Deep Gradient Flow Methods for Option Pricing in Diffusion Models

SIAM Conference on Financial Mathematics and Engineering

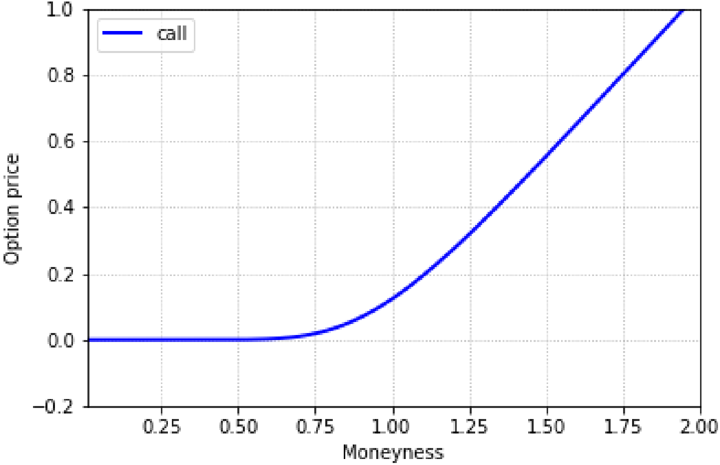
Jasper Rou

Antonis Papapantoleon & Emmanuil Georgoulis

June 8, 2023

Registration and travel support for this presentation was provided by the Society for Industrial and Applied Mathematics.

Option


call
Option price
Moneyness
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Pricing

Feynman-Kac formula:

$$\frac{\partial u}{\partial t} + \sum_{i,j=0}^{n} a^{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} - \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} - ru = 0,$$

$$u(T) = \Phi(S_T)$$

Pricing

Feynman-Kac formula:

$$\frac{\partial u}{\partial t} + \sum_{i,j=0}^{n} a^{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} - \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} - ru = 0,$$

$$u(T) = \Phi(S_T)$$

Can we solve this PDE using a neural network?

Deep Galerkin Method [1]

$$\frac{\partial u}{\partial t} + \sum_{i,j=0}^{n} a^{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} - \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} - ru = 0,$$

$$u(T) = \Phi(S_T)$$

[1] Justin Sirignano and Konstantinos Spiliopoulos (2018). "DGM: A deep learning algorithm for solving partial differential equations". In: *Journal of computational physics* 375, pp. 1339–1364

Deep Galerkin Method [1]

$$\frac{\partial u}{\partial t} + \sum_{i,j=0}^{n} a^{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} - \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} - ru = 0,$$

$$u(T) = \Phi(S_T)$$

Minimize

$$\left\| \frac{\partial u}{\partial t} + \sum_{i,j=0}^{n} a^{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} - \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} - ru \right\|^2_{[0,T]\times\Omega} + \| u(T) - \Phi(S_T) \|^2_{\Omega}.$$

[1] Justin Sirignano and Konstantinos Spiliopoulos (2018). "DGM: A deep learning algorithm for solving partial differential equations". In: *Journal of computational physics* 375, pp. 1339–1364

Splitting method

$$\frac{\partial u}{\partial t} = -\sum_{i,j=0}^{n} a^{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} + ru$$

Splitting method

$$\frac{\partial u}{\partial t} = -\sum_{i,j=0}^{n} a^{ij}\frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} + ru$$

$$= -\sum_{i,j=0}^{n} \frac{\partial}{\partial x_j}\left(a^{ij}\frac{\partial u}{\partial x_i}\right) + \sum_{i,j=0}^{n} \frac{\partial a^{ij}}{\partial x_j}\frac{\partial u}{\partial x_i} + \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} + ru$$

Splitting method

$$
\begin{aligned}
\frac{\partial u}{\partial t} &= -\sum_{i,j=0}^{n} a^{ij}\frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} + ru \\
&= -\sum_{i,j=0}^{n} \frac{\partial}{\partial x_j}\left(a^{ij}\frac{\partial u}{\partial x_i}\right) + \sum_{i,j=0}^{n} \frac{\partial a^{ij}}{\partial x_j}\frac{\partial u}{\partial x_i} + \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} + ru \\
&= -\sum_{i,j=0}^{n} \frac{\partial}{\partial x_j}\left(a^{ij}\frac{\partial u}{\partial x_i}\right) + \sum_{i=0}^{n}\left(b^i + \sum_{j=0}^{n}\frac{\partial a^{ij}}{\partial x_j}\right)\frac{\partial u}{\partial x_i} + ru.
\end{aligned}
$$

Splitting method

$$
\begin{aligned}
\frac{\partial u}{\partial t} &= -\sum_{i,j=0}^{n} a^{ij}\frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} + ru \\
&= -\sum_{i,j=0}^{n} \frac{\partial}{\partial x_j}\left(a^{ij}\frac{\partial u}{\partial x_i}\right) + \sum_{i,j=0}^{n} \frac{\partial a^{ij}}{\partial x_j}\frac{\partial u}{\partial x_i} + \sum_{i=0}^{n} b^i \frac{\partial u}{\partial x_i} + ru \\
&= -\sum_{i,j=0}^{n} \frac{\partial}{\partial x_j}\left(a^{ij}\frac{\partial u}{\partial x_i}\right) + \sum_{i=0}^{n}\left(b^i + \sum_{j=0}^{n}\frac{\partial a^{ij}}{\partial x_j}\right)\frac{\partial u}{\partial x_i} + ru. \\
&= -\nabla \cdot (A\nabla u) + ru + F(u), \\
F(u) &= \mathbf{b} \cdot \nabla u.
\end{aligned}
$$

Time Deep Nitsche Method [2]

$$\begin{cases} u_\tau - \nabla \cdot (A \nabla u) + ru + F(u) = 0, & (\tau, \mathbf{x}) \in [0, T] \times \Omega, \\ u(0, \mathbf{x}) = \Phi(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\tau, \mathbf{x}) = g_D(t, \mathbf{x}), & (\tau, \mathbf{x}) \in [0, T] \times \Gamma_D, \\ \mathbf{n} \cdot A \nabla u(\tau, \mathbf{x}) = g_N(\tau, \mathbf{x}), & (\tau, \mathbf{x}) \in [0, T] \times \Gamma_N. \end{cases}$$

[2] Emmanuil H Georgoulis, Michail Loulakis, and Asterios Tsiourvas (2023). "Discrete gradient flow approximations of high dimensional evolution partial differential equations via deep neural networks". In: *Communications in Nonlinear Science and Numerical Simulation* 117, p. 106893

Time Deep Nitsche Method [2]

$$\begin{cases} u_\tau - \nabla \cdot (A\nabla u) + ru + F(u) = 0, & (\tau, \mathbf{x}) \in [0, T] \times \Omega, \\ u(0, \mathbf{x}) = \Phi(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\tau, \mathbf{x}) = g_D(t, \mathbf{x}), & (\tau, \mathbf{x}) \in [0, T] \times \Gamma_D, \\ \mathbf{n} \cdot A\nabla u(\tau, \mathbf{x}) = g_N(\tau, \mathbf{x}), & (\tau, \mathbf{x}) \in [0, T] \times \Gamma_N. \end{cases}$$

- Divide $[0, T]$ in intervals $(\tau_{k-1}, \tau_k]$ with $h = \tau_k - \tau_{k-1}$
- Seek approximations $f^k(\mathbf{x}; \theta)$ such that

$$\frac{f^k - f^{k-1}}{h} - \nabla \cdot \left(A\nabla f^k\right) + rf^k + F\left(f^{k-1}\right) = 0.$$

[2] Emmanuil H Georgoulis, Michail Loulakis, and Asterios Tsiourvas (2023). "Discrete gradient flow approximations of high dimensional evolution partial differential equations via deep neural networks". In: *Communications in Nonlinear Science and Numerical Simulation* 117, p. 106893

Time Deep **Nitsche** Method

$$\frac{f^k - f^{k-1}}{h} - \nabla \cdot \left(A \nabla f^k \right) + r f^k + F = 0.$$

Time Deep **Nitsche** Method

$$\frac{f^k - f^{k-1}}{h} - \nabla \cdot \left(A \nabla f^k \right) + r f^k + F = 0.$$

$$f^k = \arg\min_{w} L(w),$$

$$L(w) = \frac{1}{2} \left\| w - f^{k-1} \right\|^2_{L^2(\Omega)} + h I(w).$$

Time Deep **Nitsche** Method

$$\frac{f^k - f^{k-1}}{h} - \nabla \cdot \left(A \nabla f^k \right) + r f^k + F = 0.$$

$$f^k = \arg\min_w L(w),$$

$$L(w) = \frac{1}{2} \left\| w - f^{k-1} \right\|^2_{L^2(\Omega)} + hI(w).$$

$$0 = \int_\Omega \left(-\nabla \cdot (A \nabla u) + ru + F \right) v d\mathbf{x} \qquad ,$$

Time Deep **Nitsche** Method

$$\frac{f^k - f^{k-1}}{h} - \nabla \cdot \left(A \nabla f^k \right) + r f^k + F = 0.$$

$$f^k = \arg\min_w L(w),$$

$$L(w) = \frac{1}{2} \left\| w - f^{k-1} \right\|^2_{L^2(\Omega)} + h I(w).$$

$$0 = \int_\Omega \left(-\nabla \cdot (A \nabla u) + r u + F \right) v d\mathbf{x} = i'(0),$$

$$i(\tau) = I(u + \tau v)$$

$$I(u) = \int_\Omega \frac{1}{2} \left((\nabla u)^T A \nabla u + r u^2 \right) + F u dx - \int_{\Gamma_N} g_N u ds$$

$$- \int_{\Gamma_D} \mathbf{n} \cdot A \nabla u (u - g_D) ds.$$

Algorithm

1: Initialize θ_0^0.

Algorithm

1: Initialize θ_0^0.
2: Initialize a neural network approximating the initial condition

$$f^0 = \arg\min_{w} \|w - \Phi(\mathbf{X})\|_{L_2(\Omega)}.$$

Algorithm

1: Initialize θ_0^0.

2: Initialize a neural network approximating the initial condition

$$f^0 = \arg\min_w \|w - \Phi(\mathbf{X})\|_{L_2(\Omega)}.$$

3: **for** each time step $k = 1, ..., N_t$ **do**

4: Initialize $\theta_0^k = \theta^{k-1}$.

Algorithm

Initialize θ_0^0.
Initialize a neural network approximating the initial condition

$$f^0 = \arg\min_{w} \|w - \Phi(\mathbf{X})\|_{L_2(\Omega)}.$$

for each time step $k = 1, ..., N_t$ **do**
    Initialize $\theta_0^k = \theta^{k-1}$.
    for each sampling stage **do**
        Generate random points $\mathbf{x}^i$ for training.

Algorithm

Initialize θ_0^0.

Initialize a neural network approximating the initial condition

$$f^0 = \arg\min_w \|w - \Phi(\mathbf{X})\|_{L_2(\Omega)}.$$

for each time step $k = 1, ..., N_t$ **do**

  Initialize $\theta_0^k = \theta^{k-1}$.

  for each sampling stage **do**

    Generate random points $\mathbf{x}^i$ for training.

    Calculate the cost functional $L(f(\theta_n^k; \mathbf{x}^i))$.

Algorithm

Initialize θ_0^0.
Initialize a neural network approximating the initial condition

$$f^0 = \arg\min_{w} \|w - \Phi(\mathbf{X})\|_{L_2(\Omega)}.$$

for each time step $k = 1, ..., N_t$ **do**
    Initialize $\theta_0^k = \theta^{k-1}$.
    for each sampling stage **do**
        Generate random points $\mathbf{x}^i$ for training.
        Calculate the cost functional $L(f(\theta_n^k; \mathbf{x}^i))$.
        Take a descent step $\theta_{n+1}^k = \theta_n^k - \alpha_n \nabla_\theta L(f(\theta_n^k; \mathbf{x}^i))$.
    end for
end for

Black-Scholes

$$dS_t = rS_t dt + \sigma S_t dW_t, \quad S_0 > 0,$$

Black-Scholes

$$dS_t = rS_t dt + \sigma S_t dW_t, \quad S_0 > 0,$$

$$0 = u_\tau - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2} - rS\frac{\partial u}{\partial S} + ru$$

Black-Scholes

$$dS_t = rS_t dt + \sigma S_t dW_t, \quad S_0 > 0,$$

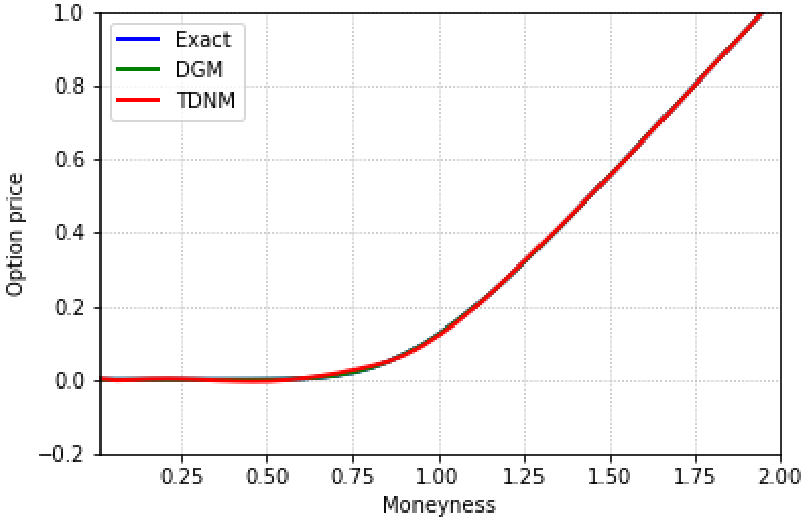
$$\begin{aligned} 0 &= u_\tau - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2} - rS\frac{\partial u}{\partial S} + ru \\ &= u_\tau - \frac{\partial}{\partial S}\left(\frac{1}{2}\sigma^2 S^2 \frac{\partial u}{\partial S}\right) + \sigma^2 S\frac{\partial u}{\partial S} - rS\frac{\partial u}{\partial S} + ru \end{aligned}$$

Black-Scholes

$$dS_t = rS_t dt + \sigma S_t dW_t, \quad S_0 > 0,$$

$$\begin{aligned} 0 &= u_\tau - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2} - rS\frac{\partial u}{\partial S} + ru \\ &= u_\tau - \frac{\partial}{\partial S}\left(\frac{1}{2}\sigma^2 S^2 \frac{\partial u}{\partial S}\right) + \sigma^2 S \frac{\partial u}{\partial S} - rS\frac{\partial u}{\partial S} + ru \\ &= u_\tau - \frac{\partial}{\partial S}\left(\frac{1}{2}\sigma^2 S^2 \frac{\partial u}{\partial S}\right) + \left(\sigma^2 - r\right) S \frac{\partial u}{\partial S} + ru. \end{aligned}$$

Black-Scholes

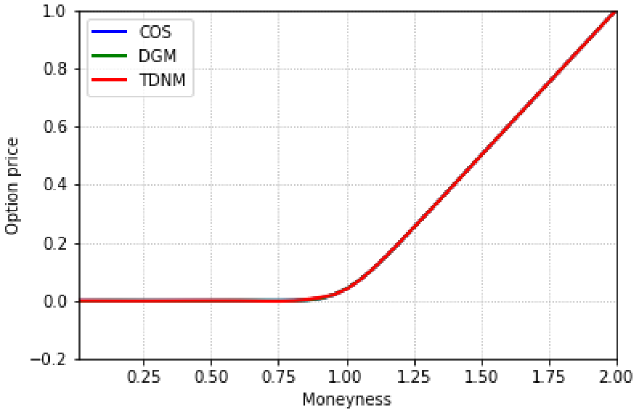


Heston

$$
\begin{aligned}
dS_t &= rS_t dt + \sqrt{V_t} S_t dW_t, \quad S_0 > 0, \\
dV_t &= \kappa(\theta - V_t) dt + \eta \sqrt{V_t} dB_t, \quad V_0 > 0.
\end{aligned}
$$

Heston

$$dS_t = rS_t dt + \sqrt{V_t} S_t dW_t, \quad S_0 > 0,$$
$$dV_t = \kappa(\theta - V_t)dt + \eta\sqrt{V_t} dB_t, \quad V_0 > 0.$$



Lifted Heston [3]

$$dS_t = rS_t dt + \sqrt{V_t^n} S_t dW_t, \qquad S_0 > 0,$$

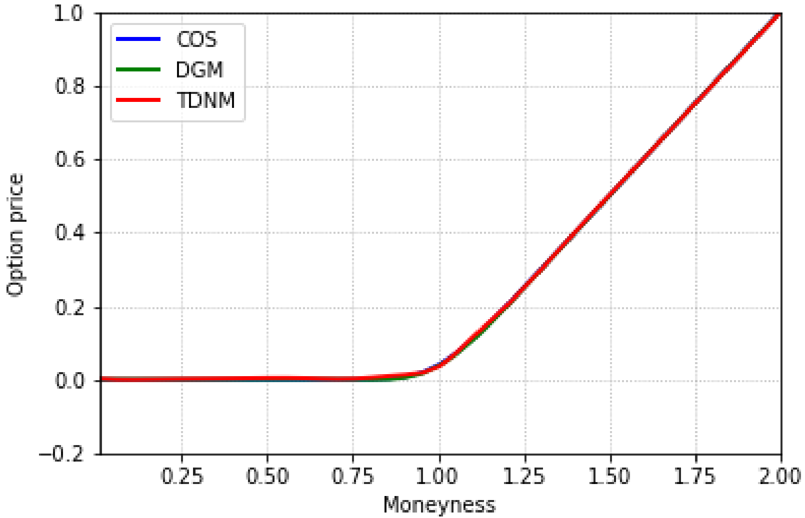
$$V_t^n = g^n(t) + \sum_{i=1}^{n} c_i^n V_t^{n,i},$$

$$dV_t^{n,i} = -\left(\gamma_i^n V_t^{n,i} + \lambda V_t^n\right) dt + \eta \sqrt{V_t^n} dB_t, \quad V_0^{n,i} = 0,$$

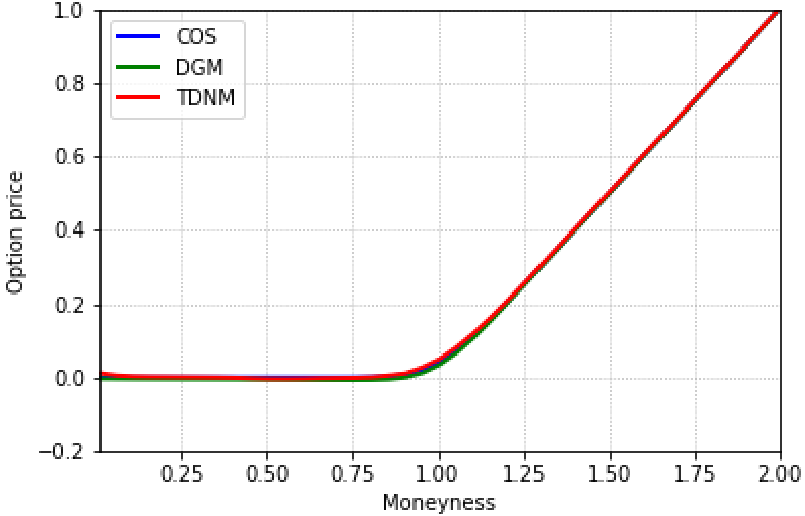
$$g^n(t) = V_0 + \lambda\theta \sum_{i=1}^{n} c_i^n \int_0^t e^{-\gamma_i^n (t-s)} ds,$$

[3] Eduardo Abi Jaber (2019). "Lifting the Heston model". In: *Quantitative Finance* 19.12, pp. 1995–2013

Lifted Heston, $n = 1$



Lifted Heston, $n = 5$



Running times

Method	Black-Scholes	Heston	LH, n=1	LH, n=5
DGM	0.34	0.65	2.0	4.4
TDNM	0.55	0.71	0.76	1.2

Table: Training time (10^4 seconds)

Running times

Method	Black-Scholes	Heston	LH, n=1	LH, n=5
DGM	0.34	0.65	2.0	4.4
TDNM	0.55	0.71	0.76	1.2

Table: Training time (10^4 seconds)

Method	Black-Scholes	Heston	LH, n=1	LH, n=5
Exact/COS	0.00051	0.013	6.9	6.9
DGM	0.011	0.011	0.011	0.011
TDNM	0.027	0.036	0.053	0.052

Table: Computing time (seconds)

Deep Gradient Flow Methods for Option Pricing in Diffusion Models

SIAM Conference on Financial Mathematics and Engineering

Jasper Rou

June 8, 2023

j.g.rou@tudelft.nl www.jasperrou.nl