(s)\bar\mu + \operatorname{diag}(s)\bigl(\gamma^S - q^S\bigr)\bigr) \\ &\quad + \bigl(r^{\mathcal B} + r^{\mathcal C} - q^{\mathcal C} + r^X\bigr)v - \bigl(r^{\mathcal B} + r^{\mathcal C} - q^{\mathcal C}\bigr)\hat v, \end{aligned}$$

while the *counterparty risk-free* price V can be expressed as

$$V_t(x) = \mathbb{E}_t^x\bigl[e^{-r(T-t)}\varGamma_{t,T}\mathfrak{N}(S_T)\bigr],$$

where

$$\varGamma_{t,T} = \exp\biggl(-\int_t^T \frac{1}{2}\varphi_u^T\varphi_u\,du - \int_t^T \varphi_u^T\,dW_u\biggr),$$

and

$$\varphi_t := \mathfrak{C}^{-1}\operatorname{diag}(\bar\sigma)^{-1}\bigl(\bar\mu + \gamma_t^S - q_t^S\bigr).$$

Indeed, the elementary expression of V may be available in closed-form for some basic financial derivatives under the simple 1D Black–Scholes model. However, in order for our demonstrator to cover a wide range of products, we solve V by SGBM instead.

Recall the discretization scheme in Equations (2a)–(2c), we can approximate the reduced BSDE with the following numerical scheme:

$$
\begin{aligned}
&\hat{\mathcal{V}}^{\pi}_{t_N} = \mathfrak{N}\big(S^{\pi}_{t_N}\big), \qquad \hat{\mathcal{Z}}^{\pi}_{t_N} = \nabla\mathfrak{N}\big(S^{\pi}_{t_N}\big)\sigma\big(t_N, S^{\pi}_{t_N}\big),\\
&\hat{\mathcal{Z}}^{\pi}_{t_k} = -\theta_2^{-1}(1-\theta_2)\mathbb{E}_{t_k}\big[\hat{\mathcal{Z}}^{\pi}_{t_{k+1}}\big] + \frac{1}{\Delta t}\theta_2^{-1}\mathbb{E}_{t_k}\big[\hat{\mathcal{V}}^{\pi}_{t_{k+1}}\Delta W_k^{\top}\big]\\
&\qquad + \theta_2^{-1}(1-\theta_2)\mathbb{E}_{t_k}\big[\bar{g}\big(t_{k+1}, S^{\pi}_{t_{k+1}}, \hat{\mathcal{V}}^{\pi}_{t_{k+1}}, \hat{\mathcal{Z}}^{\pi}_{t_{k+1}}, V_{t_{k+1}}\big)\Delta W_k^{\top}\big],\\
&\qquad k = N-1, \dots, 0,\\
&\hat{\mathcal{V}}^{\pi}_{t_k} = \mathbb{E}_{t_k}\big[\hat{\mathcal{V}}^{\pi}_{t_{k+1}}\big] + \Delta t\theta_1\bar{g}\big(t_k, S^{\pi}_{t_k}, \hat{\mathcal{V}}^{\pi}_{t_k}, \hat{\mathcal{Z}}^{\pi}_{t_k}, V_{t_k}\big)\\
&\qquad + \Delta t(1-\theta_1)\mathbb{E}_{t_k}\big[\bar{g}\big(t_{k+1}, S^{\pi}_{t_{k+1}}, \hat{\mathcal{V}}^{\pi}_{t_{k+1}}, \hat{\mathcal{Z}}^{\pi}_{t_{k+1}}, V_{t_{k+1}}\big)\big], \quad k = N-1, \dots, 0,
\end{aligned}
$$

where $0 \le \theta_1 \le 1$ and $0 < \theta_2 \le 1$.

Furthermore, if we include the simulation of V and include explicitly the driver functions, we have to solve the following system of discretized BSDEs:

$$
\begin{aligned}
&v^{\pi}_N(s) = \hat{v}^{\pi}_N(s) = \mathfrak{N}(s), \qquad z^{\pi}_N(s) = \hat{z}^{\pi}_N(s) = \nabla\mathfrak{N}(s)\sigma(t_N, s),\\
&z^{\pi}_{k,q}(s) = -\frac{1-\theta_2}{\theta_2}\mathbb{E}^s_{t_k}\big[z^{\pi}_{k+1,q}\big(S^{\pi}_{t_{k+1}}\big)\big]\\
&\qquad + \frac{1}{\theta_2}\big[1-(1-\theta_2)r\Delta t\big]\mathbb{E}^s_{t_k}\left[v^{\pi}_{k+1}\big(S^{\pi}_{t_{k+1}}\big)\frac{\Delta W_{k+1,q}}{\Delta t}\right]\\
&\qquad - \frac{1-\theta_2}{\theta_2}\Delta t\left(\frac{\bar{\mu}+\gamma^S-q^S}{\bar{\sigma}}\right)^{\top}\big(\mathfrak{C}^{-1}\big)^{\top}\mathbb{E}^s_{t_k}\left[z^{\pi}_{k+1}\big(S^{\pi}_{t_{k+1}}\big)^{\top}\frac{\Delta W_{k,q}}{\Delta_k}\right],\\
&\qquad k = N-1, \dots, 0, q = 1, \dots d,\\
&v^{\pi}_k(s) = \frac{1-(1-\theta_1)r\Delta t}{1+\theta_1 r\Delta t}\mathbb{E}^s_{t_k}\big[v^{\pi}_{k+1}\big(S^{\pi}_{t_{k+1}}\big)\big]\\
&\qquad - \frac{\theta_1\Delta t}{1+\theta_1 r\Delta t}\left(\frac{\bar{\mu}+\gamma^S-q^S}{\bar{\sigma}}\right)^{\top}\big(\mathfrak{C}^{-1}\big)^{\top}\big(z^{\pi}_k(s)\big)^{\top}\\
&\qquad - \frac{(1-\theta_1)\Delta t}{1+\theta_1 r\Delta t}\left(\frac{\bar{\mu}+\gamma^S-q^S}{\bar{\sigma}}\right)^{\top}\big(\mathfrak{C}^{-1}\big)^{\top}\mathbb{E}^s_{t_k}\big[\big(z^{\pi}_{k+1}\big(S^{\pi}_{t_{k+1}}\big)\big)^{\top}\big],\\
&\qquad k = N-1, \dots, 0,\\
&\hat{z}^{\pi}_{k,q}(s) = -\frac{1-\theta_2}{\theta_2}\mathbb{E}^s_{t_k}\big[\hat{z}^{\pi}_{k+1,q}\big(S^{\pi}_{t_{k+1}}\big)\big] \qquad (7)\\
&\qquad + \frac{1}{\theta_2}\big[1-(1-\theta_2)\Delta t\big(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}}\big)\big]\mathbb{E}^s_{t_k}\left[\hat{v}^{\pi}_{k+1}\big(S^{\pi}_{t_{k+1}}\big)\frac{\Delta W_{k,q}}{\Delta t}\right]\\
&\qquad - \frac{1-\theta_2}{\theta_2}\Delta t\left(\frac{\bar{\mu}+\gamma^S-q^S}{\bar{\sigma}}\right)^{\top}\big(\mathfrak{C}^{-1}\big)^{\top}\mathbb{E}^s_{t_k}\left[\big(\hat{z}^{\pi}_{k+1}\big(S^{\pi}_{t_{k+1}}\big)\big)^{\top}\frac{\Delta W_{k,q}}{\Delta t}\right]\\
&\qquad + \frac{1-\theta_2}{\theta_2}\Delta t\big(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}}+r^X\big)\mathbb{E}^s_{t_k}\left[v^{\pi}_{k+1}\big(S^{\pi}_{t_{k+1}}\big)\frac{\Delta W_{k,q}}{\Delta t}\right],
\end{aligned}
$$

$$
\begin{aligned}
&k = N-1,\dots,0, \quad q = 1,\dots,d,\\
\hat{v}_k^\pi(s) = {} & \frac{1-(1-\theta_1)\Delta t(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}})}{1+\theta_1\Delta t(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}})}\mathbb{E}^s_{t_k}\big[\hat{v}^\pi_{k+1}\big(S^\pi_{t_{k+1}}\big)\big]\\
&-\frac{\theta_1\Delta t}{1+\theta_1\Delta t(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}})}\left(\frac{\bar{\mu}+\gamma^S-q^S}{\bar{\sigma}}\right)^\top\big(\mathfrak{C}^{-1}\big)^\top\big(\hat{z}_k^\pi(s)\big)^\top\\
&-\frac{(1-\theta_1)\Delta t}{1+\theta_1\Delta t(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}})}\left(\frac{\bar{\mu}+\gamma^S-q^S}{\bar{\sigma}}\right)^\top\big(\mathfrak{C}^{-1}\big)^\top\mathbb{E}^s_{t_k}\big[\big(\hat{z}^\pi_{k+1}\big(S^\pi_{t_{k+1}}\big)\big)^\top\big]\\
&+\frac{\theta_1\Delta t(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}}+r^X)}{1+\theta_1\Delta t(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}})}v_k(s)\\
&+\frac{(1-\theta_1)\Delta t(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}}+r^X)}{1+\theta_1\Delta t(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}})}\mathbb{E}^s_{t_k}\big[v^\pi_{k+1}\big(S^\pi_{t_{k+1}}\big)\big], \quad k = N-1,\dots,0.
\end{aligned}
$$

This system is the one that we have implemented in our demonstrator. Note that the notation $(\frac{\bar{\mu}+\gamma^S-q^S}{\bar{\sigma}})$ is a shorthand for the vector $(\frac{\bar{\mu}_i+\gamma_i^S+q_i^S}{\bar{\sigma}_i})_{1\le i\le d}$.

To close-out this section, we would like to mention that there is an analytic expression for the reference price under the current setting, which is given by

$$
\begin{aligned}
\hat{\mathcal{V}}_t &= \mathbb{E}_t\left[e^{-\int_t^T(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}})\,du}\Gamma_{t,T}\mathfrak{N}(S_T) + \int_t^T e^{-\int_t^s(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}})\,du}\Gamma_{t,s}\big(r^X+r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}}\big)V_s\,ds\right]\\
&= e^{-(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}}-r)(T-t)}V_t + \big(r^X+r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}}\big)\mathbb{E}_t\left[\int_t^T e^{-(r^{\mathcal{B}}+r^{\mathcal{C}}-q^{\mathcal{C}})(s-t)}\Gamma_{t,s}V_s\,ds\right].
\end{aligned}
$$

There is however no elementary expression or simple way to evaluate this quantity.

3.3 Function descriptions

There are two parallel versions of our algorithm, both are written in Python. One of them is based on generic Python code and the Numpy, Scipy and Numba packages for performance and efficiency, which we would refer to as Version 1 from now on for simplicity. The second one makes use of the CUDA toolkit, the CUDA Python portion of the Numba package and the pyculib library and shall be referred to as Version 2. A `numerical_test` script has been provided for running basic comparison tests between the two versions under a predefined test setting.

Next, we list the main functions within our implementation. Since the two versions have similar architecture, we would only present Version 2. We have:

- the main function for the whole algorithm: `cuda_sgbm_xva`;
- the forward simulation function for the basis values and partition ordering: `cuda_jit_montecarlo`
- the main function for the backward approximation stage: `cuda_sgbminnerblock`
- the parallel regression function: `cuda_regression`
- an example class to store all the information related to the test case, including Equation (7): `ArithmeticBasketPut`

All of the above functions form the complete demonstrator but each individual component can be used or replaced separately, which gives us flexibility for future development.

Version 1 only depends on Python and the Numba library. In addition to that, Version 2 also requires the CUDA driver and the pyculib library.

Table 1 Model parameters

S_0	q^S	γ^S	$\bar{\mu}$	$\bar{\sigma}$	ρ
$\begin{pmatrix} 40 \\ \vdots \\ 40 \end{pmatrix}$	$\begin{pmatrix} 0.06 \\ \vdots \\ 0.06 \end{pmatrix}$	$\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0.06 \\ \vdots \\ 0.06 \end{pmatrix}$	$\begin{pmatrix} 0.2 \\ \vdots \\ 0.2 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.25 & & \dots & 0.25 \\ 0.25 & 1 & 0.25 & \dots & 0.25 \\ \vdots & & \ddots & & \vdots \\ 0.25 & & \dots & & 1 \end{pmatrix}$

r	$r^{\mathcal{B}}$	$r^{\mathcal{C}}$	$q^{\mathcal{C}}$	r^X
0.06	0	0	0	0.1

K	T	(θ_1, θ_2)
40	1	(0, 1)

4 Numerical experiments

In this section, we present the tests we have implemented in our demonstrator. In these tests we assume that the bank sells a portfolio consisting of an arithmetic put option, whose payoff is given by

$$g(s) = -\left(K - \frac{1}{d}\sum_{i=1}^{d} s_i\right)^+,$$

where K is the strike price. The detailed setting for the numerical test is presented in Table 1. This set of parameters is based on the one used in [10] and it has been adopted here as an easy to scale up problem. The main purpose of the tests is to investigate whether the code can be easily extended to solving high-dimensional problems, so we choose a set of parameters whose properties do not change when we change the dimensionality of the problem. For a more practical problem, a test based on the German stock index model from [17] has been conducted in [1].

Note that here we have adopted a test case with the borrowing rate and lending rate being equal for the purpose of comparing our default-free prices to previously known result. Additional tests have been performed to assure that the convergence behavior will not be affected by different sets of parameters.

The numerical test has been conducted on a common desktop computer with an Intel(R) Core(TM) i5-7400 processor and a GeForce GTX 1080 graphic card.

4.1 Performance of GPU computing

The first set of tests is performed to check whether there is any benefit of GPU computing for our demonstrator. We perform the same test 10 times separately with the two versions and take the averaged computing time over these 10 runs to compare the efficiency between them. Moreover, in accordance with the work in [1], we use 5 sets of parameters for testing the consistency of this SGBM algorithm. The parameters are presented in Table 2. We have conducted tests with the stock price dimensionality being 1, 5 and 10. The results are shown in the Tables 3, 4 and 5.

As we expected from the theoretical work, the approximate values converge with respect to the progressively increasing grid sizes and the standard deviations decrease as we progress through the parameters sets. More importantly, there is a clear speed-up of Version 2 over Version 1. The biggest improvement comes from the forward simulation stage

Table 2 Testing parameters (paths, steps, bundles)

Parameters	Low dimension (≤ 15)	High dimension (> 15)
1	(2048, 4, 8)	(16384, 4, 8)
2	(4096, 8, 16)	(32768, 8, 16)
3	(8192, 12, 32)	(65536, 12, 32)
4	(16384, 16, 64)	(131072, 16, 64)
5	(32768, 20, 128)	(262144, 20, 128)

Table 3 Test result for dimension 1

Code Version	Parameters	Risk Free Price (Standard Deviation)	Risk Adjusted Price (Standard Deviation)	Time (s)	Speedup
Version 1	Set 1	−2.060 (0.01)	−2.402 (0.01)	0.42	1.24
Version 2		−2.049 (0.007)	−2.389 (0.008)	0.34	
Version 1	Set 2	−2.066 (0.003)	−2.408 (0.003)	0.93	3.88
Version 2		−2.063 (0.002)	−2.405 (0.002)	0.24	
Version 1	Set 3	−2.065 (0.002)	−2.406 (0.002)	2.58	7.17
Version 2		−2.068 (0.003)	−2.410 (0.004)	0.36	
Version 1	Set 4	−2.066 (0.002)	−2.407 (0.002)	6.81	10.32
Version 2		−2.066 (0.001)	−2.407 (0.001)	0.66	
Version 1	Set 5	−2.066 (0.0006)	−2.407 (0.0007)	16.69	13.79
Version 2		−2.066 (0.0006)	−2.407 (0.0007)	1.21	

Table 4 Test result for dimension 5

Code Version	Parameters	Risk Free Price (Standard Deviation)	Risk Adjusted Price (Standard Deviation)	Time (s)	Speedup
Version 1	Set 1	−1.007 (0.006)	−1.175 (0.007)	1.89	3.57
Version 2		−1.001 (0.01)	−1.167 (0.01)	0.53	
Version 1	Set 2	−1.011 (0.002)	−1.178 (0.002)	6.99	8.13
Version 2		−1.010 (0.002)	−1.178 (0.002)	0.86	
Version 1	Set 3	−1.011 (0.001)	−1.178 (0.001)	20.59	9.36
Version 2		−1.011 (0.002)	−1.178 (0.002)	2.20	
Version 1	Set 4	−1.012 (0.0008)	−1.179 (0.0010)	54.24	10.16
Version 2		−1.012 (0.0009)	−1.180 (0.001)	5.34	
Version 1	Set 5	−1.013 (0.0005)	−1.180 (0.0005)	134.61	10.96
Version 2		−1.012 (0.0005)	−1.180 (0.0006)	12.28	

Table 5 Test result for dimension 10

Code Version	Parameters	Risk Free Price (Standard Deviation)	Risk Adjusted Price (Standard Deviation)	Time (s)	Speedup
Version 1	Set 1	−0.834 (0.005)	−0.973 (0.006)	6.38	5.55
Version 2		−0.838 (0.004)	−0.978 (0.004)	1.15	
Version 1	Set 2	−0.839 (0.002)	−0.977 (0.002)	24.74	8.36
Version 2		−0.840 (0.002)	−0.979 (0.002)	2.96	
Version 1	Set 3	−0.841 (0.001)	−0.980 (0.002)	74.12	8.58
Version 2		−0.840 (0.001)	−0.979 (0.001)	8.64	
Version 1	Set 4	−0.841 (0.0008)	−0.980 (0.0009)	198.08	9.18
Version 2		−0.841 (0.0006)	−0.979 (0.0007)	21.58	
Version 1	Set 5	−0.841 (0.0003)	−0.980 (0.0003)	498.71	9.32
Version 2		−0.841 (0.0007)	−0.980 (0.0008)	53.51	

where we run each independent sample in parallel, and greatly reduce the computational time. There still seems to be room for improvement regarding the backward simulation stage as even if we perform the regression in parallel on the GPU in Version 2 instead of sequentially in Version 1, the speed-up is not obvious. The main reason may be that some part of the code is not performed on the GPU to keep some flexibility of our demonstrator. Some optimization is only available for the basic Python code but not for CUDA Python.

Nevertheless, our tests show that CUDA Python improves the efficiency and may provide a good balance between the speed of execution and the speed of development. In particular, as the workload increases due to using more time steps, samples and bundles, the speed-up get higher due to GPU computing.

Finally, whereas the build-in floating point format of the GPU is 32 bits, 64 bits format is needed for ensuring high accuracy with our result.

4.2 Scalability

Next, we would like to check if our algorithm can be used for high-dimensional test cases. The setting is the same as in the last section. However, Version 1 is too time-consuming for a common desktop, so the focus is on Version 2. We perform tests up to 40 dimensions, as shown in Table 6.

It can be seen that with the help of the GPU, our method can be generalized to higher dimension. The application of GPU computing definitely expands the applicability of our method. However, we also notice that the time ratio increase is higher then the dimensional ratio. This is probably related to the architecture and the build-in hardware of the GPU. It maybe worthwhile to tune the GPU setting within our demonstrator corresponding to the GPU at hand, but this is hardware-dependent.

To sum up, the application of GPU computing not only speeds up our algorithm, but it also allows us to deal with more demanding problems with about the same hardware.

Table 6 Test result for high-dimensional cases with Version 2

Dimension	Parameters	Risk Free Price (Standard Deviation)	Risk Adjusted Price (Standard Deviation)	Time (s)
15	Set 1	−0.777 (0.005)	−0.906 (0.006)	2
	Set 2	−0.780 (0.002)	−0.909 (0.002)	8
	Set 3	−0.779 (0.0008)	−0.908 (0.0009)	22
	Set 4	−0.781 (0.0006)	−0.910 (0.0007)	60
	Set 5	−0.781 (0.0004)	−0.909 (0.0004)	144
20	Set 6	−0.746 (0.002)	−0.870 (0.002)	29
	Set 7	−0.748 (0.0007)	−0.872 (0.0008)	116
	Set 8	−0.749 (0.0004)	−0.872 (0.0005)	345
	Set 9	−0.749 (0.0002)	−0.873 (0.0003)	911
	Set 10	−0.749 (0.0001)	−0.873 (0.0001)	2278
30	Set 6	−0.715 (0.002)	−0.834 (0.002)	84
	Set 7	−0.716 (0.0008)	−0.835 (0.0009)	337
	Set 8	−0.717 (0.0005)	−0.836 (0.0006)	997
	Set 9	−0.718 (0.0002)	−0.836 (0.0002)	2675
	Set 10	−0.718 (0.0001)	−0.837 (0.0002)	6715
40	Set 6	−0.698 (0.001)	−0.814 (0.002)	176
	Set 7	−0.701 (0.0007)	−0.817 (0.0009)	701
	Set 8	−0.701 (0.0005)	−0.817 (0.0006)	2086
	Set 9	−0.702 (0.0002)	−0.818 (0.0003)	5562
	Set 10	−0.702 (0.0002)	−0.818 (0.0002)	13,893

5 Conclusion and outlook

5.1 Conclusion

Although we just developed a demonstrator, our study gives us interesting insight in the performance of the SGBM algorithm for BSDEs. We have shown that we can apply the SGBM algorithm to solve high-dimensional BSDEs with the help of GPU computing, which is an important feature for using BSDEs in practice. With a suitable BSDE model, we can deal with complicated financial risk management problems with our solver and since our code is based on a general framework for BSDEs, our demonstrator can be adopted to different pricing and valuation situations. We also demonstrated that we can incorporate GPU computing into a native Python code. We believe that this work serves as a basis for developing BSDE-based software for financial applications.

The authors encourage readers to download and test our demonstrator, which solves the problem which was stated in this work. The aim is to developed this tool further in terms of financial applications and computational efficiency.

Next, we will mention some possible generalizations for our algorithm as a guideline to future use. We will divide the outlook into two parts, a financial and a computational part.

5.2 Financial outlook

In this work, we used the classical Black–Scholes model for the stock dynamics. It is of interest to generalize the stock model by other diffusion SDEs, of the form:

$$\begin{aligned} dS_t &= \mu(t,S_t)\,dt + \sigma(t,S_t)\,dW_t, \\ S_0 &= s_0. \end{aligned}$$

This would already result in a different implementation regarding the SGBM regress-later component.

We may also alter the model dynamics to better fit the market, for example, with the inclusion of initial margins and capital requirement. Recall that in Sect. 3.2.3 we have made some assumptions about the variation margin X and the close-out value M. Simply by adjusting these assumptions, a completely different BSDE driver may result. There are four possible combinations for the variation margin and the close-out value. Each choice gives rise to a different BSDE driver, which can be seen in Table 7.

When X and M are the adjusted values $\hat{V}$, it results in a linear BSDE for $\hat{V}$. This means that the variation margin and the close-out value are marked to the model. As all the values are marked in $\hat{V}$ and both values match, the equation reduces to an option pricing equation, with different interest rates. When both of them are the risk-free interest rates, a linear BSDE results that contains an exogenous process V. This means that both the

Table 7 Different BSDE drivers for various choices of X and M

Case	X	M	$g(t,S_t,\hat{\mathcal{V}}_t,\hat{\mathcal{Z}}_t,\theta_t^{\mathcal{B}}-\hat{\mathcal{V}}_t,\theta_t^{\mathcal{C}}-\hat{\mathcal{V}}_t)$
1	$\hat{V}$	$\hat{V}$	$-\hat{\mathcal{Z}}_t\sigma(t,S_t)^{-1}(\mu(t,S_t)+\mathrm{diag}(\gamma_t^{\mathcal{S}}-q^{\mathcal{S}})S_t)+r_t^X\hat{V}_t$
2	V	V	$-\hat{\mathcal{Z}}_t\sigma(t,S_t)^{-1}(\mu(t,S_t)+\mathrm{diag}(\gamma_t^{\mathcal{S}}-q^{\mathcal{S}})S_t)+(r_t^{\mathcal{B}}+r_t^{\mathcal{C}}-q_t^{\mathcal{C}})(V_t-\hat{\mathcal{V}}_t)+r_t^XV_t$
3	V	$\hat{V}$	$-\hat{\mathcal{Z}}_t\sigma(t,S_t)^{-1}(\mu(t,S_t)+\mathrm{diag}(\gamma_t^{\mathcal{S}}-q^{\mathcal{S}})S_t)+(r_t^{\mathcal{B}}+r_t^{\mathcal{C}}-q_t^{\mathcal{C}})(V_t-\hat{\mathcal{V}}_t)+r_t^XV_t+(r_t^{\mathcal{B}}+R^{\mathcal{C}}r_t^{\mathcal{C}}-r_t-R^{\mathcal{C}}q_t^{\mathcal{C}})(\hat{V}_t-V_t)^++[(r_t^{\mathcal{B}}-r_t^F)R^{\mathcal{B}}+r_t^{\mathcal{C}}-q_t^{\mathcal{C}}](\hat{V}_t-V_t)^-$
4	$\hat{V}$	V	$-\hat{\mathcal{Z}}_t\sigma(t,S_t)^{-1}(\mu(t,S_t)+\mathrm{diag}(\gamma_t^{\mathcal{S}}-q^{\mathcal{S}})S_t)+r_t^XV_t+[(r_t^{\mathcal{B}}-r_t)+R^{\mathcal{C}}(r_t^{\mathcal{C}}-q_t^{\mathcal{C}})](V_t-\hat{V}_t)^++[(r^{\mathcal{B}}-r_t^F)R^{\mathcal{B}}+r_t^{\mathcal{C}}-q_t^{\mathcal{C}}](V_t-\hat{V}_t)^-$

variation margin and the close-out value are marked to the market and are not dictated by the internal model. Then there is an additional stochastic process V in the driver for $\hat{V}$, which is the case we have used in this work.

Finally, when there is a mismatch between M and X, the resulting BSDE is no longer linear. Take case 4 as an example, it implies that the variation margin is collected according to an internal model while the close-up value is given by the market. The mismatch between the variation margin and the close-up value results in non-linear terms in the driver. These non-linear terms come from the cash flow when one party defaults.

We should notice that since we have a general BSDE algorithm in the form of SGBM, we may adapt our code to these new models by simply changing the model setting in the example class without changing the actual function. This is one of the advantages of using BSDEs in pricing and risk management.

5.3 Computational outlook

As we have mentioned before, one possible way of improvement is to sacrifice some flexibility and move the remaining solver parts of the code also to the GPU. Alternatively, one may further optimize the code within the GPU as the GPU set of routines seems to be still developing.

Furthermore, other features may be included in the software, for example, different payoff functions or a different SGBM regression basis. Finally, to have a fully independent software, a user interface and modular code are recommended.

Acknowledgements
The authors would also like to thank all the supportive consultants and programmers from VORtech, BV, for their help and advice and valuable discussion for this work.

Funding
This work is supported by EU Framework Programme for Research and Innovation Horizon 2020 (H2020-MSCA-ITN-2014, Project 643045, 'EID WAKEUPCALL').

Abbreviations
XVA, Total Valuation Adjustment; SGBM, Stochastic Grid Bundling Method; SDE, Stochastic Differential Equation; BSDE, Backward Stochastic Differential Equation; FBSDE, Forward-Backward Stochastic Differential Equation; GPU, Graphics Processing Unit; repo, Repurchasing Agreement.

Availability of data and materials
The platform independent Python 3 code generated during the current study are available at https://github.com/kwchau/sgbm_xva under the MIT license.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
All authors were involved in planning the work, drafting and writing the manuscript. Discussion and comments on the results and the manuscript as well as a critical revision have been made by all the authors at all stages. KWC is the main author for the software mentioned in this manuscript. All authors read and approved the final manuscript.

Author details
[1]Delft Institute of Applied Mathematics, Delft University of Technology, Delft, The Netherlands. [2]VORtech BV, Delft, The Netherlands. [3]*Present address:* Data Science Department, Netherlands Organisation for Applied Scientific Research, The Hague, The Netherlands. [4]Research Group of Scientific Computing, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands.

Received: 8 July 2019 Accepted: 11 February 2020 Published online: 19 February 2020

References

1. Chau KW, Oosterlee CW. Stochastic grid bundling method for backward stochastic differential equations. Int J Comput Math. 2019;96(11):2272–301.
2. Pardoux E, Peng SG. Adapted solution of a backward stochastic differential equation. Syst Control Lett. 1990;14(1):55–61.
3. Pardoux E, Peng SG. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In: Rozovskii BL, Sowers RB, editors. Stochastic partial differential equations and their applications: proceedings of IFIP WG 7/1 international conference University of North Carolina at Charlotte, NC. June 6–8, 1991. Berlin: Springer; 1992. p. 200–17.
4. Lemor J-P, Gobet E, Warin X. Rate of convergence of an empirical regression method for solving generalized backward stochastic differential equations. Bernoulli. 2006;12(5):889–916.
5. Briand P, Labart C. Simulation of BSDEs by Wiener chaos expansion. Ann Appl Probab. 2014;24(3):1129–71.
6. Crisan D, Manolarakis K. Solving backward stochastic differential equations using the cubature method: application to nonlinear pricing. SIAM J Financ Math. 2012;3(1):534–71.
7. Ruijter MJ, Oosterlee CW. A Fourier cosine method for an efficient computation of solutions to BSDEs. SIAM J Sci Comput. 2015;37(2):A859–89.
8. Chau KW, Oosterlee CW. On the wavelet-based SWIFT method for backward stochastic differential equations. IMA J Numer Anal. 2018;38(2):1051–83.
9. Jain S, Oosterlee CW. The stochastic grid bundling method: efficient pricing of Bermudan options and their greeks. Appl Math Comput. 2015;269:412–31.
10. Leitao A, Oosterlee CW. GPU acceleration of the stochastic grid bundling method for early-exercise options. Int J Comput Math. 2015;92(12):2433–54.
11. Bender C, Steiner J. Least-squares Monte Carlo for backward SDEs. In: Carmona AR, Del Moral P, Hu P, Oudjane N, editors. Numerical methods in finance; Bordeaux; June 2010. Berlin: Springer; 2012. p. 257–89.
12. Gobet E, Lemor J, Warin X. A regression-based Monte Carlo method to solve backward stochastic differential equations. Ann Appl Probab. 2005;15(3):2172–202.
13. Pyculib; [cited 26 June 2019]. Available from: http://pyculib.readthedocs.io/en/latest/#.
14. Lesniewski A, Richter A. Managing counterparty credit risk via BSDEs. ArXiv e-prints. 2016.
15. Kharroubi I, Lim T, Ngoupeyou A. Mean-variance hedging on uncertain time horizon in a market with a jump. Appl Math Optim. 2013;68(3):413–44.
16. Basel Committee on Banking Supervision. Margin requirements for non-centrally cleared derivatives; 2015.
17. Reisinger C, Wittum G. Efficient hierarchical approximation of high-dimensional option pricing problems. SIAM J Sci Comput. 2007;29(1):440–58.