Deep BSDE Notes

Kenrick Raymond So

Table of Contents

Chapter 1: Preliminary Concepts

I Links Between SDEs and PDEs

A stochastic differential equation (SDE) together with it's initial condition determines a diffusion process. This can be used to define a deterministic function of space and time in two distinct ways:

- By considering the expected value of some future payoff as a function of the initial position and time
- By considering the probability of being in a certain state at a given time, given knowledge of the initial state and time.

The two viewpoints are the dual problem of one another. Specifically, the evolving probability density solves the forward Kolmogorov equation, which is the adjoint of the backward Kolmogorov equation.

Backward Kolmogorov Equation (Kohn, 2011; Gardiner, 2004): Suppose $y(t)$ solves the scalar SDE

$$dy = f(y,s)ds + g(y,s)dW_s$$

and let

$$u(x,t) = \mathbb{E}_{y(t)=x}[\Phi[y(T)]]$$

be the expected value of some payoff $\Phi(\cdot)$ at maturity T given $y(t) = x$. Then u solves

$$u_t + f(x,t)u_x + \frac{1}{2}[g(x,t)]^2 u_x x = 0.$$

This PDE is the backward Kolmogorov equation. Intuitively, the backward Kolmogorov equation describes how the expected value of a future payoff $\Phi(y(T))$ depends on your current location x and current time t. If you want to compute the expected payoff of a stochastic process that starts at position x at time t, then this expected value $u(x,t)$ evolves backward in time according to the backward Kolmogorov equation. Suppose we are interested in the discounted final payoff with the form

$$u(x,t) = \mathbb{E}_{y(t)=x}\left[e^{-\int_t^T b(y(s),s)ds}\Phi(y(T))\right]$$

for some specified function $b(y)$, typically the interest rate. Then u solves

$$u_t + f(x,t)u_x + \frac{1}{2}[g(t,x)]^2 u_{xx} - b(x,t)u = 0. \tag{1}$$

The backward Kolmogorov equation is a special case of the backward parabolic PDE that the Feynman-Kac formula gives a probabilistic representation for.

Vector-Valued Diffusion: Suppose y solves a vector-valued SDE

$$dy_i = f_i(y,s)ds + \sum_j g_{i,j}(y,s)dW_j$$

where each component of W is an independent Brownian motion. Then,

$$u(x,t) = \mathbb{E}_{y(t)=x}[\Phi(y(T))]$$

solves

$$u_t + \mathcal{L}u = 0.$$

Here, the infinitesimal generator of the diffusion process $y(s)$ is given by

$$\mathcal{L}u(x,t) = \sum_i f_i \frac{\partial u}{\partial x_i} + \frac{1}{2}\sum_{i,j,k} g_{i,k}g_{j,k}\frac{\partial^2 u}{\partial x_i \partial x_j}.$$

A straightforward application of the multidimensional Ito's lemma results in the multidimensional Feynman-Kac given by

$$u_t + \mathcal{L}u - bu = 0.$$

Forward Kolmogorov Equation: The solution of the SDE is a Markov process, so it has a well defined transition probability $p(\cdot, s; x, t)$ is the probability density of the state at time s given that it started at position x at time t. To describe a Markov process, p must satisfy the Chapman–Kolmogorov equation

$$p(z,s;x,t) = \int_{\mathbb{R}^n} p(z_1,s_1;x,t)p(z,s;z_1,s_1)dz_1$$

for any $t < s_1 < s$. This equation expresses the idea that the probability of transitioning from state x at time t to state z at a later time s can be computed by considering all the possible intermediate states z_1 the process could pass through at some intermediate time s_1. The total probability of this happening is obtained by integrating over all such possible intermediate states $z_1 \in \mathbb{R}^n$. Let $\rho_0(x)$ be the probability density of the state at time t. The probability density as a function of z for any time $s > t$ is given by

$$\rho(z,s) = \int_{\mathbb{R}^n} p(z,s;x,t)\rho_0(x)dx. \tag{2}$$

The important fact about transition probaibilities is that it solves the forward Kolmogorov equation in s and z

$$-p_s - \sum_i \frac{\partial}{\partial z_i}[f_i(z,s)p] + \frac{1}{2}\sum_{i,j,k}\frac{\partial^2}{\partial z_i \partial z_j}[g_{i,k}(z,s)g_{j,k}(z,s)p] = 0 \tag{3}$$

with initial condition

$$p = \delta_x(z) \text{ at } s = t.$$

Using the inifinitesimal generator notation, the forward Kolmogorov equation can be written as

$$-p_s + \mathcal{L}^* p = 0$$

where

$$\mathcal{L}^* p = -\sum_i \frac{\partial}{\partial z_i}(f_i p) + \sum_{i,j} \frac{\partial^2}{\partial z_i \partial z_j}(\frac{1}{2}(gg^T)_{i,j} p)$$

Nonlinear Feynman–Kac Formula. The classical Feynman–Kac formula provides a probabilistic representation for the solution of certain linear parabolic partial differential equations. This framework can be extended to the nonlinear setting, where the PDE includes a nonlinear dependence on the solution and its gradient. Consider the nonlinear PDE

$$\frac{\partial}{\partial t}u(t,x) + \mathcal{L}u(t,x) + f(t,x,u(t,x),D_x u(t,x)) = 0, \quad u(T,x) = g(x), \tag{4}$$

where $\mathcal{L}$ is a second-order differential operator defined by

$$\mathcal{L}u(t,x) = \frac{1}{2}\mathrm{Tr}\left[\sigma(x)\sigma(x)^\top D_x^2 u(t,x)\right] + \mu(x)^\top D_x u(t,x),$$

and $D_x u(t,x)$, $D_x^2 u(t,x)$ are the gradient and Hessian of u with respect to the spatial variable x. The function f introduces a nonlinearity in u and its derivatives. Let X_t be the solution to the FSDE

$$dX_t = \mu(X_t)\,dt + \sigma(X_t)\,dW_t, \quad X_0 = x,$$

and define $Y_t := u(t, X_t)$. Then, by applying Itô's formula to $u(t, X_t)$, we obtain the dynamics of Y_t:

$$\begin{aligned} dY_t &= \left(\frac{\partial u}{\partial t}(t,X_t) + \mathcal{L}u(t,X_t)\right)dt + D_x u(t,X_t)\,\sigma(X_t)\,dW_t \\ &= -f(t,X_t,Y_t,Z_t)\,dt + Z_t\,dW_t, \end{aligned}$$

where $Z_t := D_x u(t,X_t)\,\sigma(X_t)$. This BSDE is solved backward in time with terminal condition $Y_T = g(X_T)$.

II Backward SDE

Recall that a BSDE has the following form:

$$\begin{aligned} -dY_t &= f(t,Y_t,Z_t)dt - Z_t dW_t \\ Y_T &= \xi \end{aligned} \tag{5}$$

or equivalently,

$$Y_t = \xi + \int_t^T f(s,Y_s,Z_s)ds - \int_t^T Z_s dW_s. \tag{6}$$

Here, the random variable $\xi : \Omega \to \mathbb{R}^d$ is $\mathcal{F}_t$ measurable.

Definition II.1. *A continuous solution of a BSDE is a pair* $(Y, Z) = (Y_t, Z_t)_{t\in[0,T]}$ *such that* Y *is a continuous adapted* $\mathbb{R}^d$*-valued process and* Z *is a predictable* $\mathbb{R}^{n\times d}$*-valued process with* $\int_0^T |Z_t|^2 ds < \infty, \mathbb{P} - a.s.$ *that satisfies the BSDE. The solution* (Y, Z) *is square integrable if* $(Y, Z) \in \mathcal{H}_T^{2,d} \times \mathcal{H}_T^{2,n\times d}$.

Definition II.2. *The driver* f *is standard if* $f(\cdot, 0, 0) \in \mathcal{H}_T^{2,d}$ *and* $f(\omega, t, y, z)$ *is uniformly Lipschitz in* (y, z)*. This latter property means that* $\exists C > 0$ *such that* $d\mathbb{P} \otimes dt$ *a.s.,*

$$\left|f\left(\cdot, y_1, z_1\right) - f\left(\cdot, y_2, z_2\right)\right| \leq C\left(|y_1 - y_2| + |z_1 - z_2|\right) \quad \forall \left(y_1, z_1\right), \left(y_2, z_2\right) \in \mathbb{R}^d \times \mathbb{R}^{n\times d}$$

If in addition $\xi \in \mathcal{L}_T^{2,d}$*, we say that the data* (f, ξ) *are standard data.*

If (f, ξ) are the standard data, then there exists a unique continuous square-integrable solution to the BSDE (El Karoui et al., 1997b).

III Forward Backward SDE

Typically, a coupled FBSDE is referred to as an FBSDE, while a decoupled FBSDE is referred to as a BSDE. A Forward Backward SDE (FBSDE) has the following form

$$Y_t = \xi + \int_t^T f(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s \tag{7}$$

where the underlying process X_s is a diffusion process satisfying the following forward SDE

$$dX_s = b(X_s)ds + \sigma(X_s)dW_s.$$

For a FBSDE to have a unique solution, the following conditions must be satisfied (El Karoui et al., 1997b):

1. The coefficients b and σ in the F-SDE must be uniformly Lipschitz continuous with respect to x. Let there exist a constant $C \in \mathbb{R}_+$such that for all $t \geq 0$ within the domain and any x, x' in $\mathbb{R}^d$, the following inequalities hold:

$$\left|b(t, x) - b\left(t, x'\right)\right| + \left|\sigma(t, x) - \sigma\left(t, x'\right)\right| \leq C\left(1 + \left|x - x'\right|\right),$$

   the coefficients b and σ are then considered Lipschitz continuous if this is met.

2. The coefficient b and σ must satisfy a linear growth condition. Let there exist another constant D such that for any (t, x) following condition is met:

$$|\sigma(t, x) + b(t, x)| \leq D(1 + |x|).$$

3. The driver function $f\left(t, X_t, Y_t, Z_t\right)$ is uniformly Lipschitz continuous with respect to the pair (Y, Z).

4. $\xi \in \mathbb{L}_T^2\left(\mathbb{R}^d\right)$, and $(f(t, 0, 0, 0))_{t\leq T} \in H_T^2\left(\mathbb{R}^d\right)$.

IV Reflected FBSDE

Reflected Backward Stochastic Differential Equations (RBSDEs) (El Karoui et al., 1997a; Karoui et al., 1997) extend standard BSDEs by adding a reflection term that keeps the solution above a given barrier. This is useful for modeling problems with early exercise features, such as American options. In the case of American options, the barrier is the option's payoff at each time t, denoted by $g(t, X_t)$. The reflection term increases only when the solution Y_t reaches this barrier, ensuring that $Y_t \geq g(t, X_t)$ at all times. For a given terminal condition ξ and driver function f, a RBSDE is given by

$$Y_t = \xi + \int_t^T f(s, Y_s, Z_s)ds - \int_t^T Z_s dWs + K_T - K_t, \quad 0 \leq t \leq T. \tag{8}$$

Here, (Y_t, Z_t) are $\mathcal{F}_t$ adapted processes and K_t is continuous and increasing. The process K_t plays a role in enforcing the constraint $Y_t \geq g(t, X_t)$. It increases only when $Y_t = g(t, X_t)$, meaning the reflection is active only when needed. This ensures that the solution stays above the obstacle without introducing unnecessary adjustments. Mathematically, this condition is enforced by the Skorokhod condition:

$$\int_0^T (Y_t - g(t, X_t))\, dK_t = 0. \tag{9}$$

This implies that K_t increases only on the set where $Y_t = g(t, X_t)$. In other words, K_t only acts to keep Y_t above the barrier and does nothing when the solution is already strictly above it. The process Y_t represents the price of the option, and the constraint $Y_t \geq g(t, X_t)$ ensures that the option is never valued below its payoff. The optimal stopping feature is encoded through the minimality of the reflection term K_t, which activates only when immediate exercise is optimal.

V Option Pricing using Neural Networks

VI Option Hedging using Neural Networks

Chapter 2: Deep BSDE

I Original Framework

This section is based on Güler et al. (2019) and Raissi (2018). This work is also referred to as local DBSDE. Consider the following system of coupled forward-backward stochastic differential equations (FBSDEs),

$$\begin{aligned} dX_t &= \mu(t, X_t, Y_t, Z_t)\, dt + \sigma(t, X_t, Y_t)\, dW_t, \qquad & X_0 &= \xi, \\ dY_t &= \phi(t, X_t, Y_t, Z_t)\, dt + Z_t^\top \sigma(t, X_t, Y_t)\, dW_t, \qquad & Y_T &= g(X_T), \end{aligned} \tag{10}$$

where (X_t, Y_t, Z_t) are adapted stochastic processes. The forward process X_t represents the state dynamics of the system (e.g., the evolution of an asset), while the backward components (Y_t, Z_t) typically represent a value process (such as a derivative price) and an associated control. This coupled FBSDE is closely related to a quasi-linear parabolic partial differential equation (PDE) of the form

$$u_t = f(t, x, u, Du, D^2u),$$

with terminal condition $u(T, x) = g(x)$, where $u(t, x)$ is the unknown solution, interpreted as the value function or option price at time t and state x. The PDE coefficients are related to the FBSDE coefficients via

$$f(t, x, u, Du, D^2u) = \phi(t, x, u, Du) - \mu(t, x, u, Du)^\top Du - \frac{1}{2}\mathrm{Tr}\left[\sigma(t, x, u)\sigma(t, x, u)^\top D^2u\right], \tag{11}$$

where Du and D^2u are the gradient and Hessian of u, respectively. This connection is a generalization of the nonlinear Feynman–Kac formula and provides a probabilistic representation of solutions to certain nonlinear PDEs. It was shown in Gobet (2020) that under sufficient regularity conditions, the solution of the PDE and the solution of the coupled FBSDE are linked via

$$\begin{aligned} Y_t &= u(t, X_t), \qquad && \text{(value process / derivative price)} \\ Z_t &= \nabla u(t, X_t), \qquad && \text{(control process / hedging strategy).} \end{aligned}$$

The FSDE describes the evolution of the underlying state variable, while the BSDE encodes the value of a contingent claim or optimal control objective, which is solved backward from the terminal condition. In financial applications, Y_t corresponds to the price of a derivative written on the underlying state X_t, and Z_t reflects the sensitivity of the price with respect to the driving Brownian motion, which is often interpreted as the hedging strategy. Rather than solving a high-dimensional PDE directly, one can simulate the corresponding FBSDE and recover the solution probabilistically. This motivates the use of deep learning and Monte Carlo methods for solving high-dimensional problems where PDE-based grid methods become infeasible.

If the solution $u(t, x)$ to the nonlinear PDE in Equation 11 were known, we could apply a time-discretization method such as the Euler–Maruyama scheme to solve the corresponding FBSDE in Equation 10. However, since the exact solution is typically unavailable, we instead approximate it by parameterizing the function $u(t, x)$ with a neural network, denoted

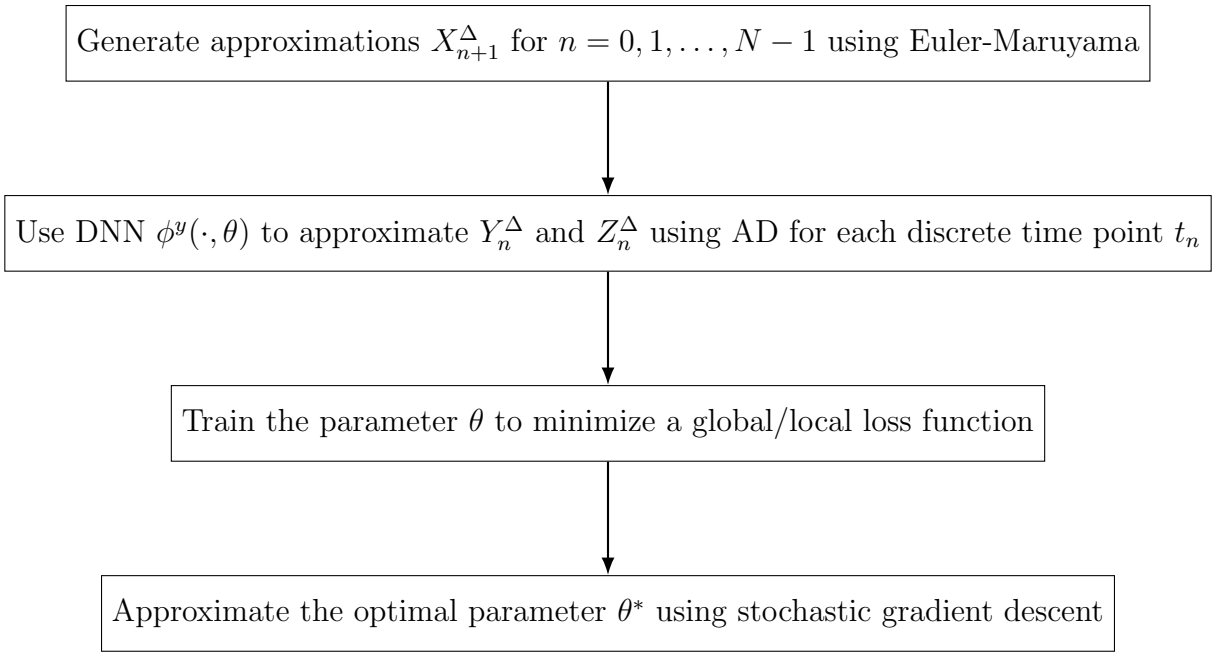

Figure 1: Flowchart for Local DBSDE

$u^{\Theta}(t, x)$, where Θ represents the set of learnable parameters. Here, the gradient $\nabla_x u^{\Theta}(t, x)$ is calculated using automatic differentiation (AD), AD works by decomposing the neural network into a sequence of elementary operations and then efficiently computing derivatives using the chain rule. This allows us to obtain the hedging term Z_t without manual computation of derivatives. The neural network takes the initial input (t_0, X_0) and produces an estimate of $Y_0 = u^{\Theta}(t_0, X_0)$. The associated Z_0 is computed using automatic differentiation. With (Y_0, Z_0) available, we simulate the next forward state (t_1, X_1) using the discretized forward SDE. The dependency of X_1 on (Y_0, Z_0) reflects the nonlinear coupling between the forward and backward components. This recursive procedure is repeated over the discretized time grid until terminal time T.

Training the Neural Network: To train the neural network, we define a loss function that measures how well the neural network approximates the solution of the BSDE. The loss function to be minimized is given by,

$$\min_{\Theta} \sum_{m=1}^{M} \sum_{n=0}^{N-1} | \underbrace{Y_{n+1}^m(\Theta)}_{\text{NN Output}} - \underbrace{Y_n^m(\Theta) - \varphi\left(t_n, X_n^m, Y_n^m(\Theta), Z_n^m(\Theta)\right) \Delta t_n}_{\text{Euler Scheme Previous Time Step}}$$
$$\underbrace{- \left(Z_n^m(\Theta)\right)^T \sigma\left(t_n, X_n^m, Y_n^m(\Theta)\right) \Delta W_n^m}_{\text{Euler Scheme Previous Time Step}} |^2$$
$$+ \sum_{m=1}^{M} | \underbrace{Y_N^m(\Theta) - g\left(X_N^m\right)}_{\text{Terminal Condition Term}} |^2.$$

The idea is to compare the output of the network at the next time step, $Y_{n+1}^m(\Theta)$, with

the value predicted by the Euler scheme using information from the current time step. This includes the current value $Y_n^m(\Theta)$, the driver function φ, and the noise term involving $Z_n^m(\Theta)$ and the Brownian increment ΔW_n^m. If the network is correctly learning the solution, the difference between these two should be small. We compute this error across all N time steps and M simulated paths, then add an additional term to ensure that the output of the network at the final time step, $Y_N^m(\Theta)$, matches the known terminal condition $g(X_N^m)$. By minimizing the total loss, the network learns the parameters Θ that give a good approximation of both the solution over time and the correct final value.

II DNN for Gradient

This section is based on (Kapllani and Teng, 2024). Consider the following decoupled high-dimensional FBSDE,

$$\begin{cases} X_t = x_0 + \int_0^t a(s, X_s)ds + \int_0^t b(s, X_s)dW_s \\ Y_t = g(X_T) + \int_t^T f(s, X_s)ds - \int_t^T Z_s dW_s. \end{cases} \tag{12}$$

Here, $X_t := (X_t, Y_t, Z_t)$ and W_t is a d-dimensional Brownian motion. By the nonlinear Feynman-Kac, consider the semi-linear parabolic PDE

$$\frac{\partial u(t,x)}{\partial t} + \nabla_x u(t,x)a(t,x) + \frac{1}{2}Tr[bb^T Hess_x u(t,x)] + f(t,x,u,\nabla_x u(t,x)b(t,x)) = 0.$$

Assume that the regularity conditions of the FBSDE are satisfied, then the FBSDE can be represented $\mathbb{P} - a.s.$ by

$$Y_t = u(t, X_t), \quad Z_t = \nabla_x u(t, X_t)b(t, X_t).$$

To improve the accuracy of the first- and second- order approximations of the gradient, differential deep laerning can be used. This way, the stochastic gradient descent algorithm incorporates explicit information about the dynamics of Z_t. Applying the Malliavin derivative to the FBSDE yields a system of BSDEs,

$$\begin{cases} D_s X_t = 1_{s\le t}\left[b(s, X_s) + \int_s^t \nabla_x a(r, X_r)D_s X_r\, dr + \int_s^t \nabla_x b(r, X_r)D_s X_r\, dW_r\right], \\ D_s Y_t = 1_{s\le t}\left[\nabla_x g(X_T)D_s X_T + \int_t^T f_D(r, X_r, D_s X_r)\, dr - \int_t^T \left((D_s Z_r)^\top dW_r\right)^\top\right], \end{cases} \tag{13}$$

Here, $D_s X_t := (D_s X_t, D_s Y_t, D_s Z_t)$ and $f_D(t, X_t, D_s X_t) := \nabla_x f(t, X_t)D_s + \nabla_y f(t, X_t)D_s Y_t + \nabla_z f(t, X_t)D_s Z_t$. The solution to this BSDE system is a pair of triples of stochastic processes $\{X_t, Y_t, Z_t\}$ and $\{(D_s X_t, D_s Y_t, D_s Z_t)\}$ such that the system holds $\mathbb{P} - a.s.$

The Euler-Maruyama gives an approximation of the Malliavin derivative as

$$D_n X_m^\Delta = \begin{cases} 1_{n=m} b\left(t_n, X_n^\Delta\right), \quad 0 \le m \le n \le N \\ D_n X_{m-1}^\Delta + \nabla_x a\left(t_{m-1}, X_{m-1}^\Delta\right) D_n X_{m-1}^\Delta \Delta t_{m-1} \\ +\nabla_x b\left(t_{m-1}, X_{m-1}^\Delta\right) D_n X_{m-1}^\Delta \Delta W_{m-1}, \quad 0 \le n < m \le N \end{cases}$$

and

$$D_n Y_n^\Delta = D_n Y_{n+1}^\Delta + f_D\left(t_n, X_n^\Delta, D_n X_n^\Delta\right)\Delta t_n - D_n Z_n^\Delta \Delta W_n$$

An outline of the implementation of this algorithm is given by,

1. Generate approximations X_{n+1}^Δ for $n = 0, 1, \ldots, N-1$ and their discrete Malliavin derivatives $D_n X_n^\Delta, D_n X_{n+1}^\Delta$.

2. At each discrete time point t_n, for $n = 0, 1, \ldots, N$, use neural networks

$$\phi^y\left(\cdot;\theta^y\right) : \mathbb{R}^{1+d} \to \mathbb{R}, \quad \phi^z\left(\cdot;\theta^z\right) : \mathbb{R}^{1+d} \to \mathbb{R}^{1\times d}, \quad \phi^\gamma\left(\cdot;\theta^\gamma\right) : \mathbb{R}^{1+d} \to \mathbb{R}^{d\times d}$$

to approximate the discrete processes $\left(Y_n^\Delta, Z_n^\Delta, \Gamma_n^\Delta\right)$, where the network input is the time $t_n \in \mathbb{R}_+$ and the Markovian state $X_n^\Delta \in \mathbb{R}^d$. Namely,

$$Y_n^{\Delta,\theta} = \phi^y\left(t_n, X_n^\Delta; \theta^y\right), \quad Z_n^{\Delta,\theta} = \phi^z\left(t_n, X_n^\Delta; \theta^z\right), \quad \Gamma_n^{\Delta,\theta} = \phi^\gamma\left(t_n, X_n^\Delta; \theta^\gamma\right).$$

3. Train the parameters $\theta = (\theta^y, \theta^z, \theta^\gamma)$ using a global differential-type loss function that includes local terms enforcing the discretized BSDE dynamics:

$$\begin{aligned}
\mathbf{L}^\Delta(\theta) &:= \omega_1 \mathbf{L}^{y,\Delta}(\theta) + \omega_2 \mathbf{L}^{z,\Delta}(\theta), \\
\mathbf{L}^{y,\Delta}(\theta) &:= \mathbb{E}\left[\sum_{n=0}^{N-1}\left|Y_{n+1}^{\Delta,\theta} - Y_n^{\Delta,\theta} + f\left(t_n, \mathbf{X}_n^{\Delta,\theta}\right)\Delta t - Z_n^{\Delta,\theta}\Delta W_n\right|^2\right. \\
&\qquad \left. + \left|Y_N^{\Delta,\theta} - g\left(X_N^\Delta\right)\right|^2\right], \\
\mathbf{L}^{z,\Delta}(\theta) &:= \mathbb{E}\left[\sum_{n=0}^{N-1}\left|D_n Y_{n+1}^{\Delta,\theta} - Z_n^{\Delta,\theta} + f_D\left(t_n, \mathbf{X}_n^{\Delta,\theta}, \mathbf{D}_n \mathbf{X}_n^{\Delta,\theta}\right)\Delta t - \Gamma_n^{\Delta,\theta} D_n X_n^\Delta \Delta W_n\right|^2\right. \\
&\qquad \left. + \left|Z_N^{\Delta,\theta} - g_x\left(X_N^\Delta\right) b\left(t_N, X_N^\Delta\right)\right|^2\right],
\end{aligned}$$

where

$$D_n Y_{n+1}^{\Delta,\theta} = Z_{n+1}^{\Delta,\theta} b^{-1}\left(t_{n+1}, X_{n+1}^\Delta\right) D_n X_{n+1}^\Delta,$$

and $\omega_1, \omega_2 \in [0,1]$ with $\omega_1 + \omega_2 = 1$. For notational convenience:

$$\mathbf{X}_n^{\Delta,\theta} := \left(X_n^\Delta, Y_n^{\Delta,\theta}, Z_n^{\Delta,\theta}\right), \quad \mathbf{D}_n \mathbf{X}_n^{\Delta,\theta} := \left(D_n X_n^\Delta, Z_n^{\Delta,\theta}, \Gamma_n^{\Delta,\theta} D_n X_n^\Delta\right).$$

4. Approximate the optimal parameters $\theta^* \in \arg\min_{\theta\in\Theta} \mathbf{L}^\Delta(\theta)$ using a stochastic gradient descent method, and obtain the final estimated parameters $\hat{\theta}$. The final approximations for the discrete processes are then given by:

$$\left(Y_n^\Delta, Z_n^\Delta, \Gamma_n^\Delta\right) := \left(Y_n^{\Delta,\hat{\theta}}, Z_n^{\Delta,\hat{\theta}}, \Gamma_n^{\Delta,\hat{\theta}}\right), \quad \text{for } n = 0, 1, \ldots, N.$$

This method is faster and more efficient than AD.

Chapter 3: Ideas for Future Work

1. One idea for future work could be the development of advanced **stochastic correlation models** to enhance the pricing of high-dimensional options. As financial markets become increasingly complex, the interdependencies between multiple assets often evolve dynamically, especially during periods of market stress. Current models that rely on static correlations may fail to accurately capture these time-varying relationships. By incorporating stochastic correlation structures, future research could improve the precision of option pricing, offer better risk management tools, and provide more realistic models for portfolio optimization.

2. I've seen literature on delta-gamma hedging, but is there any literature on calculating the other greeks in a high-dimensional setting?

Chapter 4: Code

Code for implementing option pricing in 100-dimensions based on Han et al. (2018). Note to use Python 3.9 with TensorFlow 2.10.

```python
import numpy as np
import tensorflow as tf


class Equation(object):
    """
    Base class for defining Partial Differential Equation (PDE)-related functions.
    This class provides a template for sampling forward Stochastic Differential Equations (SDEs),
    defining the generator function (f_tf), and the terminal condition (g_tf).
    """

    def __init__(self, eqn_config):
        """
        Initialize the Equation class with configuration parameters.

        Args:
            eqn_config: An object containing configuration parameters such as:
                - dim: Dimension of the problem.
                - total_time: Total time horizon for the PDE.
                - num_time_interval: Number of time intervals for discretization.
        """
        self.dim = eqn_config.dim  # Dimension of the problem
        self.total_time = eqn_config.total_time  # Total time horizon
        self.num_time_interval = eqn_config.num_time_interval  # Number of time intervals
        self.delta_t = self.total_time / self.num_time_interval  # Time step size
        self.sqrt_delta_t = np.sqrt(self.delta_t)  # Square root of the time step size
        self.y_init = None  # Initial value for the solution (to be defined in subclasses)

    def sample(self, num_sample):
        """
        Sample forward SDE paths. This method must be implemented in subclasses.

        Args:
            num_sample: Number of samples to generate.

        Raises:
            NotImplementedError: If the method is not implemented in a subclass.
        """
        raise NotImplementedError

    def f_tf(self, t, x, y, z):
        """
        Generator function in the PDE. This method must be implemented in subclasses.

        Args:
            t: Current time step.
            x: State variable.
            y: Solution variable.
            z: Gradient of the solution.

        Raises:
            NotImplementedError: If the method is not implemented in a subclass.
        """
        raise NotImplementedError

    def g_tf(self, t, x):
        """
        Terminal condition of the PDE. This method must be implemented in subclasses.

        Args:
            t: Current time step.
            x: State variable.

```

```
        Raises:
            NotImplementedError: If the method is not implemented in a subclass.
        """
        raise NotImplementedError


class PricingDefaultRisk(Equation):
    """
    Nonlinear Black-Scholes equation with default risk.
    This model is based on the PNAS paper: doi.org/10.1073/pnas.1718942115.
    """

    def __init__(self, eqn_config):
        """
        Initialize the PricingDefaultRisk class with specific parameters.

        Args:
            eqn_config: Configuration object containing problem parameters.
        """
        super(PricingDefaultRisk, self).__init__(eqn_config)
        self.x_init = np.ones(self.dim) * 100.0  # Initial value of the asset
        self.sigma = 0.2  # Volatility of the asset
        self.rate = 0.02  # Interest rate
        self.delta = 2.0 / 3  # Risk aversion parameter
        self.gammah = 0.2  # High default intensity
        self.gammal = 0.02  # Low default intensity
        self.mu_bar = 0.02  # Drift term
        self.vh = 50.0  # Threshold for high default intensity
        self.vl = 70.0  # Threshold for low default intensity
        self.slope = (self.gammah - self.gammal) / (self.vh - self.vl)  # Slope of the piecewise linear
    function

    def sample(self, num_sample):
        """
        Generate sample paths for the forward SDE.

        Args:
            num_sample: Number of sample paths to generate.

        Returns:
            dw_sample: Brownian motion increments.
            x_sample: Simulated asset paths.
        """
        dw_sample = np.random.normal(size=[num_sample, self.dim, self.num_time_interval]) * self.sqrt_delta_t
        x_sample = np.zeros([num_sample, self.dim, self.num_time_interval + 1])
        x_sample[:, :, 0] = np.ones([num_sample, self.dim]) * self.x_init  # Initialize asset paths
        for i in range(self.num_time_interval):
            x_sample[:, :, i + 1] = (1 + self.mu_bar * self.delta_t) * x_sample[:, :, i] + (
                self.sigma * x_sample[:, :, i] * dw_sample[:, :, i])  # Euler-Maruyama scheme
        return dw_sample, x_sample

    def f_tf(self, t, x, y, z):
        """
        Generator function for the PDE.

        Args:
            t: Current time step.
            x: State variable.
            y: Solution variable.
            z: Gradient of the solution.

        Returns:
            Tensor representing the generator function value.
        """
        # Piecewise linear function for default intensity
        piecewise_linear = tf.nn.relu(
            tf.nn.relu(y - self.vh) * self.slope + self.gammah - self.gammal) + self.gammal
        return (-(1 - self.delta) * piecewise_linear - self.rate) * y
```

```
131
132     def g_tf(self, t, x):
133         """
134         Terminal condition for the PDE.
135
136         Args:
137             t: Current time step.
138             x: State variable.
139
140         Returns:
141             Tensor representing the terminal condition value.
142         """
143         return tf.reduce_min(x, 1, keepdims=True)  # Minimum value of the state variable
144
145
146 class PricingDiffRate(Equation):
147     """
148     Nonlinear Black-Scholes equation with different interest rates for borrowing and lending.
149     This model is based on Section 4.4 of the Comm. Math. Stat. paper: doi.org/10.1007/s40304-017-0117-6.
150     """
151
152     def __init__(self, eqn_config):
153         """
154         Initialize the PricingDiffRate class with specific parameters.
155
156         Args:
157             eqn_config: Configuration object containing problem parameters.
158         """
159         super(PricingDiffRate, self).__init__(eqn_config)
160         self.x_init = np.ones(self.dim) * 100  # Initial value of the asset
161         self.sigma = 0.2  # Volatility of the asset
162         self.mu_bar = 0.06  # Drift term
163         self.rl = 0.04  # Lending interest rate
164         self.rb = 0.06  # Borrowing interest rate
165         self.alpha = 1.0 / self.dim  # Weighting factor for dimensions
166
167     def sample(self, num_sample):
168         """
169         Generate sample paths for the forward SDE.
170
171         Args:
172             num_sample: Number of sample paths to generate.
173
174         Returns:
175             dw_sample: Brownian motion increments.
176             x_sample: Simulated asset paths.
177         """
178         dw_sample = np.random.normal(size=[num_sample, self.dim, self.num_time_interval]) * self.sqrt_delta_t
179         x_sample = np.zeros([num_sample, self.dim, self.num_time_interval + 1])
180         x_sample[:, :, 0] = np.ones([num_sample, self.dim]) * self.x_init  # Initialize asset paths
181         factor = np.exp((self.mu_bar - (self.sigma**2) / 2) * self.delta_t)  # Drift factor
182         for i in range(self.num_time_interval):
183             x_sample[:, :, i + 1] = (factor * np.exp(self.sigma * dw_sample[:, :, i])) * x_sample[:, :, i]
184         return dw_sample, x_sample
185
186     def f_tf(self, t, x, y, z):
187         """
188         Generator function for the PDE.
189
190         Args:
191             t: Current time step.
192             x: State variable.
193             y: Solution variable.
194             z: Gradient of the solution.
195
196         Returns:
197             Tensor representing the generator function value.
198         """
```

```
        temp = tf.reduce_sum(z, 1, keepdims=True) / self.sigma  # Weighted sum of gradients
        return -self.rl * y - (self.mu_bar - self.rl) * temp + (
            (self.rb - self.rl) * tf.maximum(temp - y, 0))  # Nonlinear term for borrowing and lending rates

    def g_tf(self, t, x):
        """
        Terminal condition for the PDE.

        Args:
            t: Current time step.
            x: State variable.

        Returns:
            Tensor representing the terminal condition value.
        """
        temp = tf.reduce_max(x, 1, keepdims=True)  # Maximum value of the state variable
        return tf.maximum(temp - 120, 0) - 2 * tf.maximum(temp - 150, 0)  # Payoff function
```

References

El Karoui, N., Pardoux, E., and Quenez, M. (1997a). *Reflected Backward SDEs and American Options*, page 215–231. Publications of the Newton Institute. Cambridge University Press.

El Karoui, N., Peng, S., and Quenez, M. C. (1997b). Backward stochastic differential equations in finance. *Mathematical Finance*, 7(1):1–71.

Gardiner, C. W. (2004). *Handbook of stochastic methods*. Springer series in synergetics. Springer, Berlin, Germany, 3 edition.

Gobet, E. (2020). *Monte-Carlo methods and stochastic processes*. Chapman & Hall/CRC, Philadelphia, PA.

Güler, B., Laignelet, A., and Parpas, P. (2019). Towards robust and stable deep learning algorithms for forward backward stochastic differential equations.

Han, J., Jentzen, A., and E, W. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510.

Kapllani, L. and Teng, L. (2024). A forward differential deep learning-based algorithm for solving high-dimensional nonlinear backward stochastic differential equations.

Karoui, N. E., Kapoudjian, C., Pardoux, E., Peng, S., and Quenez, M. C. (1997). Reflected solutions of backward SDE's, and related obstacle problems for PDE's. *The Annals of Probability*, 25(2):702 – 737.

Kohn, R. (2011). *PDE for Finance Notes, Section 1*. `https://math.nyu.edu/~kohn/pde_finance.html`.

Raissi, M. (2018). Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations.