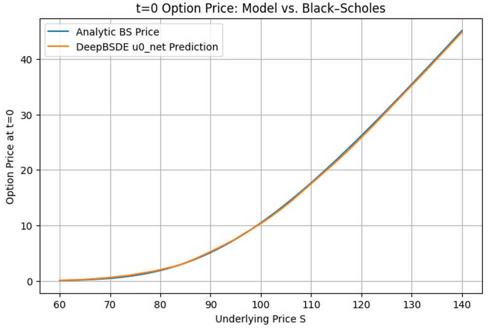
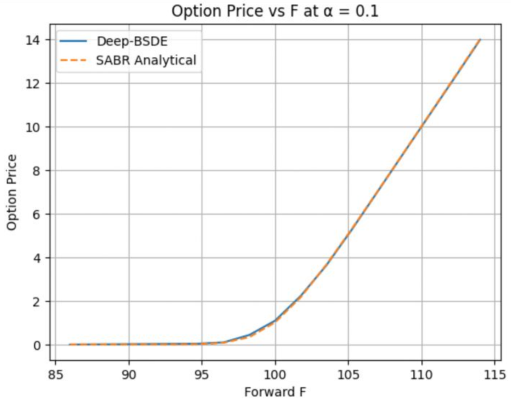
Solving Feynman-kac and Fokker-Planck equations via Neural Networks

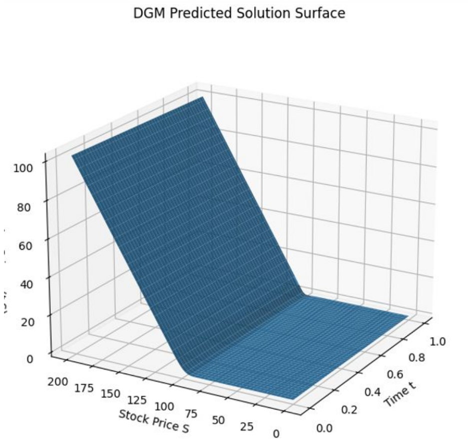
Annie Zhang

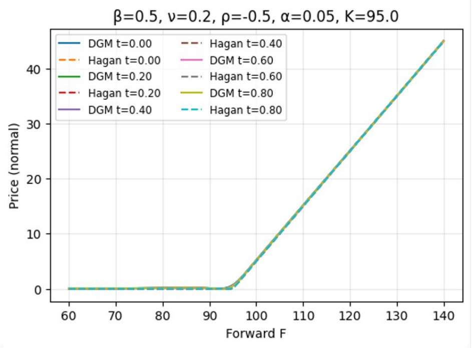
Deep Backward SDE



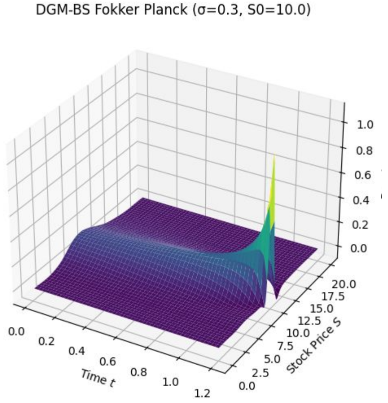


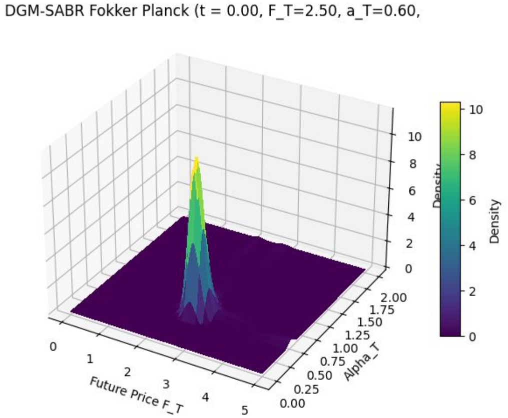
Deep Galerkin Method



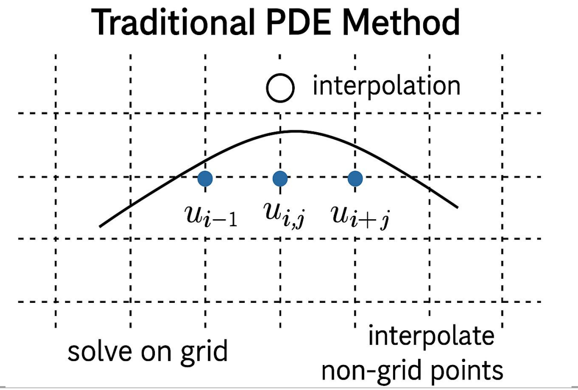


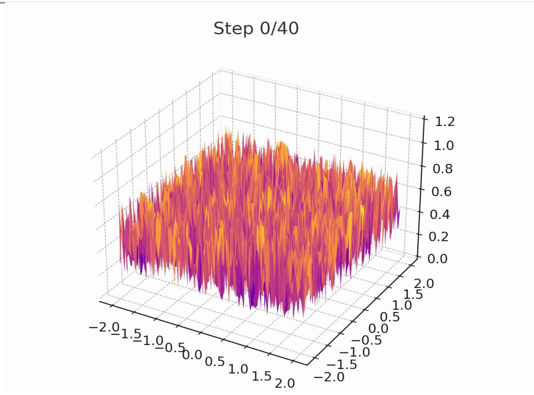
Deep Galerkin Method





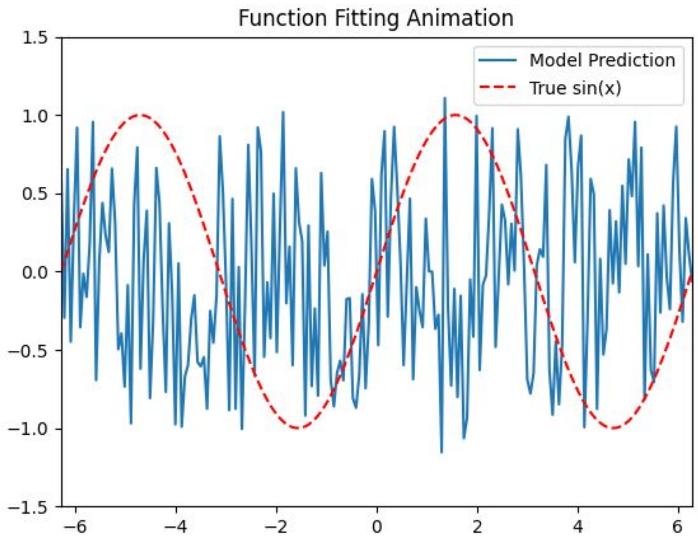
Equation	Method	Model	Input Parameters	Output	Training Time
Feynman-Kac	Black-Scholes	Deep Backward SDE	S_0 for fixed σ, K, r	NPV at $t = 0$ $\frac{\partial u}{\partial S}$(Delta) at $t_1, t_2 \ldots t_N$	~ 3 min
	SABR	Deep Backward SDE	F_0, α_0 for fixed β, ν, ρ, K	NPV at $t = 0$ $\frac{\partial u}{\partial F}$(Delta) and $\frac{\partial u}{\partial \alpha}$(Vega) at $t_1, t_2 \ldots t_N$	~ 5 min
	Black-Scholes	Deep Galerkin method (DGM)	t, S, r, σ, K	NPV at any time t_n w.r.t to S, r, σ, K. Greeks from derivatives	~ 7 min
	SABR	Deep Galerkin method (DGM)	$t, F, \alpha, \beta, \rho, \nu, K$	NPV at any time t_n w.r.t to $F, \alpha, \beta, \rho, \nu, K$. Greeks from derivatives	~ 10 min
Fokker-Planck	Black-Scholes	Deep Galerkin method (DGM)	t, S, r, σ, K	CDF $F_{S_t}(t, S, r, \sigma, S_0)$. PDF from $\frac{\partial F_{S_t}}{\partial S_T}$	~ 15 min
	SABR	Deep Galerkin method (DGM)	$t, F, \alpha, \beta, \rho, \nu, K$	Joint CDF $G_{F_t, \alpha_t}(t, F, \alpha, \beta, \rho, \nu, K, \mathrm{F}_0, \alpha_0)$ PDF from $\frac{\partial^2 G_{F_t, \alpha_t}}{\partial F_T \partial \alpha_T}$	~ 30 min



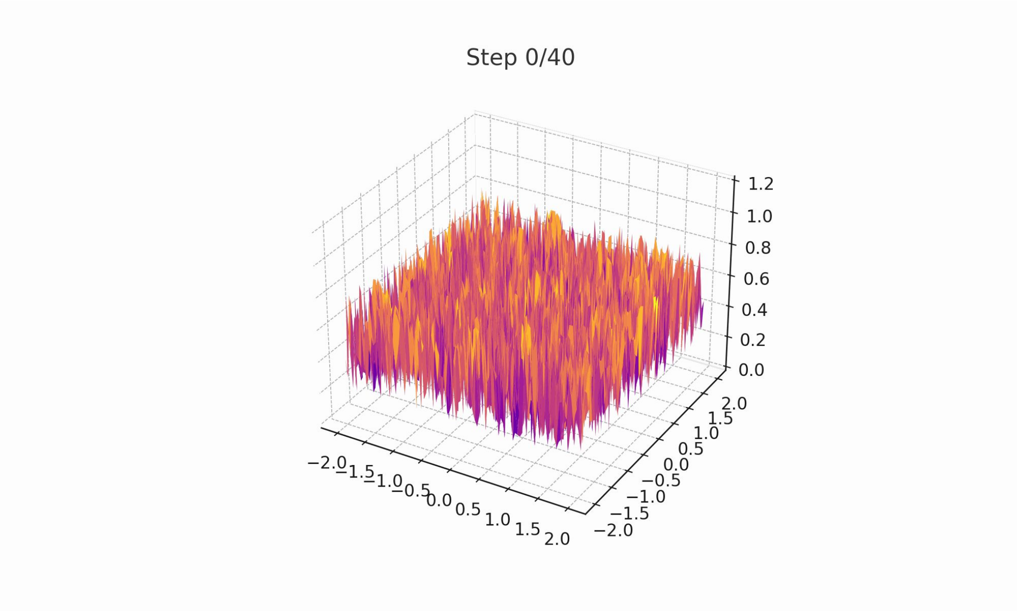


Require grid construction (e.g. finite difference)	Mesh-Free — no need to construct grids, possible to go high dimension
One setting per run - re-solve if r,K,σ change	One model fits multiple market settings (e.g. output option price as a function of (t, S, α, ρ, β, v, T) in SABR)
Discrete time space - approximate Greeks via finite difference	Continuous function - extract Greeks via automatic differentiation (Autograd in PyTorch).
Compute price only at grid points - interpolation required	Option price at any (t, S) no interpolation required

Using Neural Network Learning Functions

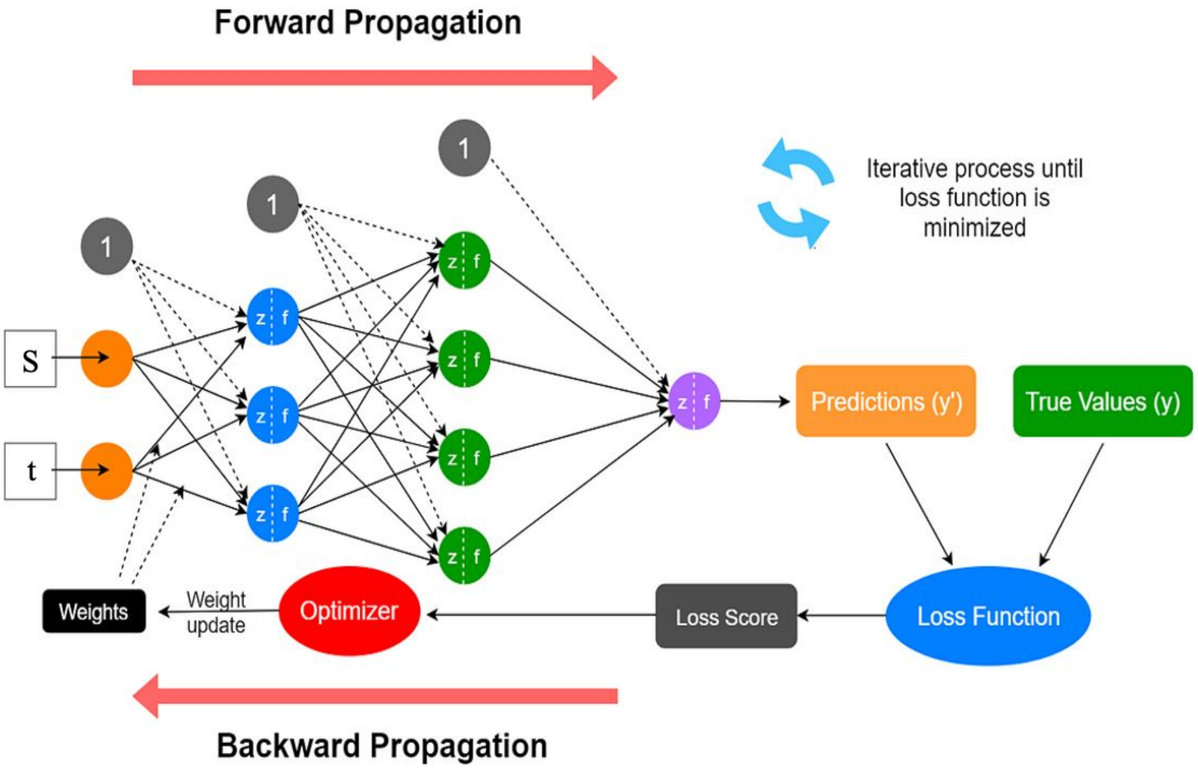


Neural Network Learning sin(x)



Neural Network Learning Gaussian Surface

How Do Neural Networks Learn?



How Training Works:

- Inputs like (t, S, volatility...) go into the network
- The network **predicts a option price**
- A **loss function** compares prediction to true price
- The **optimizer** adjusts weights to reduce the error
- This process **repeats** until **error is minimized**

Goal:

- Train the network to **map** (t, S, volatility…) → Option Price

Deep BSDE - Black Scholes

We consider the Black–Scholes PDE for a European call option:

$$\frac{\partial u}{\partial t}(t,S) + rS\frac{\partial u}{\partial S}(t,S) + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2}(t,S) = ru(t,S), \quad u(T,S) = (S-K)^+$$

Under the risk-neutral measure $\mathbb{Q}$, the forward SDE for the stock price is:

$$dS_t = rS_t\,dt + \sigma S_t\,d\widetilde{W}_t, \quad S_0 = s$$

Define the value process:

$$Y_t := u(t,S_t) = \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-t)}(S_T-K)^+ \mid \mathcal{F}_t\right]$$

Applying Itô's lemma to $Y_t = u(t,S_t)$, we get:

$$dY_t = \left(\frac{\partial u}{\partial t} + rS_t\frac{\partial u}{\partial S} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 u}{\partial S^2}\right)dt + \sigma S_t \frac{\partial u}{\partial S}(t,S_t)\,d\widetilde{W}_t$$

Since u satisfies the Black–Scholes PDE, we have:

$$dY_t = rY_t\,dt + \sigma S_t \delta(t,S_t)\,d\widetilde{W}_t, \quad \text{where } \delta(t,S_t) := \frac{\partial u}{\partial S}(t,S_t)$$

Deep BSDE - Black Scholes

Discretize the time interval $[0, T]$ into N steps:

$$0 = t_0 < t_1 < \cdots < t_N = T, \quad \Delta t = t_{n+1} - t_n$$

Simulate the forward SDE using Euler–Maruyama:

$$S_{n+1} = S_n + rS_n \, \Delta t + \sigma S_n \, \Delta W_n, \quad \Delta W_n \sim \mathcal{N}(0, \Delta t)$$

Discretize the BSDE:

$$Y_{n+1} = (1 + r\Delta t) \, Y_n + \sigma S_n \cdot \delta(t_n, S_n) \cdot \Delta W_n$$

In the Deep BSDE method:

- Y_0 is a learnable scalar parameter.
- $\delta(t_n, S_n)$ is predicted by a neural network (MLP).
- The above update is applied recursively from t_0 to t_N.

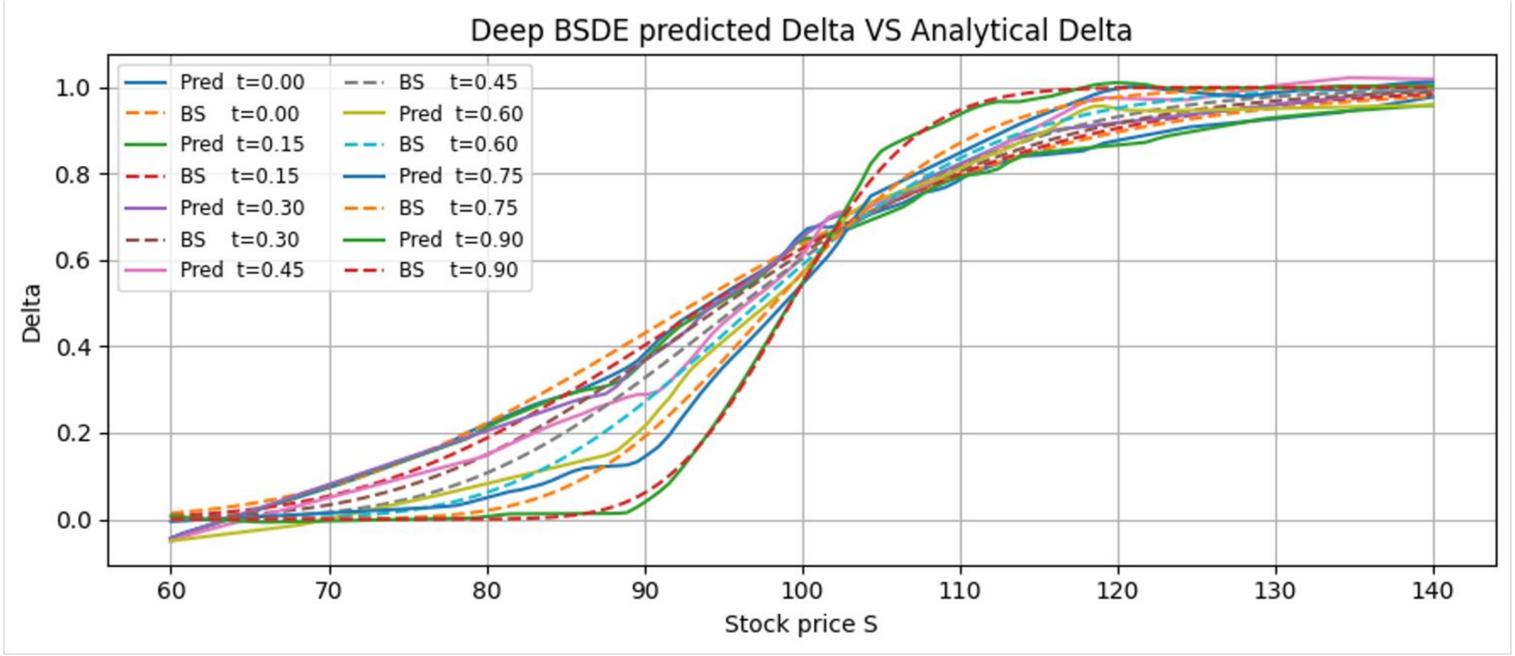
At maturity $t_N = T$, the terminal condition is:

$$Y_T \approx (S_T - K)^+$$

We define the loss function as the expected squared error at the terminal time:

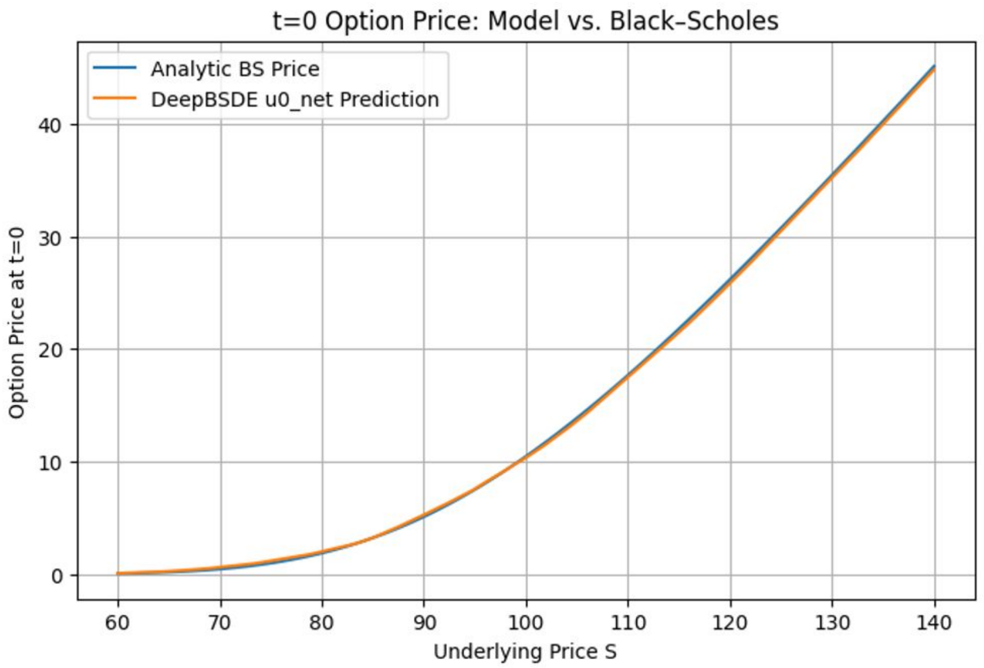
$$\mathcal{L} = \mathbb{E}\left[\left(Y_T - (S_T - K)^+\right)^2\right]$$

Result & Bench Mark



Predicted Deltas match analytical values well across time, especially near the strike, indicating that the model correctly captures the **hedging ratio Delta** at each time step.

Result & Benchmark



- The model learns accurate Deltas at each time step for hedging. (As shown previously)
- These Deltas are used to simulate terminal hedge values.
- We only need to train another MLP at t = 0 to output the correct initial NPV as a function of initial stock price S0, which is the option price we want at t = 0

Deep BSDE - SABR

We consider the SABR stochastic volatility model:

$$\begin{cases} dF_t = \alpha_t F_t^{\beta}\, dW_t \\ d\alpha_t = \nu \alpha_t\, dZ_t \\ dW_t\, dZ_t = \rho\, dt \end{cases}$$

We define the value function:

$$u(t, F, \alpha) = \mathbb{E}[(F_T - K)^+ \mid F_t = F, \alpha_t = \alpha]$$

Under the risk-neutral measure, the function $u(t, F, \alpha)$ satisfies the following parabolic PDE:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\alpha^2 F^{2\beta} \frac{\partial^2 u}{\partial F^2} + \rho\nu\alpha^2 F^{\beta} \frac{\partial^2 u}{\partial F\, \partial \alpha} + \frac{1}{2}\nu^2 \alpha^2 \frac{\partial^2 u}{\partial \alpha^2} = 0$$

with terminal condition:

$$u(T, F, \alpha) = (F - K)^+$$

Deep BSDE - SABR

We apply Itô's lemma to the composite process $Y_t = u(t, F_t, \alpha_t)$:

$$
\begin{aligned}
du(t, F_t, \alpha_t) = {} & \frac{\partial u}{\partial t}\, dt + \frac{\partial u}{\partial F}\, dF_t + \frac{\partial u}{\partial \alpha}\, d\alpha_t \\
& + \frac{1}{2}\frac{\partial^2 u}{\partial F^2}\, d\langle F\rangle_t + \frac{1}{2}\frac{\partial^2 u}{\partial \alpha^2}\, d\langle \alpha\rangle_t + \frac{\partial^2 u}{\partial F \partial \alpha}\, d\langle F, \alpha\rangle_t
\end{aligned}
$$

We compute the quadratic variations:

$$
\begin{aligned}
d\langle F\rangle_t &= \alpha_t^2 F_t^{2\beta}\, dt \\
d\langle \alpha\rangle_t &= \nu^2 \alpha_t^2\, dt \\
d\langle F, \alpha\rangle_t &= \rho\nu\alpha_t^2 F_t^{\beta}\, dt
\end{aligned}
$$

Substitute all components into the Itô expansion:

$$
\begin{aligned}
du = {} & \left[\frac{\partial u}{\partial t} + \frac{1}{2}\alpha_t^2 F_t^{2\beta}\frac{\partial^2 u}{\partial F^2} + \frac{1}{2}\nu^2\alpha_t^2\frac{\partial^2 u}{\partial \alpha^2} + \rho\nu\alpha_t^2 F_t^{\beta}\frac{\partial^2 u}{\partial F \partial \alpha}\right] dt \\
& + \alpha_t F_t^{\beta}\frac{\partial u}{\partial F}\, dW_t + \nu\alpha_t\frac{\partial u}{\partial \alpha}\, dZ_t
\end{aligned}
$$

Deep BSDE - SABR

Now, recall that $u(t, F, \alpha)$ satisfies the SABR PDE:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\alpha^2 F^{2\beta}\frac{\partial^2 u}{\partial F^2} + \rho\nu\alpha^2 F^{\beta}\frac{\partial^2 u}{\partial F\,\partial\alpha} + \frac{1}{2}\nu^2\alpha^2\frac{\partial^2 u}{\partial\alpha^2} = 0$$

Therefore, the entire drift term (the dt-part) cancels out.
We are left with only the stochastic (martingale) part:

$$du = \alpha_t F_t^{\beta}\,\frac{\partial u}{\partial F}\,dW_t + \nu\alpha_t\,\frac{\partial u}{\partial \alpha}\,dZ_t$$

Discretize time: $0 = t_0 < t_1 < \cdots < t_N = T$
Euler–Maruyama forward simulation:

$$\begin{cases} F_{t_{n+1}} = F_{t_n} + \alpha_{t_n} F_{t_n}^{\beta}\,\Delta W_n \\ \alpha_{t_{n+1}} = \alpha_{t_n} + \nu\alpha_{t_n}\,\Delta Z_n \end{cases}$$

Deep BSDE - SABR

Discretized BSDE:

$$u(t_{n+1}, F_{t_{n+1}}, \alpha_{t_{n+1}}) \approx u(t_n, F_{t_n}, \alpha_{t_n}) \\ + \alpha_{t_n} F_{t_n}^{\beta}\, u_F(t_n, F_{t_n}, \alpha_{t_n})\, \Delta W_n + \nu \alpha_{t_n}\, u_\alpha(t_n, F_{t_n}, \alpha_{t_n})\, \Delta Z_n$$

We approximate:

- $Y_{t_n} \approx u(t_n, F_{t_n}, \alpha_{t_n})$
- $\delta_F(t_n) \approx u_F(t_n, F_{t_n}, \alpha_{t_n})$ is predicted by a neural network
- $\delta_\alpha(t_n) \approx u_\alpha(t_n, F_{t_n}, \alpha_{t_n})$ is predicted by a neural network

At each step t_n, a neural network takes $(t_n, F_{t_n}, \alpha_{t_n})$ as input and predicts δ_F and δ_α.

Terminal Condition and Loss

At $t = T$, we match the terminal payoff:

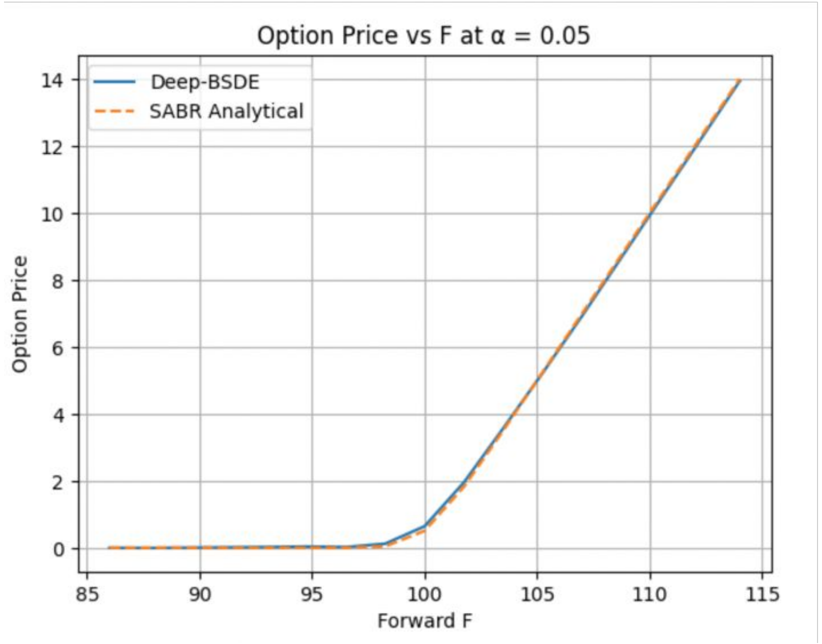
$$Y_T \approx u(T, F_T, \alpha_T) \approx (F_T - K)^+$$

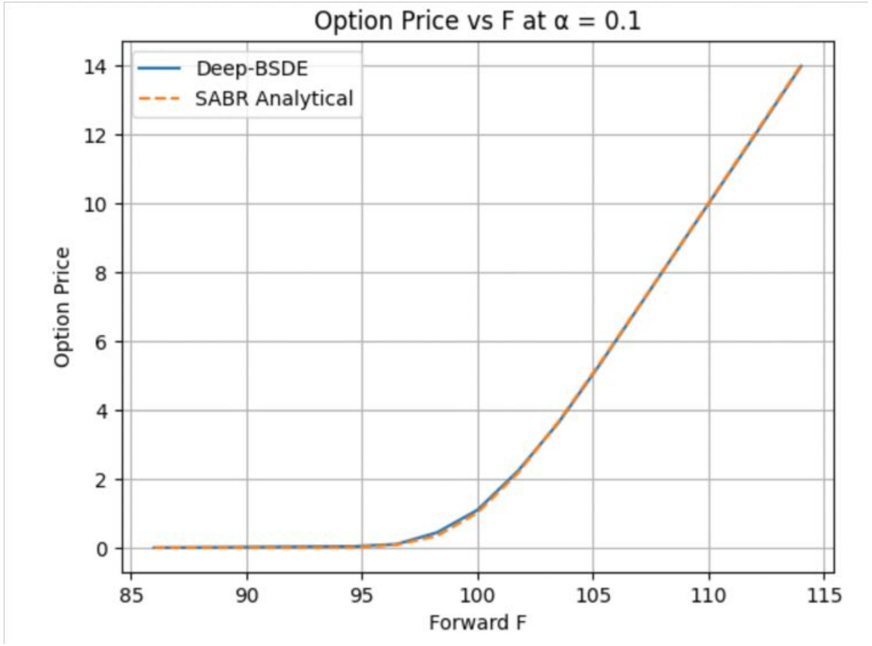
Loss function:

$$\mathcal{L} = \mathbb{E}\left[\left(Y_T - (F_T - K)^+\right)^2\right]$$

The Deep BSDE method learns to approximate gradients u_F, u_α, and recursively predicts $u(t, F, \alpha)$ backward from maturity using Monte Carlo paths and neural networks.

Effect of α on SABR Pricing: Deep BSDE vs Analytical





- Deep BSDE captures the pricing surface under different initial volatility α and forward price F.
- When α increases (from 0.05 to 0.1), the curvature around ATM increases.

Deep Galerkin Method (DGM) - Black Scholes

- Consider the Black-Scholes PDE:

$$\frac{\partial u}{\partial t}(t,S) + rS\frac{\partial u}{\partial S}(t,S) + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2}(t,S) = ru(t,S)$$

- We have the terminal condition as payoff:

$$u(T,S) = (S-K)^+$$

- With boundary condition when K>>S and K<<S:

$$\begin{cases} u(t,S) \approx S - Ke^{-r(T-t)}, & \text{as } S \to \infty \\ u(t,S) \approx 0, & \text{as } S \to 0 \end{cases}$$

Deep Galerkin Method (DGM) - Black Scholes

- Given the condition for BS PDE, we could construct the loss function with 3 components as follow:
- PDE Loss:

$$\mathcal{L}_{\mathrm{PDE}} = \mathbb{E}_{(t,S)\sim\mathrm{interior}}\left[\left(u_t + rSu_S + \frac{1}{2}\sigma^2 S^2 u_{SS} - ru\right)^2\right]$$

- Terminal Loss:

$$\mathcal{L}_{\mathrm{Terminal}} = \mathbb{E}_{S\sim\mathrm{terminal}}\left[\left(u(T,S) - (S-K)^+\right)^2\right]$$

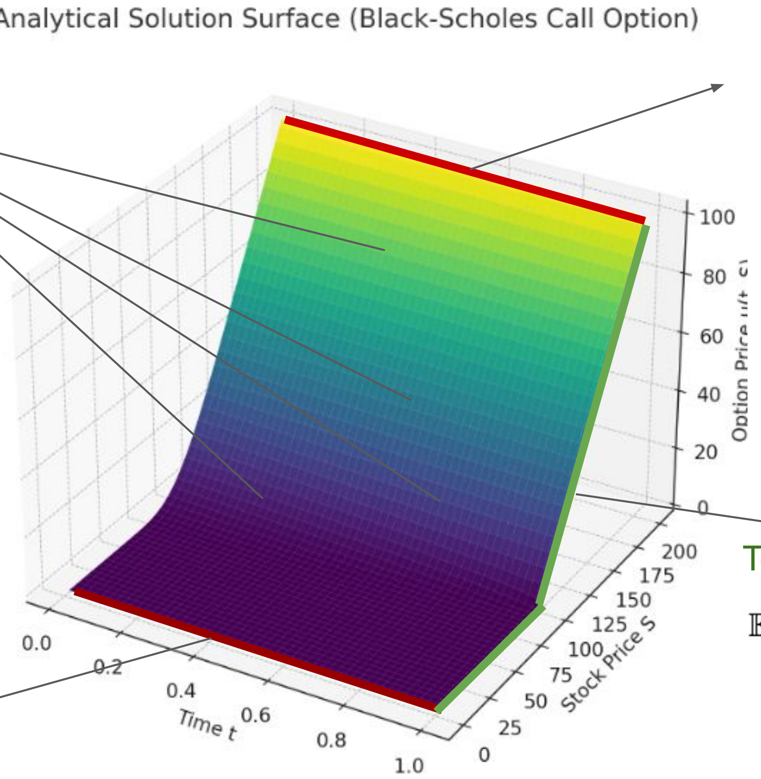
- Boundary Loss:

$$\mathcal{L}_{\mathrm{Boundary}} = \mathbb{E}\left[\left(u(t,S=0)^2\right) + \left(u(t,S=S_{\max}) - (S_{\max} - Ke^{-r(T-t)})\right)^2\right]$$

Deep Galerkin Method (DGM) - Black Scholes

PDE Loss:
controls the whole inner surface part of the solution surface

$$\mathcal{L}_{\text{PDE}} = \mathbb{E}_{(t,S)\sim\text{interior}}\left[\left(u_t + rSu_S + \frac{1}{2}\sigma^2 S^2 u_{SS} - ru\right)^2\right]$$



Boundary Condition: when S >> K, u(t, S) -> S - Ke^r(T-t)

$$\lambda_{\text{bdry}} \cdot \mathbb{E}_{(t,x)\sim\mathcal{D}_{\text{bdry}}}\left[\left|u_\theta(t,x) - b(t,x)\right|^2\right]$$

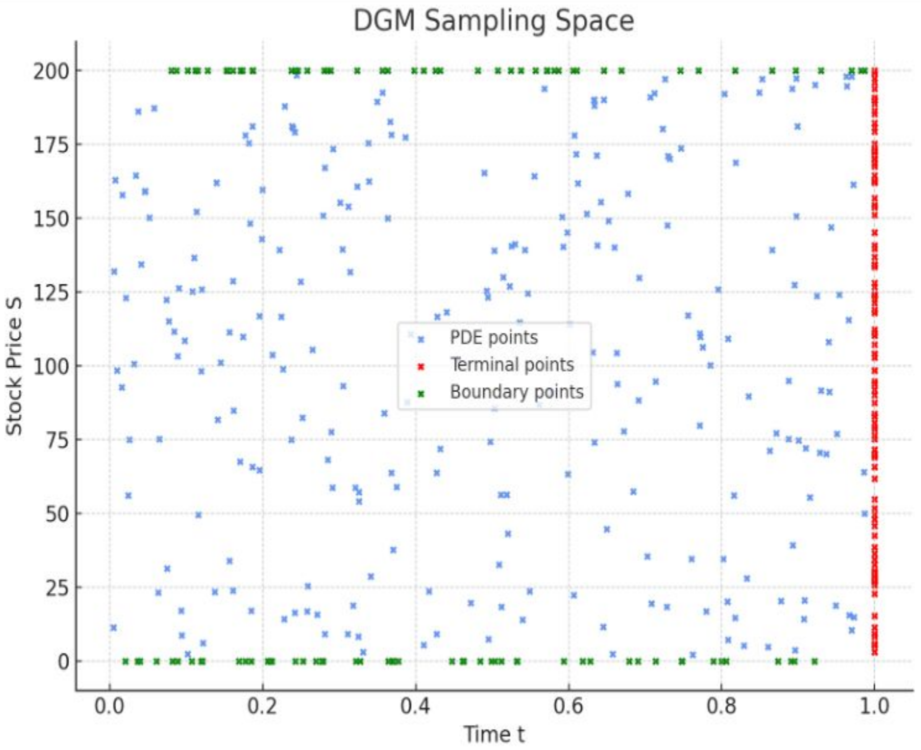
Terminal Condition: Payoff

$$\mathbb{E}_{S\sim\text{terminal}}\left[\left(u(T,S) - (S-K)^+\right)^2\right]$$

Boundary Condition:
when S << K, u(t, S) -> 0

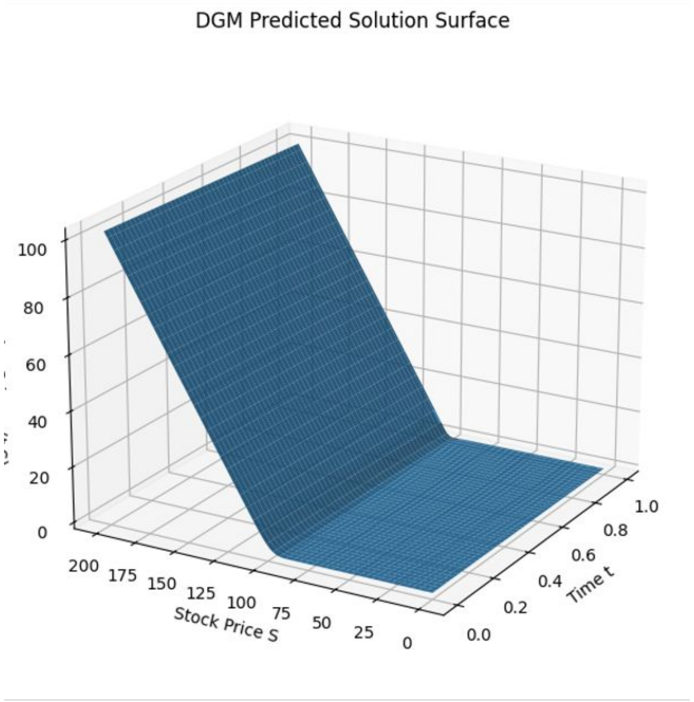
$$\lambda_{\text{bdry}} \cdot \mathbb{E}_{(t,x)\sim\mathcal{D}_{\text{bdry}}}\left[\left|u_\theta(t,x) - b(t,x)\right|^2\right]$$

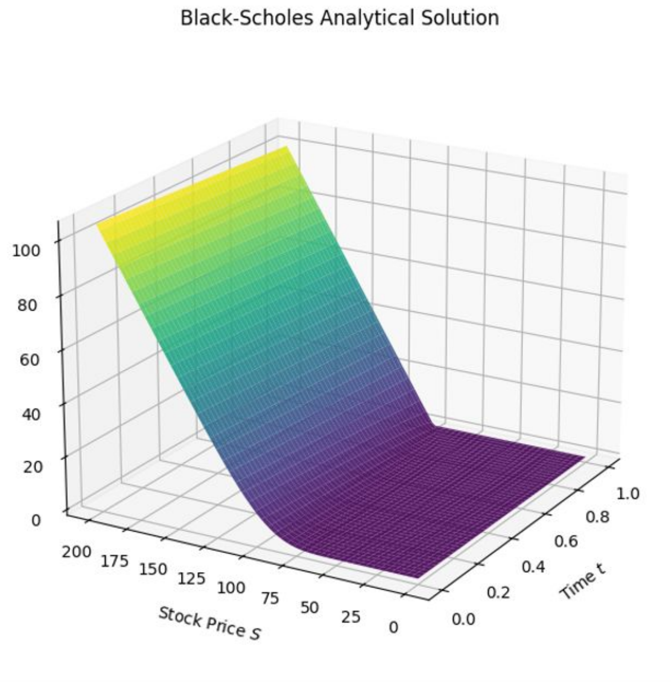
DGM – Mesh-Free Sampling Space



- **PDE points (blue):** uniformly sampled inside the domain to minimize the PDE residual.
- **Terminal points (red):** enforce the terminal condition at t = T
- **Boundary points (green):** enforce boundary conditions u(t, S_min)→0 and u(t,S_max)
- This approach allows solving high-dimensional PDEs without **requiring grid discretization.**

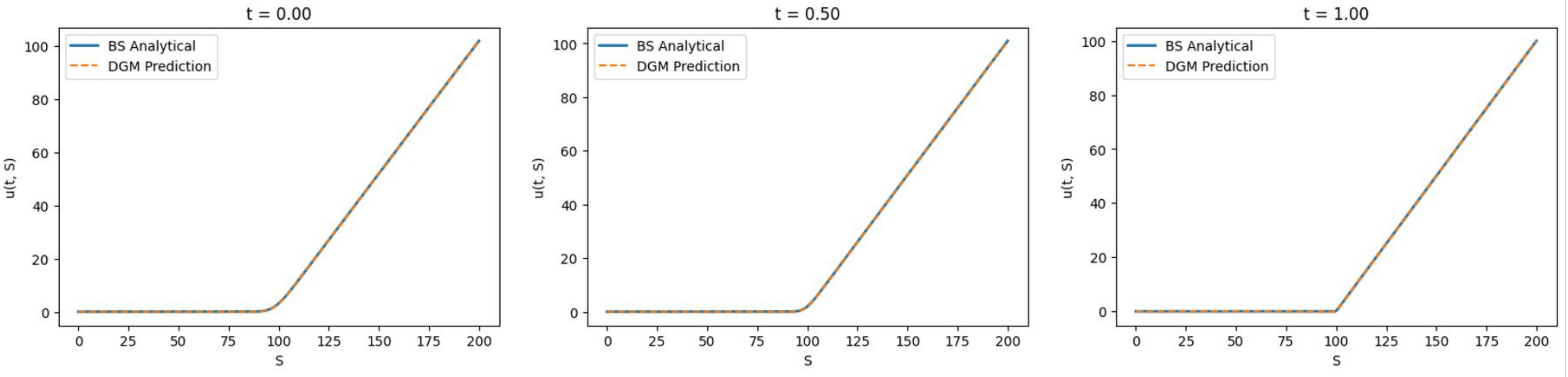
Result & Benchmark





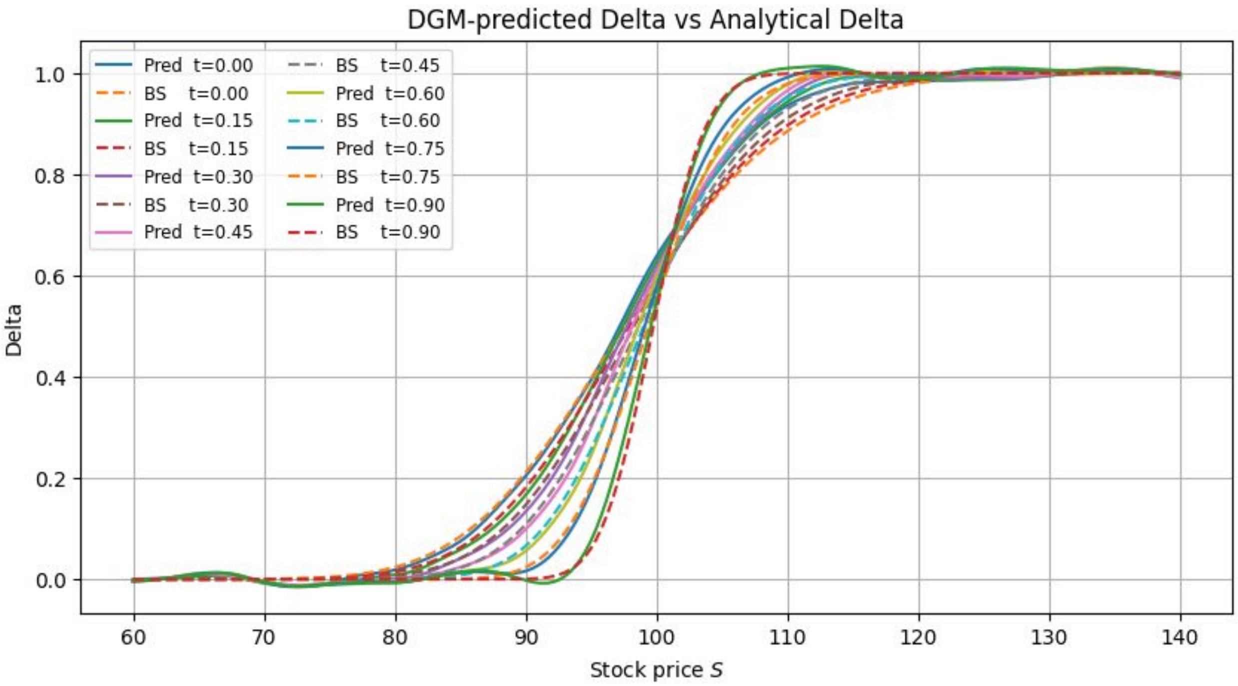
- DGM successfully approximates the solution surface of the Black-Scholes PDE.
- Neural networks provide a smooth and differentiable function, usable for pricing and Greeks extraction.

Result & Benchmark



- DGM predictions are compared with Black-Scholes analytical solutions at multiple time steps.
- Across all snapshots, the predicted curves match the analytical ones almost perfectly.
- This indicates that the DGM model learns the solution accurately not just in space S, but also consistently over time.

Result & Benchmark



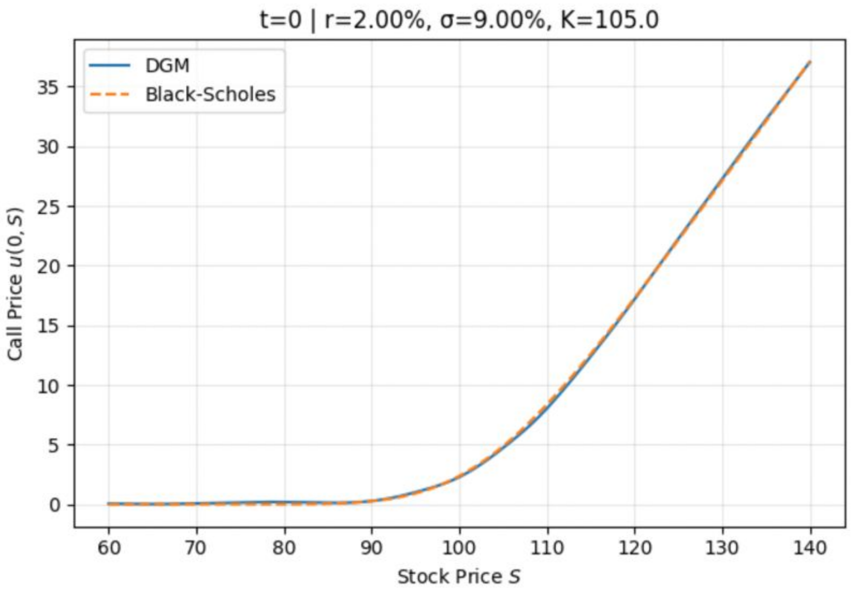
- DGM provides option price as a continuous and differentiable function to extract Delta
- DGM model not only fits the option price surface, but also captures its gradient — hedging ratio across time.

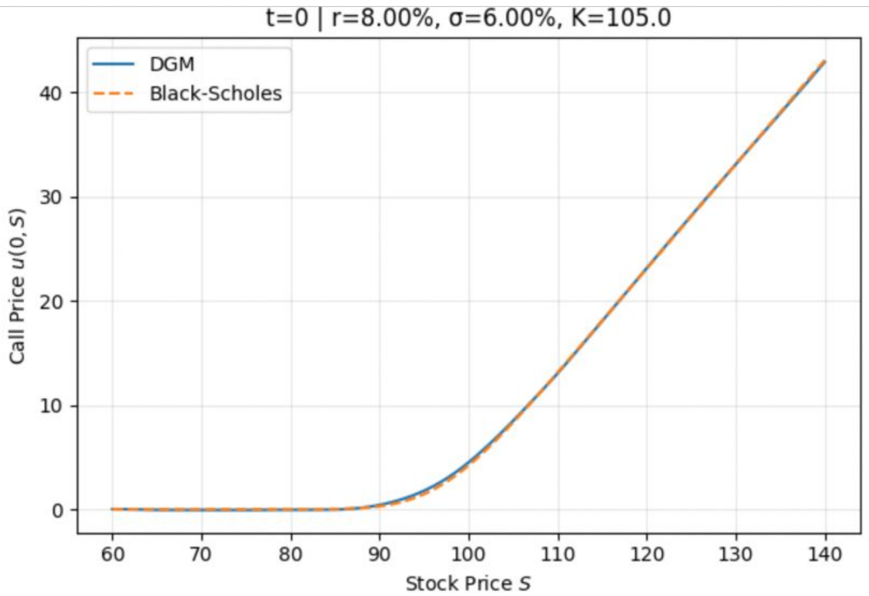
Generalized DGM - BS

- **Current Issue –** In practice, option pricing parameters like r, K, σ often vary across products or market conditions. Traditional DGM models require retraining when any parameter changes, which is computationally expensive.

- **Solution -** Treat r, K, σ as the the input features for DGM learn a family of solution u(t, S, K, r, σ). This enables amortized inference — one model works for many settings, saving time during deployment.

Result – Benchmark





- Generalized DGM accurately predicts option prices under varying parameters.
- One model, many markets — achieving amortized inference and saving retraining time.

DGM - SABR

- Consider the follow SABR PDE:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\alpha^2 F^{2\beta}\frac{\partial^2 u}{\partial F^2} + \rho\nu\alpha^2 F^{\beta}\frac{\partial^2 u}{\partial F\,\partial\alpha} + \frac{1}{2}\nu^2\alpha^2\frac{\partial^2 u}{\partial\alpha^2} = 0$$

- With terminal condition:

$$u(T, F, \alpha) = (F - K)^+$$

- And boundary condition:

$$\begin{cases} u(t, F, \alpha) \approx F - K, & \text{as } F \to \infty \\ u(t, F, \alpha) \approx 0, & \text{as } F \to 0 \end{cases}$$

DGM - SABR

- PDE Loss:

$$\mathrm{Loss}_{\mathrm{PDE}} = \mathbb{E}_{(t,F,\alpha)\sim\mathrm{interior}}\left[\left(u_t + \frac{1}{2}\alpha^2 F^{2\beta} u_{FF} + \rho\nu\alpha^2 F^{\beta} u_{F\alpha} + \frac{1}{2}\nu^2\alpha^2 u_{\alpha\alpha}\right)^2\right]$$

- Terminal Loss:

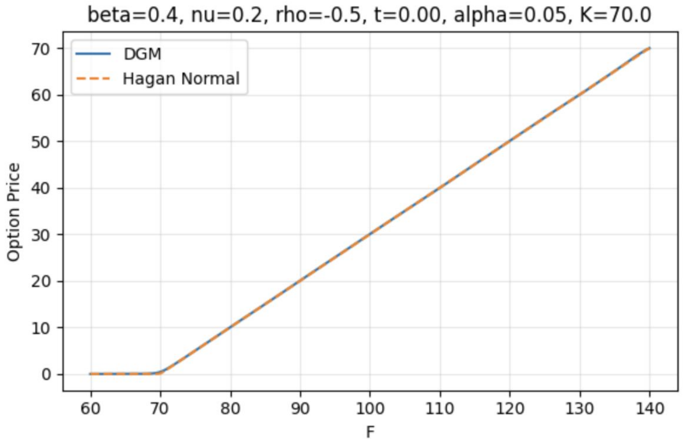
$$\mathrm{Loss}_{\mathrm{Terminal}} = \mathbb{E}_{F\sim\mathrm{terminal}}\left[\left(u(T,F,\alpha) - (F-K)^+\right)^2\right]$$

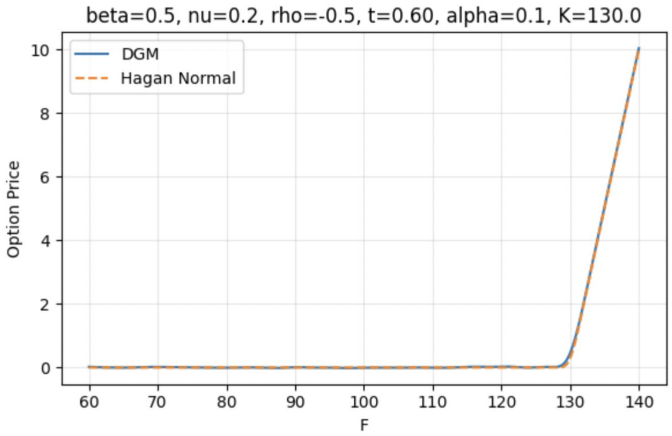
- Boundary Loss:

$$\mathrm{Loss}_{\mathrm{Boundary}} = \mathbb{E}\left[u(t,F=0,\alpha)^2\right] + \mathbb{E}\left[\left(u(t,F=F_{\max},\alpha) - (F_{\max}-K)^+\right)^2\right]$$

Generalized DGM - SABR

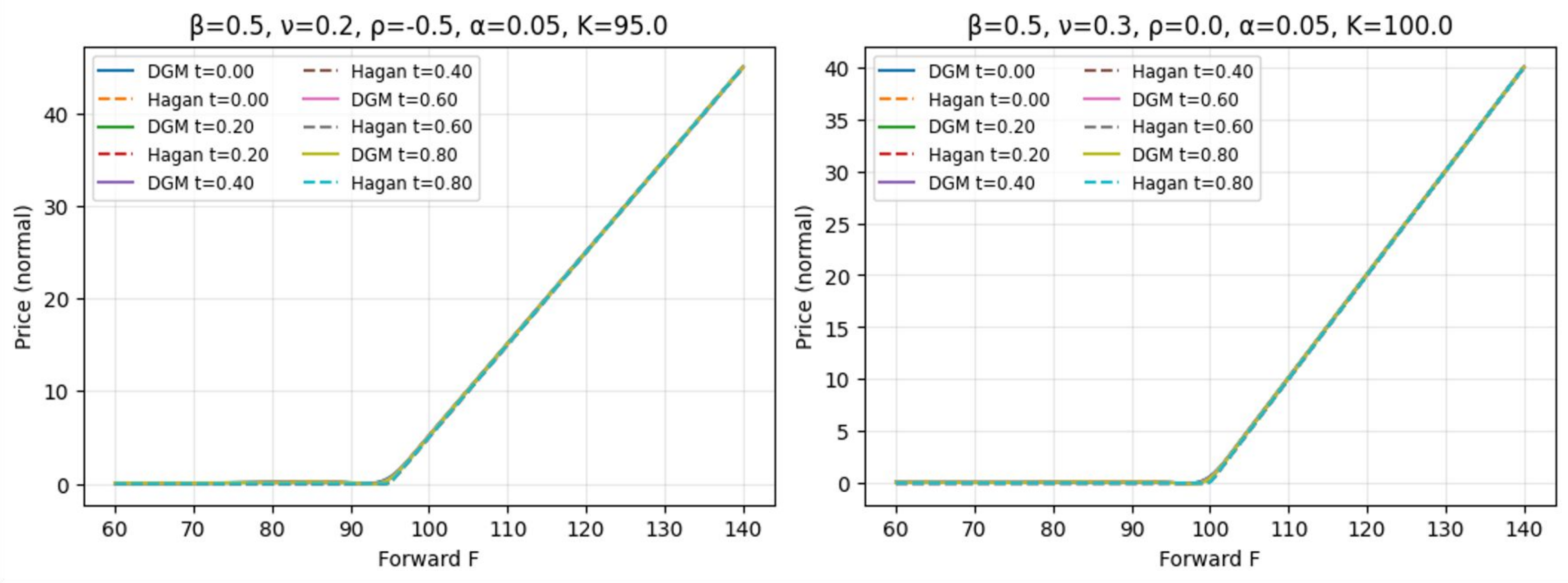
- Similar to Black-Scholes, we can fitted neural network to learn a 7D solution space that maps the all the input parameters, t,F,α,β,v,ρ, K, to the solution of SABR model uθ (t,F,α,β,v,ρ, K) ≈ SABR(t,F,α,β,v,ρ, K)





- DGM accurately matches Hagan's SABR Normal pricing under different market conditions.
- Left: linear payoff structure (ATM), Right: sharp curvature (deep OTM).
- DGM can handle varying SABR dynamics and nonlinear payoffs robustly.

Result & Benchmark



- DGM matches Hagan Normal approximation across time
- One model handles time + parameter variation without retraining

DGM - Fokker Planck Core Trick

- Consider the forward PDE with delta function as the initial condition:

$$\frac{\partial p}{\partial t} = \cdots, \quad p(0,y) = \delta(y-x)$$

- **Delta function is hard** so we train the neural network to approximate the Cumulative Density Function (CDF) from backward Kolmogorov equation of the forward PDE as follow:

$$C(t,x;T,y) = \mathbb{P}(X_T \leq y \mid X_t = x) = \int_{-\infty}^{y} p(t,x;T,z)dz$$

- With terminal condition:

$$C(T,x;T,y) = 1_{x \leq y}$$

- Get the **Transition Probability Densities Function (TPDF)** by taking the derivatives:

$$p(t,x;T,y) = \frac{\partial C(t,x;T,y)}{\partial y}$$

DGM Fokker Planck - Geometric Brownian Motion (BS)

- Learning Objective:

$$u(t, S, \sigma, r, S_T) = \mathbb{P}(S_T \leq S_T^{(\text{target})} \mid S_t = S)$$

- Consider the Backward Kolmogorov Equation:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2} + rS\frac{\partial u}{\partial S} = 0$$

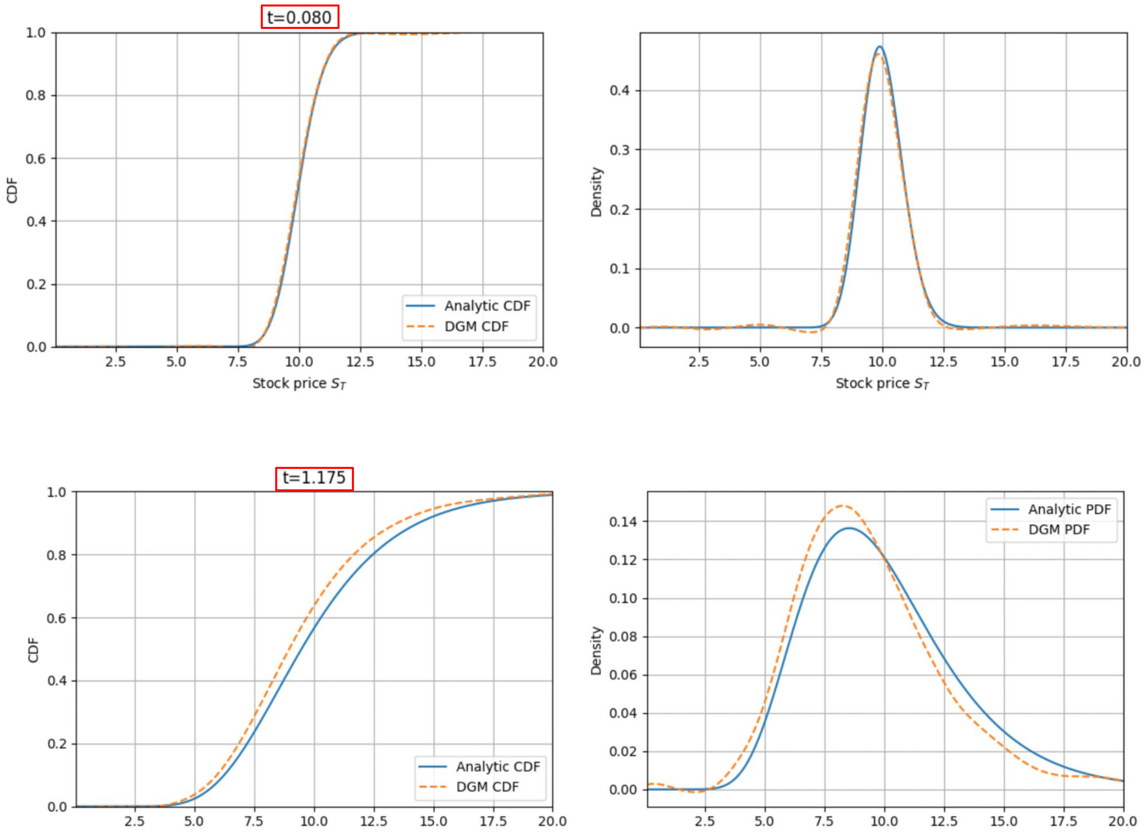
- Terminal Condition:

$$u(T, S, \sigma, r, S_T) = 1_{S \leq S_T}$$

- Get the **TPDF** by taking the derivatives w.r.t to S:

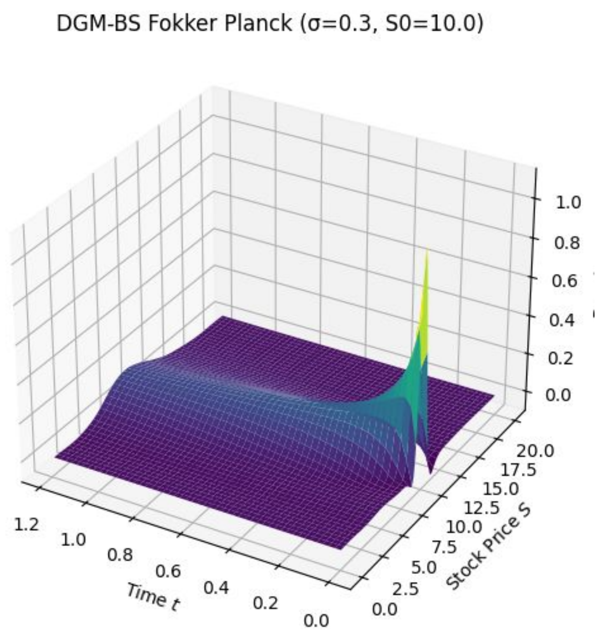
$$p(t, S; \sigma, r, S_T) = \frac{\partial u}{\partial S_T}$$

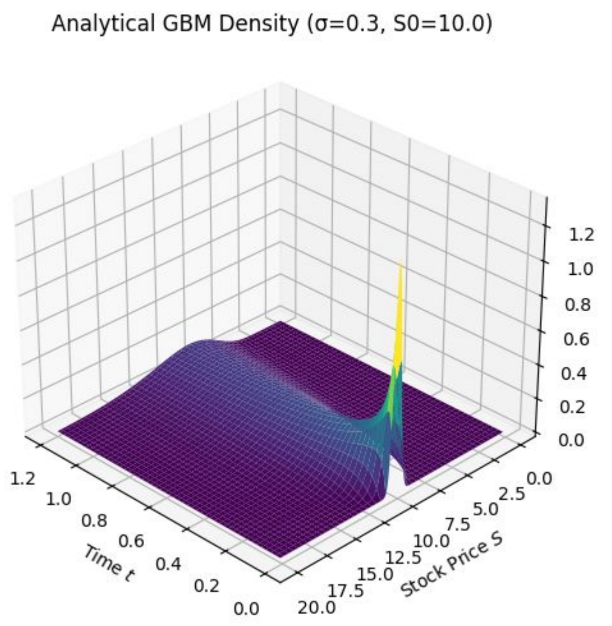
Benchmark: DGM Recovers Accurate Distributions Across Time



- Learns CDF directly; obtains PDF via differentiation
- Matches analytical solutions at both t = 0.080 and t = 1.175
- Robust across time: handles sharp peaks and smoothed tails
- Captures full market uncertainty beyond simple pricing

DGM Recovers the Joint Density p(t, S) Over Time





- DGM solution (left) vs. Analytical GBM (right)
- Accurately captures the full evolution of $p(t, S)$
- Matches surface shape, peak dynamics, and smoothness

DGM Fokker Planck - SABR

- Learning Objective:

$$u(t, F, \alpha, \rho, \beta, \nu, F_T, \alpha_T) = \mathbb{P}(F_T \leq F_T^{(\text{target})},\, \alpha_T \leq \alpha_T^{(\text{target})} \mid F_t = F,\, \alpha_t = \alpha)$$

- Consider the Backward Kolmogorov Equation for SABR:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\alpha^2 F^{2\beta} \frac{\partial^2 u}{\partial F^2} + \frac{1}{2}\nu^2\alpha^2 \frac{\partial^2 u}{\partial \alpha^2} + \rho\nu\alpha^2 F^{\beta} \frac{\partial^2 u}{\partial F \partial \alpha} = 0$$

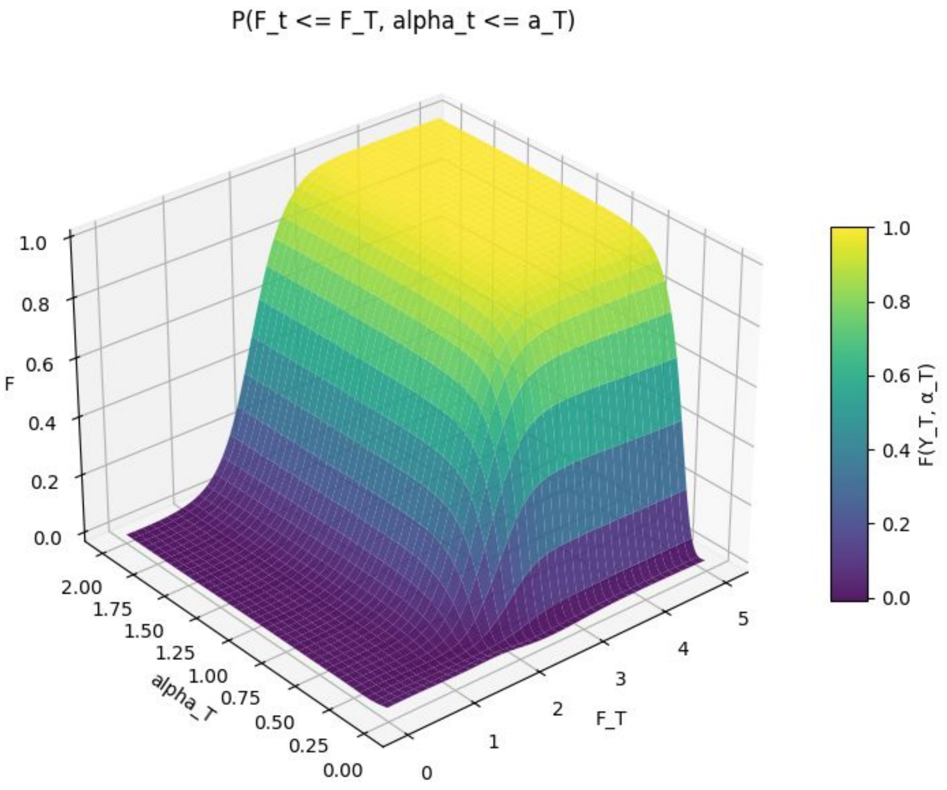
- Terminal Condition:

$$u(T, F, \alpha, \rho, \beta, \nu, F_T, \alpha_T) = 1_{F_T \geq F} \cdot 1_{\alpha_T \geq \alpha}$$

- Get the TPDF by taking the derivatives w.r.t. F and a:

$$p(t, F, \alpha; F_T, \alpha_T, \rho, \beta, \nu) = \frac{\partial^2 u}{\partial F_T \, \partial \alpha_T}$$

Learned CDF of Joint Distribution (F, α)

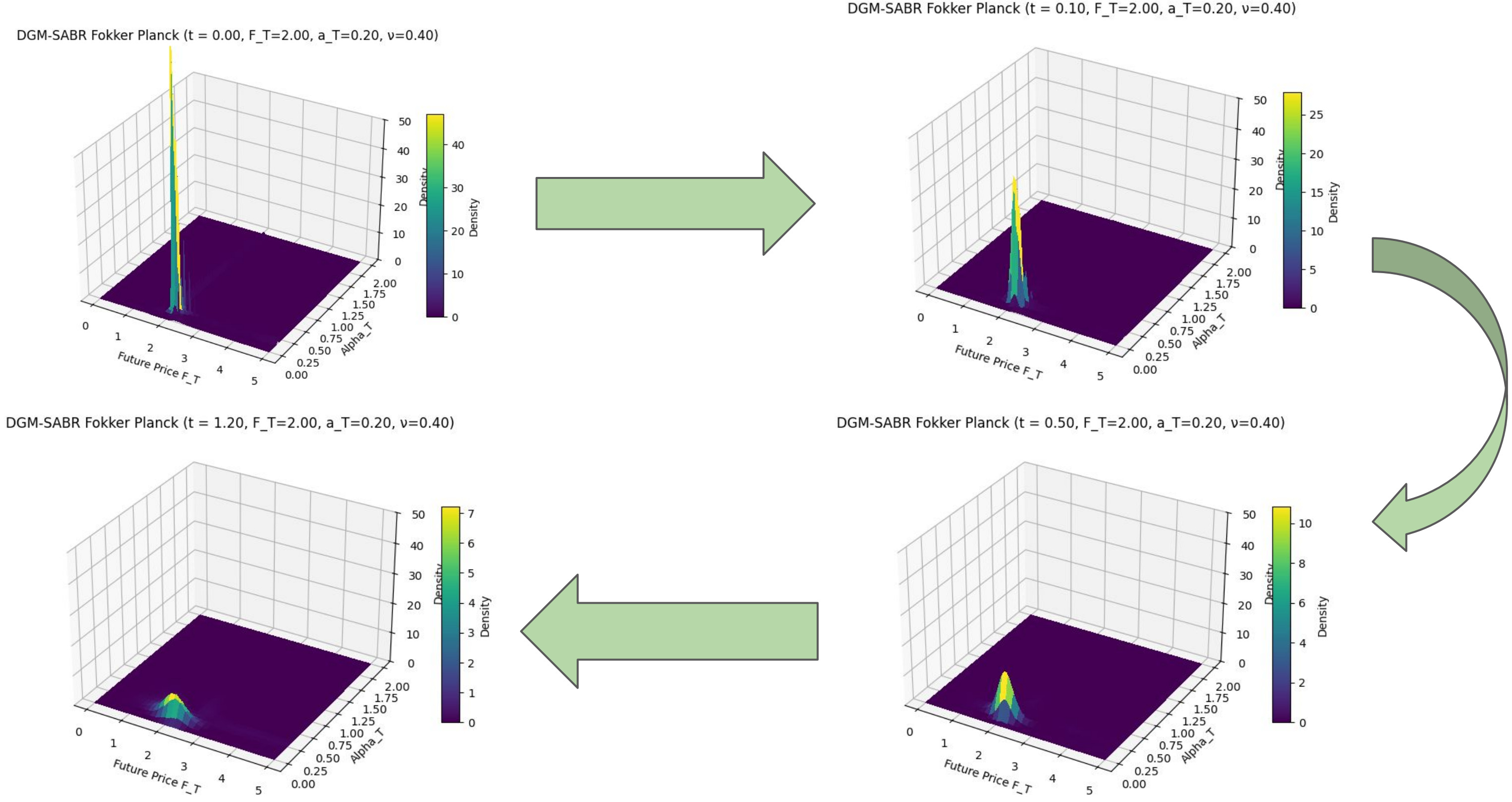


- Learned joint CDF:

$$\mathbb{P}(F_T \le f,\, \alpha_T \le a)$$

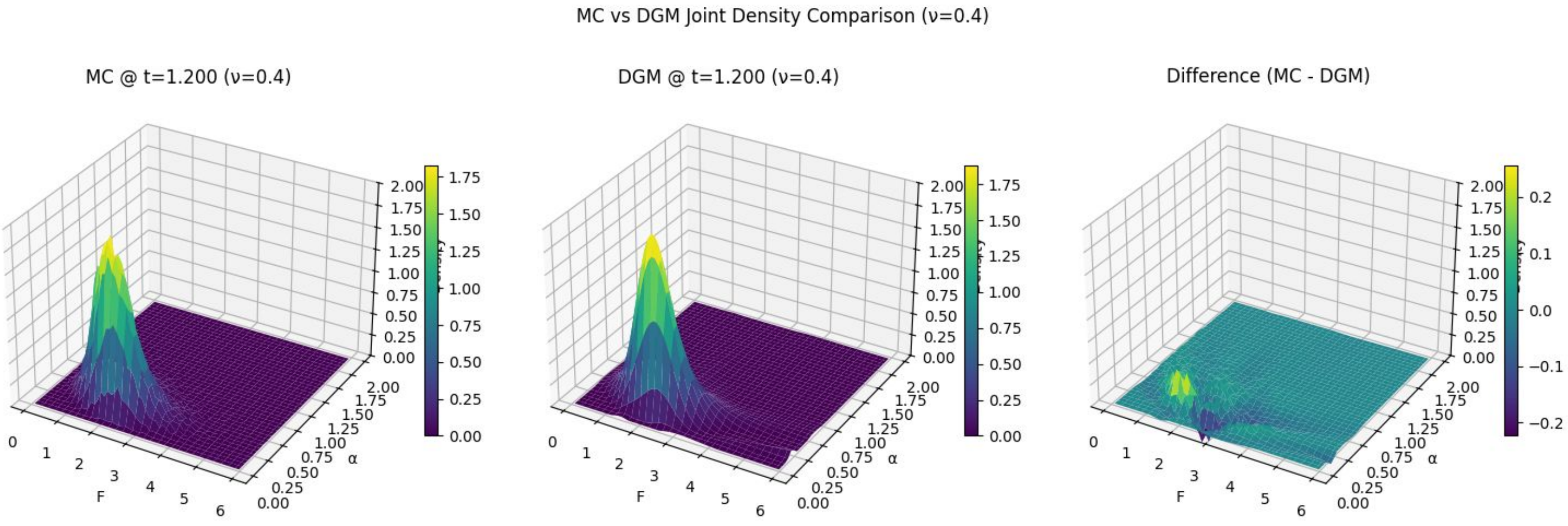
- Smoothly captures probability mass over 2D domain
- Derivatives yield joint density via auto-differentiation

Joint PDF by Auto-Differentiation



- SABR’s joint PDF evolution across time
- The density gradually spreads, skews, and stabilizes.

Result & Benchmark



MC simulated joint density

DGM-predicted density

Difference (MC - DGM)

Matched closely in both shape and location of the probability mass