Supplementary Notes

Yicheng Zhuang Yukun Wang
Yanru Guan Hongyan Xu Yuluyang Xie

August 2, 2024

Contents

1 Mathematical Derivation

1.1 Option Pricing and Risk-Neutral Density Relationship

Let the current time be t, the expiration time be T, the current asset price be S_t, the strike price be K, and the risk-free rate be r. The price C of a European call option can be expressed as:

$$C(S_t, K, \tau) = e^{-r\tau}\mathbb{E}^Q[(S_T - K)^+|S_t]$$

where $\tau = T - t$ is the time to maturity and $\mathbb{E}^Q$ denotes the expectation under the risk-neutral measure Q.

To express the option price in terms of the risk-neutral density $f_{S_T}(s)$, we first define the conditional probability density function $f_{S_T}(s|S_t)$, which represents the density of S_T given S_t at time t. According to the law of total probability, the relationship between the conditional density $f_{S_T}(s|S_t)$ and the marginal density $f_{S_T}(s)$ is[1]:

$$f_{S_T}(s|S_t) = \frac{f_{S_T}(s)}{f_{S_t}(S_t)}$$

However, we can directly use the marginal density $f_{S_T}(s)$ for risk-neutral option pricing.

According to the definition of the option price, we have:

$$C(S_t, K, \tau) = e^{-r\tau}\int_{-\infty}^{\infty}(S_T - K)^+ f_{S_T}(s)\,ds$$

To represent the relationship between the risk-neutral density and the option price more intuitively, we introduce the concept of log-returns X_τ:

$$X_\tau = \ln\left(\frac{S_T}{S_t}\right)$$

The corresponding density function $q_{X_\tau}(x)$ is related to $f_{S_T}(s)$ by:

$$q_{X_\tau}(x) = S_t e^x f_{S_T}(S_t e^x)$$

Thus, the option prices can be re-expressed as:

$$C(S_t, K, \tau) = e^{-r\tau}S_t\int_{-\infty}^{\infty}(e^x - \frac{K}{S_t})^+ q_{X_\tau}(x)\,dx$$

Similarly, for the put option, we have:

$$P(S_t, K, \tau) = e^{-r\tau}S_t\int_{-\infty}^{\infty}(\frac{K}{S_t} - e^x)^+ q_{X_\tau}(x)\,dx$$

We aim to estimate $q_{X_\tau}(x)$ from a set of observed prices of calls $\{C_i\}_{i=1}^{N_C}$ and puts $\{P_j\}_{j=1}^{N_P}$ by minimizing[2]

$$\frac{1}{N_C}\sum_{i=1}^{N_C} w_i^C L(C_i, \hat{C}_i) + \frac{1}{N_P}\sum_{j=1}^{N_P} w_j^P L(P_j, \hat{P}_j),$$

where $L(\cdot,\cdot)$ is the loss function, for example, the absolute or relative mean square error (MSE); $\hat{C}_i$ and $\hat{P}_j$ are the option prices given by the model; w_i^C and w_j^P are the regularized weights that may be related to the option liquidity levels and we set $w_i^C = w_j^P = 1.0$ for simplicity.

No-arbitrage conditions ensure the rationality and consistency of the option market, preventing the existence of risk-free arbitrage opportunities[3]. We summarize six main no-arbitrage conditions as follows:

1.2 No-Arbitrage Conditions

No-arbitrage conditions ensure the rationality and consistency of the option market, preventing the existence of risk-free arbitrage opportunities. We summarize six main no-arbitrage conditions as follows:

$$\begin{aligned}
\frac{\partial C}{\partial K} \leq 0 \quad &\text{and} \quad \frac{\partial P}{\partial K} \geq 0 \\
\frac{\partial^2 C}{\partial K^2} \geq 0 \quad &\text{and} \quad \frac{\partial^2 P}{\partial K^2} \geq 0 \\
\lim_{K\to\infty} C(S_t, K, \tau) = C(S_t, \infty, \tau) = 0 \quad &\text{and} \quad \lim_{K\to 0} P(S_t, K, \tau) = P(S_t, 0, \tau) = 0 \\
C(S_t, K, 0) = (S_t - K)^+ \quad &\text{and} \quad P(S_t, K, 0) = (K - S_t)^+ \quad \text{when} \quad \tau = 0 \\
\frac{\partial C}{\partial \tau} \geq 0 \quad &\text{and} \quad \frac{\partial P}{\partial \tau} \geq 0 \\
(S_t - Ke^{-r\tau})^+ \leq C(S_t, K, \tau) \leq S_t \quad &\text{and} \quad (Ke^{-r\tau} - S_t)^+ \leq P(S_t, K, \tau) \leq Ke^{-r\tau}
\end{aligned}$$

1.2.1 Condition 1: Negative Partial Derivative of Option Price with Respect to Strike Price

For European call options:

$$\frac{\partial C(S_t, K, \tau)}{\partial K} = -e^{-r\tau} \int_{\ln(K/S_t)}^{+\infty} q_{X_\tau}(x)\, dx$$

To ensure that $\frac{\partial C}{\partial K \leq 0}$, the integral against $q_{X_\tau}(x)$ on the right-hand side must be uniformly non-negative, given the negative sign in front of the right-hand side and the fact that the discount factor $e^{-r\tau}$ is positive[4]. Since $q_{X_\tau}(x)$ is a density function and thus point-wise non-negative, the integral against it is indeed non-negative uniformly. Therefore,

$$\frac{\partial C}{\partial K} \leq 0 \quad \text{and} \quad \frac{\partial P}{\partial K} \geq 0$$

always hold in our setup.

1.2.2 Condition 2: Non-Negative Second Partial Derivative of Option Price with Respect to Strike Price

The second condition is to ensure the pricing function is free from butterfly spread arbitrage among option strikes for any fixed option maturity[5]. We have:

$$\frac{\partial^2 C(S_t, K, \tau)}{\partial K^2} = \frac{\partial}{\partial K}\left\{-e^{-r\tau} \int_{\ln(K/S_t)}^{+\infty} q_{X_\tau}(x)\, dx\right\} = \frac{1}{K} e^{-r\tau} q_{X_\tau}\left(\ln\left(\frac{K}{S_t}\right)\right), \quad \forall K > 0,\, \tau \geq 0.$$

Since $q_{X_\tau}(x)$ is constructed as a density function which is point-wise non-negative, $\frac{\partial^2 C}{\partial K^2}$ shall be uniformly non-negative. This is also the case for puts. Therefore,

$$\frac{\partial^2 C}{\partial K^2} \geq 0 \quad \text{and} \quad \frac{\partial^2 P}{\partial K^2} \geq 0$$

always hold.

1.2.3 Condition 3: Boundary Conditions for Option Prices at Extreme Strike Prices

The third condition imposes the boundary prices to be zero when the strike price of call options is set to infinity and the strike price of put options is set to zero. It is interpreted as that a put (or call) option has no value if its strike is zero (or infinitely high) because real-world stock prices cannot fall below zero or rise to infinitely high. More specifically,

$$\lim_{K\to\infty} C(S_t, K, \tau) = 0 \quad \text{and} \quad \lim_{K\to 0} P(S_t, K, \tau) = 0.$$

This constraint is satisfied in our setup.

1.2.4 Condition 4: Boundary Conditions for Option Prices at Zero Maturity

The fourth condition sets price conditions on the zero time-to-maturity boundary. It is equivalent to requiring the random variable X_τ is degenerate, i.e., it almost surely takes a specific value, X_0, as the option' s maturity is infinitely close to the current calendar time. In other words, its density $q_{X_\tau}(x)$ becomes a Dirac delta function:

$$\lim_{\tau\to 0} q_{X_\tau}(x) = \delta(x - X_0), \quad \text{where} \quad X_0 := \lim_{t\to T} \ln\left(\frac{S_T}{S_t}\right) = 0.$$

Therefore, option prices on the zero time-to-maturity boundary are equal to their payoffs:

$$C(S_t, K, 0) = \lim_{\tau\to 0} e^{-r\tau} S_t \int_{\mathbb{R}} \left(e^x - S_t^{-1}K\right)^+ \delta(x)dx = (S_t - K)^+$$

$$P(S_t, K, 0) = \lim_{\tau\to 0} e^{-r\tau} S_t \int_{\mathbb{R}} \left(S_t^{-1}K - e^x\right)^+ \delta(x)dx = (K - S_t)^+.$$

1.2.5 Condition 5: Monotonicity of Option Prices with Respect to Maturity

The fifth condition is equivalent to the absence of calendar spread arbitrage between adjacent maturities for any fixed strike. Thus,

$$\frac{\partial C}{\partial \tau} \geq 0 \quad \text{and} \quad \frac{\partial P}{\partial \tau} \geq 0.$$

1.2.6 Condition 6: Upper and Lower Bounds of Option Prices

The sixth condition imposes upper and lower bounds on the prices of European options. Together with the non-negativity of option prices, these pricing bounds are direct consequences of assuming the following equality (holds uniformly for any time-to-maturity):

$$\ln \mathbb{E}^Q[e^{X_\tau}] = r\tau, \quad \forall \tau > 0.$$

It is equivalent to requiring that the present value of the stock price at any future time, as a conditional expectation under the spot risk-neutral measure, equals the current stock price:

$$\mathbb{E}_t^Q[S_T] = S_t e^{r(T-t)}, \quad \forall T > t.$$

The argument is as follows. First, if $\mathbb{E}_t^Q[S_T] = S_t e^{r(T-t)}$ holds for all $T > t$, then the put-call parity holds:

$$\begin{aligned} C(S_t, K, \tau) - P(S_t, K, \tau) &= e^{-r\tau} \int_0^{+\infty} \left[(s-K)^+ - (K-s)^+\right] f_{S_T}(s) ds \\ &= e^{-r\tau} \int_0^{+\infty} s f_{S_T}(s) ds - e^{-r\tau} \int_0^{+\infty} K f_{S_T}(s) ds \\ &= e^{-r\tau} \mathbb{E}_t^Q[S_T] - K e^{-r\tau} \\ &= S_t - K e^{-r\tau}, \quad \forall \tau > 0,\, K > 0 \end{aligned}$$

The lower bound of calls is obtained by applying the put-call parity and using the fact that option prices are non-negative:

$$C(S_t, K, \tau) = P(S_t, K, \tau) + S_t - K e^{-r\tau} \geq S_t - K e^{-r\tau} \quad \textit{and} \quad C(S_t, K, \tau) \geq 0.$$

Together, we have calls bounded below as:

$$C(S_t, K, \tau) \geq (S_t - K e^{-r\tau})^+, \quad \forall \tau > 0,\, K > 0.$$

The upper bound of calls is the consequence of putting Constraint 1, i.e., $\frac{\partial C}{\partial K} \leq 0$, together with the equality we assume to hold:

$$C(S_t, K, \tau) \leq S_t.$$

Thus, calls are bounded as:

$$(S_t - K e^{-r\tau})^+ \leq C(S_t, K, \tau) \leq S_t.$$

For put options, the argument is similar. From the upper bound of calls, we know $C(S_t, K, \tau) - S_t \leq 0$. Therefore, puts are bounded above by $K e^{-r\tau}$ from the put-call parity because:

$$P(S_t, K, \tau) = K e^{-r\tau} + C(S_t, K, \tau) - S_t \leq K e^{-r\tau}.$$

By the non-negativity of option prices, both $P(S_t, K, \tau) \geq 0$ and $P(S_t, K, \tau) = K e^{-r\tau} + C(S_t, K, \tau) - S_t \geq K e^{-r\tau} - S_t$ should hold. Together, we have puts bounded below by $(K e^{-r\tau} - S_t)^+$.

Rechecking the equality $\mathbb{E}_t^Q[S_T] = S_t e^{r(T-t)}$, or equivalently, $C(S_t, 0, \tau) = S_t$, we argue that it should hold in a rational market because a call option with a zero strike will always be exercised, resulting in a payoff S_T. This is equivalent to possessing the underlying asset directly.

1.3 Model Specification

The log-returns $X_\tau = \ln\left(\frac{S_T}{S_t}\right)$ are modeled as:

$$\begin{aligned} X_\tau := X(Z, \tau) = \mu(\tau) + \sigma(\tau) \cdot Z \cdot G(Z, \tau), \quad Z \sim N(0,1), \tau \in \mathbb{R}^+ \\ X_0 := \lim_{t \to T} \ln\left(\frac{S_T}{S_t}\right) = 0 \end{aligned}$$

where $\mu(\cdot)$, $\sigma(\cdot)$, and $G(\cdot, \cdot)$ are deterministic functions, either in closed forms or parameterized by neural nets with Softplus activation $\ln(1 + \exp(x))$.

1.3.1 Single Maturities

For single maturity:

$$\mu(\tau) := \mu, \quad \sigma(\tau) := \sigma, \quad G(Z,\tau) := G(Z)$$
$$G(Z) = uZ^A + v(-Z)^A + 1$$
$$X = \mu + \sigma Z(uZ^A + v(-Z)^A + 1)$$

For calibration, the model prices are:

$$\hat{C}(S_t, K, \tau) = e^{-r\tau} S_t \cdot \frac{1}{N} \sum_{n=1}^{N} \left(e^{X(Z_n,\tau)} - \frac{K}{S_t} \right)^+$$
$$\hat{P}(S_t, K, \tau) = e^{-r\tau} S_t \cdot \frac{1}{N} \sum_{n=1}^{N} \left(\frac{K}{S_t} - e^{X(Z_n,\tau)} \right)^+$$

The model must satisfy:

$$\mu + \ln \left[\frac{1}{N} \sum_{n=1}^{N} \exp\left(\sigma Z_n \left(uZ_n^A + v(-Z_n)^A + 1\right)\right) \right] = r\tau$$

1.3.2 Multiple Maturities

For multiple maturities, we define:

$$\mu(\tau) := r \cdot \tau \cdot G_\mu(\tau; \theta_\mu),$$
$$\sigma(\tau) := \sigma\sqrt{\tau},$$
$$G(Z,\tau) := G_Z(Z; \theta_Z) + G_\tau(\tau; \theta_\tau) + 1,$$

where G_μ, G_Z, and G_τ are MLP neural networks with Softplus activation, parameterized by θ_μ, θ_Z, θ_τ. The log-return stochastic curve is:

$$X_\tau := X(Z,\tau;\theta_M) = r\tau G_\mu(\tau;\theta_\mu) + \sigma\sqrt{\tau} Z[G_Z(Z;\theta_Z) + G_\tau(\tau;\theta_\tau) + 1], \quad \forall \tau > 0,$$
$$X_0 := X(Z,\tau;\theta_M)\Big|_{\tau=0} = 0.$$

Model prices are given by:

$$\hat{C}_M = S_t e^{-r\tau} \frac{1}{N} \sum_{n=1}^{N} \left(e^{X(Z_n,\tau;\theta_M)} - \frac{K_C}{S_t} \right)^+,$$
$$\hat{P}_M = S_t e^{-r\tau} \frac{1}{N} \sum_{n=1}^{N} \left(\frac{K_P}{S_t} - e^{X(Z_n,\tau;\theta_M)} \right)^+,$$

with $\theta_M = \{\theta_\mu, \theta_Z, \theta_\tau, \sigma\}$.
The model must satisfy:

$$\ln \mathbb{E}\left[\exp\left(r\tau G_\mu(\tau;\theta_\mu) + \sigma\sqrt{\tau} Z[G_Z(Z;\theta_Z) + G_\tau(\tau;\theta_\tau) + 1]\right)\right] = r\tau, \quad \forall \tau > 0.$$

Proposition 1

Let the log-return curve X_τ follow the specification given by:

$$X_\tau := X(Z,\tau;\theta_M) = r\tau G_\mu(\tau;\theta_\mu) + \sigma\sqrt{\tau}Z[G_Z(Z;\theta_Z) + G_\tau(\tau;\theta_\tau) + 1], \quad \forall \tau > 0,$$
$$X_0 := X(Z,\tau;\theta_M)\Big|_{\tau=0} = 0.$$

The associated option prices $\hat{C}_M$ and $\hat{P}_M$ are free from static arbitrage if and only if:
(1) $J^C_\tau(\tau, K_C, \theta_M) \geq 0$ (2) $J^P_\tau(\tau, K_P, \theta_M) \geq 0$ (3) $J^M_\mu(\tau, \theta_M) = 0$
where

$$J^C_\tau(\tau, K_C, \theta_M) := \frac{1}{N}\sum_{n=1}^{N} I\left\{e^{X(Z_n,\tau;\theta_M)} \geq \frac{K_C}{S_t}\right\}\left[\left(\frac{\partial X(Z_n,\tau;\theta_M)}{\partial \tau} - r\right)e^{X(Z_n,\tau;\theta_M)} + r\frac{K_C}{S_t}\right],$$

$$J^P_\tau(\tau, K_P, \theta_M) := \frac{1}{N}\sum_{n=1}^{N} I\left\{e^{X(Z_n,\tau;\theta_M)} \leq \frac{K_P}{S_t}\right\}\left[\left(r - \frac{\partial X(Z_n,\tau;\theta_M)}{\partial \tau}\right)e^{X(Z_n,\tau;\theta_M)} - r\frac{K_P}{S_t}\right],$$

$$J^M_\mu(\tau, \theta_M) := \left|r\tau\left(G_\mu(\tau;\theta_\mu) - 1\right) + \ln\left(\frac{1}{N}\sum_{n=1}^{N}\exp\left(\sigma\sqrt{\tau}Z_n\left[G_Z(Z_n;\theta_Z) + G_\tau(\tau;\theta_\tau) + 1\right]\right)\right)\right|^2.$$

$$\min_{\theta^M} L_M = \frac{1}{N_C}\sum_{i=1}^{N_C} L(C_i, \hat{C}_i) + \frac{1}{N_P}\sum_{j=1}^{N_P} L(P_j, \hat{P}_j) + \lambda \cdot J(\theta^M),$$

where

$$J(\theta^M) = \sum_{\{\tau, K_C\}}\left(-J^C_\tau(\tau, K_C, \theta^M)\right)^+ + \sum_{\{\tau, K_P\}}\left(-J^P_\tau(\tau, K_P, \theta^M)\right)^+ + \sum_{\{\tau\}} J^M_\mu(\tau, \theta^M).$$

1.3.3 Additional Shape Flexibilities

To model log-returns with complex risk-neutral densities, we extend the RN-MLP to a mixture of two RN-MLPs:

$$X_\tau := X(Z,\tau;\theta^M_d) = \alpha \cdot X_1(Z,\tau;\theta^1_M) + (1-\alpha)\cdot X_2(Z,\tau;\theta^2_M), \quad \forall \tau > 0,$$
$$X_0 := X(Z,\tau;\theta^M_d)\Big|_{\tau=0} = 0,$$

where $\alpha \in \mathbb{R}$, $X_i, i = 1, 2$ share the same structure as RN-MLP:

$$X_i(Z,\tau;\theta^i_M) = r\tau G^i_\mu(\tau;\theta^i_\mu) + \sigma_i Q_i(Z,\tau;\theta^i_\sigma),$$
$$Q_i(Z,\tau;\theta^i_\sigma) = \sqrt{\tau}Z\left[G^i_Z(Z;\theta^i_Z) + G^i_\tau(\tau;\theta^i_\tau) + 1\right].$$

The set of parameters is $\theta^M_d = \{\alpha, \theta^1_M, \theta^2_M\}$, where $\theta^i_M = \{\sigma_i, \theta^i_\mu, \theta^i_\sigma\}$ and $\theta^i_\sigma = \{\theta^i_Z, \theta^i_\tau\}$ for $i = 1, 2$.
The RN-DMLP model's option prices are:

$$\hat{C}_{dM} = S_t e^{-r\tau}\frac{1}{N}\sum_{n=1}^{N}\left(e^{X(Z_n,\tau;\theta^M_d)} - \frac{K_C}{S_t}\right)^+,$$

$$\hat{P}_{dM} = S_t e^{-r\tau}\frac{1}{N}\sum_{n=1}^{N}\left(\frac{K_P}{S_t} - e^{X(Z_n,\tau;\theta^M_d)}\right)^+.$$

Proposition 2

Let the log-return curve X_τ follow the specification given by:

$$X_\tau := X(Z,\tau;\theta_d^M) = \alpha \cdot X_1(Z,\tau;\theta_M^1) + (1-\alpha)\cdot X_2(Z,\tau;\theta_M^2), \quad \forall \tau > 0,$$
$$X_0 := X(Z,\tau;\theta_d^M)\Big|_{\tau=0} = 0.$$

The associated option prices $\hat{C}_{dM}$ and $\hat{P}_{dM}$ are free from static arbitrage if and only if:
(1) $J_\tau^C(\tau, K_C, \theta_d^M) \geq 0$ (2) $J_\tau^P(\tau, K_P, \theta_d^M) \geq 0$ (3) $J_\mu^d M(\tau, \theta_d^M) = 0$
where

$$J_\tau^C(\tau,K_C,\theta_d^M) := \frac{1}{N}\sum_{n=1}^{N} I\left\{e^{X(Z_n,\tau;\theta_d^M)} \geq \frac{K_C}{S_t}\right\}\left[\left(\frac{\partial X(Z_n,\tau;\theta_d^M)}{\partial \tau} - r\right)e^{X(Z_n,\tau;\theta_d^M)} + r\frac{K_C}{S_t}\right],$$

$$J_\tau^P(\tau,K_P,\theta_d^M) := \frac{1}{N}\sum_{n=1}^{N} I\left\{e^{X(Z_n,\tau;\theta_d^M)} \leq \frac{K_P}{S_t}\right\}\left[\left(r - \frac{\partial X(Z_n,\tau;\theta_d^M)}{\partial \tau}\right)e^{X(Z_n,\tau;\theta_d^M)} - r\frac{K_P}{S_t}\right],$$

$$J_\mu^d M(\tau,\theta_d^M) := \left|r\tau\left(\sum_{i=1}^{2}\alpha_i G_\mu^i(\tau;\theta_\mu^i) - 1\right) + \ln\left(\frac{1}{N}\sum_{n=1}^{N}\exp\left[\sum_{i=1}^{2}\alpha_i\sigma_i Q_i(Z_n,\tau;\theta_\sigma^i)\right]\right)\right|^2.$$

$$\min_{\theta^{dM}} L_{dM} = \frac{1}{N_C}\sum_{i=1}^{N_C} L(C_i,\hat{C}_i) + \frac{1}{N_P}\sum_{j=1}^{N_P} L(P_j,\hat{P}_j) + \lambda\cdot J(\theta^{dM}),$$

where

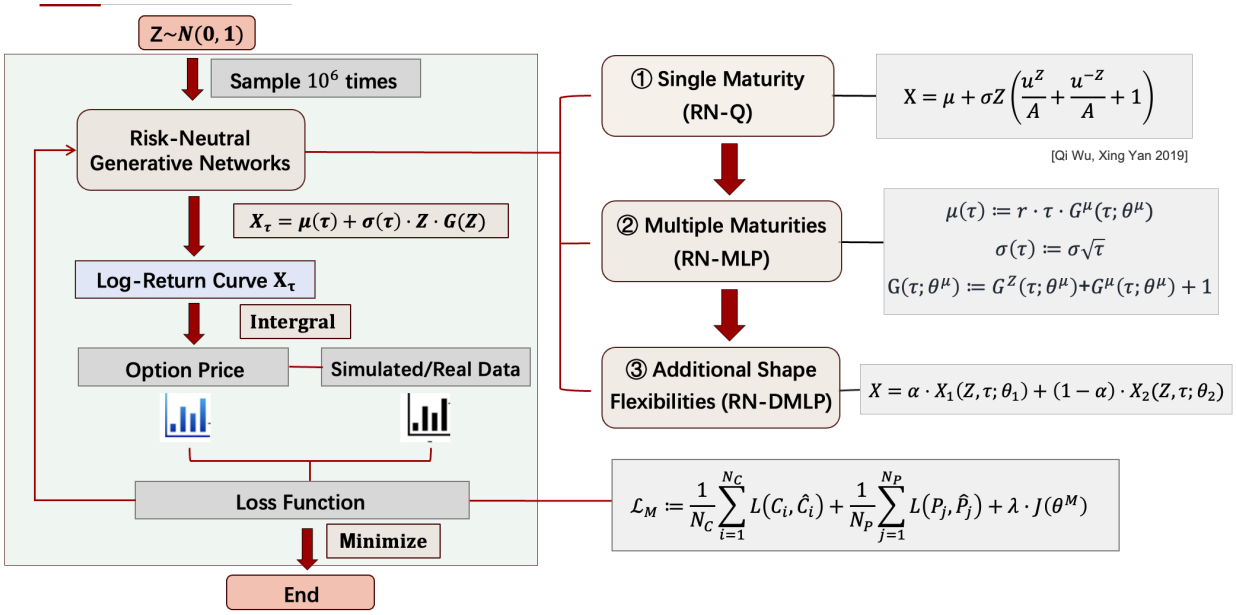
$$J(\theta^{dM}) = \sum_{\{\tau,K_C\}}\left(-J_\tau^C(\tau,K_C,\theta^{dM})\right)^+ + \sum_{\{\tau,K_P\}}\left(-J_\tau^P(\tau,K_P,\theta^{dM})\right)^+ + \sum_{\{\tau\}} J_\mu^{dM}(\tau,\theta^{dM}).$$

Comparison of the Three Models

Feature	RN-Q	RN-MLP	RN-DMLP
Usage Scenario	Single maturity calibration	Multi-maturity calibration	Multi-maturity calibration with additional shape complexities
Core Formula	$X = \mu + \sigma Z \cdot (uZ^A + v(-Z)^A + 1)$	$X_\tau := X(Z,\tau;\theta_M) = r\tau G_\mu(\tau;\theta_\mu) + \sigma\sqrt{\tau}Z [G_Z(Z;\theta_Z) + G_\tau(\tau;\theta_\tau) + 1]$	$X_\tau := X(Z,\tau;\theta_d^M) = \alpha\cdot X_1(Z,\tau;\theta_M^1) + (1-\alpha)\cdot X_2(Z,\tau;\theta_M^2)$
Parameters	μ,σ,u,v	$\theta_\mu,\theta_Z,\theta_\tau,\sigma$	$\alpha,\theta_M^1,\theta_M^2$
Flexibility	Low	Medium	High
Complexity	Low	Medium	High
Uses Neural Networks	No	Yes	Yes
Adaptability to Data	Low	Medium	High
Static No-Arbitrage Constraint	$\mu + \ln\left[\frac{1}{N}\sum_{n=1}^{N}\exp\left(\sigma Z_n (uZ_n^A + v(-Z_n)^A + 1)\right)\right] = r\tau$	$\ln\mathbb{E}\left[\exp\left(r\tau G_\mu(\tau;\theta_\mu) + \sigma\sqrt{\tau}Z [G_Z(Z;\theta_Z) + G_\tau(\tau;\theta_\tau) + 1]\right)\right] = r\tau$	$J_\mu^d M(\tau,\theta_d^M) := \left\|r\tau\left(\sum_{i=1}^{2}\alpha_i G_\mu^i(\tau;\theta_\mu^i) - 1\right) + \ln\left(\frac{1}{N}\sum_{n=1}^{N}\exp\left[\sum_{i=1}^{2}\alpha_i\sigma_i Q_i(Z_n,\tau;\theta_\sigma^i)\right]\right)\right\|^2$
Main Advantage	Simple and easy to use	Adapts to multiple maturities, enhanced adaptability with neural networks	Provides higher flexibility, capable of capturing more complex distribution shapes

2 Model

2.1 Procedure overview



Procedure overview

This flowchart illustrates the comprehensive process of using risk-neutral neural network models for option pricing. The process begins with sampling random variables Z from a standard normal distribution $N(0, 1)$, generating a large number of samples (e.g., 10^6 times) to simulate market uncertainty. These samples are fed into the risk-neutral generative networks, which are designed to model the log-return curve X_τ based on different maturity settings: single maturity (RN-Q), multiple maturities (RN-MLP), and additional shape flexibilities (RN-DMLP).

For the single maturity model (RN-Q), the log-return X is modeled as:

$$X = \mu + \sigma Z\left(\frac{u^Z + u^{-Z} + A}{A}\right).$$

For the multiple maturities model (RN-MLP), the parameters are:

$$\mu(\tau) := r \cdot \tau \cdot G_\mu(\tau; \theta_\mu), \sigma(\tau) := \sigma\sqrt{\tau}, G(Z, \tau) := G_Z(Z; \theta_Z) + G_\tau(\tau; \theta_\tau) + 1.$$

For the model with additional shape flexibilities (RN-DMLP), the log-return X is given by:

$$X = \alpha \cdot X_1(Z, \tau; \theta_1) + (1 - \alpha) \cdot X_2(Z, \tau; \theta_2).$$

The risk-neutral generative networks use these models to generate the log-return curve X_τ. This curve is then integrated to compute the option price. The computed option prices are compared against simulated or real market data to evaluate the model's performance.

A loss function is defined to measure the difference between the predicted option prices and the actual market data. The loss function used is:

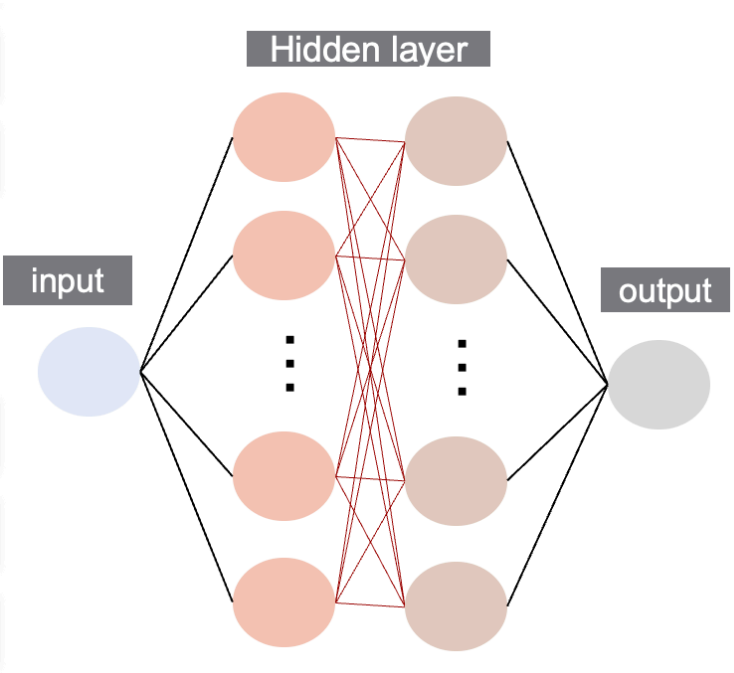
$$L_M := \frac{1}{N_C}\sum_{i=1}^{N_C} L(C_i, \hat{C}_i) + \frac{1}{N_P}\sum_{j=1}^{N_P} L(P_j, \hat{P}_j) + \lambda \cdot J(\theta^M),$$

where $L(C_i, \hat{C}_i)$ and $L(P_j, \hat{P}_j)$ represent the loss for call and put options respectively, and $J(\theta^M)$ is a regularization term.

The model parameters are optimized by minimizing the loss function, which involves adjusting the network parameters to ensure that the predicted option prices are as close as possible to the actual market prices. This optimization process iteratively improves the model's accuracy and stability in predicting option prices.

In summary, this flowchart demonstrates the integration of risk-neutral neural network models for generating log-return curves, computing option prices, and optimizing model parameters to achieve accurate and stable option pricing predictions based on market data. The use of various model settings (single maturity, multiple maturities, and additional shape flexibilities) allows for flexibility in capturing different market dynamics and ensuring robust performance.

2.2 Neural network



Neural network

Based on the detailed information provided in the article, the architecture and hyperparameter settings for the risk-neutral neural network models are as follows:

For the RN-Q model, the initial parameters used during training include $\sigma = 0.2$, $u = 1.1$, $v = 1.1$, and $A = 4.0$. These values are set to maintain consistency with the original paper's settings. The RN-MLP and RN-DMLP models share a common multi-layer perceptron (MLP) architecture. This architecture consists of two hidden layers, each comprising 32 neurons. The activation function employed in all hidden layers is the Softplus function, defined as $\ln(1 + e^x)$, which provides smooth and non-linear activation. The learning rate for training the models is fixed at 0.01. Throughout the article, these configurations are maintained as the default settings without further fine-tuning unless explicitly specified otherwise. This unified hyperparameter setting aims to avoid the need for dedicated tuning procedures and ensures a consistent network structure across different model variations.

3 Relevant Code

3.1 Heston RNMoments

The code in this part segment provides practical tools and visual insights into studying how the Heston model behaves under different parameter settings.

Firstly, we defined and implemented a simulation function for the Heston model to generate simulated data with specified parameters. This ensured we could produce trajectories of asset prices and variances under controlled conditions.

We then conducted multiple simulations using different sets of Heston model parameters to observe how these affect the asset prices and variance trajectories. For each simulation set, we calculated the required statistics (RNM2, RNM3, RNM4) to evaluate the model's performance under varying conditions.

Through simulations and statistical calculations, we assessed the effectiveness of the Heston model across different levels of correlation and volatility. Utilizing Python's scientific computing libraries like NumPy and SciPy, alongside plotting libraries such as Matplotlib, streamlined the implementation and visualization process.

Finally, the results is saved in PDF format, facilitating further analysis and reporting.

3.1.1 Simulation for Asset Prices and Variance

This Python function "heston model si" simulates the Heston model to generate trajectories of asset prices and variance over time. It takes initial parameters such as asset price (S_0), initial variance (v_0), risk-free rate (r), correlation (ρ) between asset returns and variance, rate of mean reversion (κ), long-term mean of variance (θ), volatility of volatility (σ), simulation time (T), number of time steps (N), and number of simulations (M).

We firstly initializes time step (dt), mean vector (μ), and covariance matrix (cov) for the correlated Brownian motions.

Then arrays initialization is done by creating arrays S and v to store asset prices and variances over time, initialized with starting values S_0 and v_0.

Next is simulation loop where we iterative simulates asset prices and variances for each time step:

- Draws correlated Brownian motions Z using np.random.multivariate normal;
- Updates asset prices S[i] based on the Heston model formula incorporating drift (r - 0.5 * v[i-1]) and stochastic term (np.sqrt(v[i-1] * dt) * Z[i-1, :, 0]).
- Updates variance v[i] using the Heston model dynamics ensuring it remains non-negative.

Finally we make outputs return arrays S and v containing simulated asset prices and variances over time for all simulation scenarios.

Code Listing

Listing 1: Simulation

```
#=============================================================================

def heston_model_sim(S0, v0, r, rho, kappa, theta, sigma, T, N, M):
    """
    Simulates the Heston model to generate asset prices and
    variance trajectories.

    Inputs:
     - S0, v0: initial parameters for asset price and variance
     - r       : risk-free rate
     - rho     : correlation between asset returns and variance
     - kappa   : rate of mean reversion in variance process
     - theta   : long-term mean of variance process
     - sigma   : volatility of volatility (~ of variance process)
     - T       : time of simulation in years
     - N       : number of time steps
     - M       : number of simulations / scenarios

    Outputs:
    - S: asset prices over time (numpy array of shape (N+1, M))
    - v: variance over time (numpy array of shape (N+1, M))
    """
    # initialise other parameters
    dt = T/N
    mu = np.array([0, 0])
    cov = np.array([[1, rho],
                    [rho, 1]])

    # arrays for storing prices and variances
    S = np.full(shape=(N+1, M), fill_value=S0)
    v = np.full(shape=(N+1, M), fill_value=v0)

    # sampling correlated brownian motions under risk-neutral measure
    Z = np.random.multivariate_normal(mu, cov, (N, M))

    for i in range(1, N+1):
        # Asset price simulation using Heston model
        S[i] = S[i-1] * np.exp((r - 0.5 * v[i-1]) * dt + \
        np.sqrt(v[i-1] * dt) * Z[i-1, :, 0])
        # Variance simulation using Heston model
        v[i] = np.maximum(v[i-1] + kappa * (theta - v[i-1]) * dt + \
        sigma * np.sqrt(v[i-1] * dt) * Z[i-1, :, 1], 0)

    return S, v
```

This function enables the study of how different parameters influence asset price and variance dynamics under the Heston model, crucial for financial modeling and risk management.

3.1.2 Parameter setting

Code in this part generates asset price and variance trajectories under different conditions through simulations with multiple sets of parameter settings using the Heston model. First, the initial asset price, the length of the simulation, the risk-free rate, the number of time steps per year, and the number of scenarios simulated are set.

Then, three different sets of Heston model parameters were set: including the variance mean reversion rate, the long-run variance mean, and the initial variance, and simulations were conducted for each set of parameters.

Code Listing

Listing 2: Parameter setting

```
#=============================================================================

# Parameters for simulations
S0 = 100.0              # initial asset price
T = 1.0                 # time horizon in years
r = 0.02                # risk-free rate
N = 252                 # number of time steps in simulation per year
M = 100000              # number of simulations

# Different parameter settings for the Heston model
kappa = 0.15
# rate of mean reversion of variance under risk-neutral dynamics
theta = 0.25
# long-term mean of variance under risk-neutral dynamics
v0 = 0.05
# initial variance under risk-neutral dynamics

# First set of simulations
rho = -0.9
# correlation between asset returns, variances under r-n dynamics
sigma = 0.35
# volatility of volatility
S_l, v_l = heston_model_sim(S0,v0,r,rho,kappa,theta,sigma,T,N,M)

# Second set of simulations
rho = -0.2
sigma = 0.25
S_l, v_l = heston_model_sim(S0,v0,r,rho,kappa,theta,sigma,T,N,M)

# Third set of simulations
rho = 0.85
sigma = 0.2
S_l, v_l = heston_model_sim(S0,v0,r,rho,kappa,theta,sigma,T,N,M)

# Combine all simulation results
S_all = [S_l, S_n, S_r]
sets = ['a', 'b', 'c']
```

Notably, we evaluated the performance of the model under different market conditions by adjusting the values of correlation and volatility in each simulation. Finally, all simulation results were consolidated into a single list for further analysis and presentation.

3.1.3 Generating RNM2 Statistics

This section calculates the RNM2 statistics, which is the standard deviation of log returns, for different time steps and parameter settings, and saves the results in a PDF file.

Code Listing

Listing 3: RNM2 generating

```
#=============================================================
# RNM2 Statistics Generation

# Time steps for which statistics will be calculated
steps = np.array([5, 21, 63, 126, 189, 252])

# Generate and save RNM2_Heston.pdf
pdf = PdfPages('RNM2_Heston.pdf')
figsize = (10, 4)
plt.figure(figsize=figsize)
for i in range(len(S_all)):
    S = S_all[i]
    col = 'Parameter Setting ({})'.format(sets[i])
    # Calculate RNM2 statistics (standard deviation of log returns)
    line = [np.std(np.log(S[j, :]/S0)) for j in steps]
    plt.plot(['RNM2_1W', 'RNM2_1M', 'RNM2_3M', 'RNM2_6M',\
        'RNM2_9M', 'RNM2_1Y'], line, label=col, linestyle='--')
    plt.scatter(['RNM2_1W', 'RNM2_1M', 'RNM2_3M', 'RNM2_6M',\
        'RNM2_9M', 'RNM2_1Y'], line)
    plt.legend()
pdf.savefig(bbox_inches='tight', dpi=400)
plt.show()
pdf.close()
# End of RNM2 Statistics Generation
#=============================================================
```

3.1.4 Generating RNM3 Statistics

This section calculates the RNM3 statistics, which is the skewness of log returns, for different time steps and parameter settings, and saves the results in a PDF file.

Code Listing

Listing 4: RNM3 generating

```
#=============================================================
# RNM3 Statistics Generation

# Generate and save RNM3_Heston.pdf
pdf = PdfPages('RNM3_Heston.pdf')
plt.figure(figsize=figsize)
for i in range(len(S_all)):
    S = S_all[i]
    col = 'Parameter␣Setting␣({})'.format(sets[i])
    # Calculate RNM3 statistics (skewness of log returns)
    line = [stats.skew(np.log(S[j, :]/S0)) for j in steps]
    plt.plot(['RNM2_1W', 'RNM2_1M', 'RNM2_3M', 'RNM2_6M', \
        'RNM2_9M', 'RNM2_1Y'], line, label=col, linestyle='—')
    plt.scatter(['RNM2_1W', 'RNM2_1M', 'RNM2_3M', 'RNM2_6M',\
        'RNM2_9M', 'RNM2_1Y'], line)
    plt.legend()
pdf.savefig(bbox_inches='tight', dpi=400)
plt.show()
pdf.close()
# End of RNM3 Statistics Generation
#=============================================================
```

3.1.5 Generating RNM4 Statistics

This section calculates the RNM4 statistics, which is the kurtosis of log returns, for different time steps and parameter settings, and saves the results in a PDF file.

Code Listing

Listing 5: RNM4 generating

```
#=============================================================
# RNM4 Statistics Generation

# Generate and save RNM4_Heston.pdf
pdf = PdfPages('RNM4_Heston.pdf')
plt.figure(figsize=figsize)
for i in range(len(S_all)):
    S = S_all[i]
    col = 'Parameter Setting ({})'.format(sets[i])
    # Calculate RNM4 statistics (kurtosis of log returns)
    line = [stats.kurtosis(np.log(S[j, :]/S0)) for j in steps]
    plt.plot(['RNM2_1W', 'RNM2_1M', 'RNM2_3M', 'RNM2_6M',\
        'RNM2_9M', 'RNM2_1Y'], line, label=col, linestyle='--')
    plt.scatter(['RNM2_1W', 'RNM2_1M', 'RNM2_3M', 'RNM2_6M',\
        'RNM2_9M', 'RNM2_1Y'], line)
    plt.legend()
pdf.savefig(bbox_inches='tight', dpi=400)
plt.show()
pdf.close()
# End of RNM4 Statistics Generation
#=============================================================
```

In a word, this code provides practical tools and visualisation results for studying the behaviour of the Heston model under different parameter settings, through simulation and statistical volume analysis.

3.2 Tradingdays RNMoments

In order to extract the Risk Neutral Density (RND) by generating machine learning and plotting the related graphs using statistical data over different time periods, the following code has been designed.

Through parameter parsing and configuration, data reading and preprocessing, sorting intervals, processing the dataset, separating and combining data, handling missing values and finally plotting, the code completes the entire process from reading raw data to generating charts.

Getting these charts, we can better understand the changes in risk-neutral density over different time intervals.

3.2.1 Parameter Parsing and Configuration

Firstly, the code uses argparse to process command line arguments. Specifically, it takes a parameter from the command line that specifies the directory where the results will be saved. This parameter has a default value; if you don't provide your own path, it will use the default save path.

Code Listing

Listing 6: Parameter Parsing and Configuration

```
#=========================================================================
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"


parser = argparse.ArgumentParser(description="Risk_Neutral_Density␣\
␣␣␣␣Extraction_Using_Generative_Machine_Learning_RNDSurface_plot␣")
# configuration
parser.add_argument('--result_save_dir', type=str,\
    default='../../logs/EmpiricalStudy/MultipleTaus/result_dir/', \
    help='saved␣path␣for␣model')
args = parser.parse_args()
```

This code provides the basis for configuration and parameter passing for subsequent program execution by setting environment variables and parsing command line arguments. The purpose of this is to ensure that the program works properly in a particular runtime environment and to allow the user to customise the path where the results are saved via the command line.

3.2.2 Data Reading and Preprocessing

In the main() module, the code sets a variable, isTest, to True. it then overrides the previous default save path and uses a new one. Then, it reads a CSV file containing statistics for different dates.

Code Listing

Listing 7: Parameter Parsing and Configuration

```
#=========================================================================
if __name__ == '__main__':
    isTest = True #!!!

    args.result_save_dir = '../../logs/EmpiricalRNDSurface/OneTau/\
␣␣␣␣␣␣␣␣surfplot_main/20240330/'
    RnMomentsRaw = pd.read_csv('../../logs/EmpiricalRNDSurface/\
␣␣␣␣␣␣␣␣OneTau/surfplot_dir_search/Onetau_RNDSurf_MomentsALL.csv')
```

3.2.3 Sorting Intervals

Next, the code defines a function, cat days rule, which is used to sort the data into different time intervals based on the number of days (e.g., one week, one month, three months, etc.). This function checks the input for the number of days and returns the corresponding time interval label.

Code Listing

Listing 8: Parameter Parsing and Configuration

```
#=============================================================================
    def cat_days_rule(days):
        # if days >= 1 and days <= 7: # 7
        #     colname = '1W'
        # elif days >= 29 and days <= 31: # 30
        #     colname = '1M'
        # elif days >= 88 and days <= 92: # 90
        #     colname = '3M'
        # elif days >= 175 and days <= 185: # 180
        #     colname = '6M'
        # elif days >= 264 and days <= 276: # 270
        #     colname = '9M'
        # elif days >= 360 and days <= 370: # 365
        #     colname = '1Y'
        deltaD_minus = 1; deltaD_plus = 0
        # if days >= 2 and days <= 7: # 7
        if days >= 5 and days <= 7 + 1: # 7
            colname = '1W'
        elif days >= 30 - 5 and days <= 30 + 5: # 30
            colname = '1M'
        elif days >= 90 - 10 and days <= 90 + 10: # 90
            colname = '3M'
        elif days >= 180 - 15 and days <= 180 + 15: # 180
            colname = '6M'
        elif days >= 270 - 20 and days <= 270 + 20: # 270
            colname = '9M'
        elif days >= 365 - 31 and days <= 365 + 31: # 365
            colname = '1Y'
        else:
            colname = 'nan'
        return colname
```

3.2.4 Processing the dataset

The CSV file read by the code is further processed. First, it will sort each row in the dataset to a specific time interval based on the number of days. Then, the code filters out the data that does not belong to these time intervals. Next, it groups the data by date and time interval and calculates the average of the standard deviation, skewness, and kurtosis for those groups.

Code Listing

Listing 9: Parameter Parsing and Configuration

```
#=============================================================================
    moments_dataset = RnMomentsRaw
    moments_dataset['DayInterval'] = moments_dataset['days'].\
        apply(lambda x:cat_days_rule(int(x)))
    moments_maturity = moments_dataset[(moments_dataset\
        ['DayInterval'] == '1W') |
        (moments_dataset['DayInterval'] == '1M') |
        (moments_dataset['DayInterval'] == '3M') |
        (moments_dataset['DayInterval'] == '6M') |
        (moments_dataset['DayInterval'] == '9M') |
        (moments_dataset['DayInterval'] == '1Y') ].reset_index(drop=True)
    # moments_maturity.to_csv(args.result_save_dir +\
        'Moments_dataset.csv', index=False)
    moments_maturity = moments_maturity.groupby(by=['date', 'DayInterval'])\
    .agg({'std':'mean', 'skew':'mean', 'kurt':'mean'}).reset_index()
    moments_maturity1W = moments_maturity[moments_maturity\
        ['DayInterval']== '1W'].reset_index(drop=True)
    moments_maturity1M = moments_maturity[moments_maturity\
        ['DayInterval']== '1M'].reset_index(drop=True)
    moments_maturity3M = moments_maturity[moments_maturity\
        ['DayInterval']== '3M'].reset_index(drop=True)
    moments_maturity6M = moments_maturity[moments_maturity\
        ['DayInterval']== '6M'].reset_index(drop=True)
    moments_maturity9M = moments_maturity[moments_maturity\
        ['DayInterval']== '9M'].reset_index(drop=True)
    moments_maturity1Y = moments_maturity[moments_maturity\
        ['DayInterval']== '1Y'].reset_index(drop=True)
    print(moments_maturity1W.shape[0], moments_maturity1M.shape[0],\
        moments_maturity3M.shape[0], moments_maturity6M.shape[0],\
moments_maturity9M.shape[0], moments_maturity1Y.shape[0])
```

3.2.5 Separating and combining data

The processed data sets will be split into multiple DataFrames based on time intervals (e.g., one week, one month, three months, etc.). The code then merges these separate DataFrames into a total dataset. Next, it creates a new DataFrame with all the unique dates and merges the statistics for the different time intervals into this new DataFrame.

Code Listing

Listing 10: Parameter Parsing and Configuration

```
#=============================================================
moments_maturities = pd.concat([moments_maturity1W,\
    moments_maturity1M,moments_maturity3M, moments_maturity6M,\
    moments_maturity9M,moments_maturity1Y], axis=0)
# moments_maturity = moments_maturity.groupby(by=['date','DayInterval']).\
    agg({'std':'mean', 'skew':'mean', 'kurt':'mean'}).reset_index()
moments_dates = pd.DataFrame(moments_maturity['date'].unique(),\
columns=['date'])
moments_dates = moments_dates.merge(moments_maturity1W[['date','std',\
    'skew', 'kurt']], on=['date'], how='left')
moments_dates = moments_dates.merge(moments_maturity1M[['date','std',\
    'skew', 'kurt']], on=['date'], how='left', suffixes=['', '_1M'])
moments_dates = moments_dates.merge(moments_maturity3M[['date','std',\
    'skew', 'kurt']], on=['date'], how='left', suffixes=['', '_3M'])
moments_dates = moments_dates.merge(moments_maturity6M[['date','std',\
    'skew', 'kurt']], on=['date'], how='left', suffixes=['', '_6M'])
moments_dates = moments_dates.merge(moments_maturity9M[['date','std',\
    'skew', 'kurt']], on=['date'], how='left', suffixes=['', '_9M'])
moments_dates = moments_dates.merge(moments_maturity1Y[['date','std',\
    'skew', 'kurt']], on=['date'], how='left', suffixes=['', '_1Y'])

moments_dates_stat = moments_dates.T.isna().sum()
moments_dates_stat[moments_dates_stat <= 12]
moments_dates_stat.min()
moments_dates_stat.max()
```

3.2.6 Handling Missing Values

The code checks the merged DataFrame for missing values and handles them. Specifically, it counts the number of missing values for each date and further analyses the dates with fewer missing values.

Code Listing

Listing 11: Parameter setting

```
#=============================================================================
plot_raw = moments_dates.iloc[moments_dates_stat[moments_dates_stat\
    <= 2].index, :]
# plot_raw = plot_raw[plot_raw['date'] >= 20180101]
plot_raw_rnm2 = plot_raw[['date', 'std', 'std_1M', 'std_3M',\
    'std_6M', 'std_9M', 'std_1Y']].reset_index(drop=True)
plot_raw_rnm3 = plot_raw[['date', 'skew', 'skew_1M', 'skew_3M',\
    'skew_6M', 'skew_9M', 'skew_1Y']].reset_index(drop=True)
plot_raw_rnm4 = plot_raw[['date', 'kurt', 'kurt_1M', 'kurt_3M',\
    'kurt_6M', 'kurt_9M', 'kurt_1Y']].reset_index(drop=True)

if plot_raw.shape[0]>=10:
    np.random.seed(101)
    # idx_rand = np.random.randint(plot_raw.shape[0], size=10)
    idx_rand = np.random.choice(plot_raw.shape[0],10,replace=False)
    idx_rand = np.sort(idx_rand)
    plot_dataset_rnm2=plot_raw_rnm2.iloc[idx_rand,:].set_index('date').T
    plot_dataset_rnm3=plot_raw_rnm3.iloc[idx_rand,:].set_index('date').T
    plot_dataset_rnm4=plot_raw_rnm4.iloc[idx_rand,:].set_index('date').T
else:
    plot_dataset_rnm2 = plot_raw_rnm2.set_index('date').T
    plot_dataset_rnm3 = plot_raw_rnm3.set_index('date').T
    plot_dataset_rnm4 = plot_raw_rnm4.set_index('date').T

# manually selected dates
manually_dates = pd.DataFrame([20200612, 20200911, 20220114,\
    20220310, 20230113], columns=['date'])
manually_plot_raw=pd.merge(manually_dates,plot_raw,on=['date'],how='left')
plot_raw_rnm2 = manually_plot_raw[['date', 'std', 'std_1M',\
'std_3M', 'std_6M', 'std_9M', 'std_1Y']].reset_index(drop=True)
plot_raw_rnm3 = manually_plot_raw[['date', 'skew', 'skew_1M',\
'skew_3M','skew_6M', 'skew_9M', 'skew_1Y']].reset_index(drop=True)
plot_raw_rnm4 = manually_plot_raw[['date', 'kurt', 'kurt_1M',\
'kurt_3M','kurt_6M', 'kurt_9M', 'kurt_1Y']].reset_index(drop=True)
plot_dataset_rnm2 = plot_raw_rnm2.set_index('date').T
plot_dataset_rnm3 = plot_raw_rnm3.set_index('date').T
plot_dataset_rnm4 = plot_raw_rnm4.set_index('date').T
```

3.2.7 Plotting

Finally, the code uses matplotlib and PdfPages to generate and save graphs. It will plot graphs for different statistics (standard deviation, skewness, kurtosis) and save these graphs as PDF files. During the plotting process, the code handles some manually selected dates to ensure that the charts present representative data.

Code Listing

Listing 12: Plotting

```
#=============================================================================
    pdf = PdfPages(args.result_save_dir + 'RNM2_manualdates.pdf')
    figsize=(10, 4)
    plt.figure(figsize=figsize)
    for col in plot_dataset_rnm2.columns:
        plt.plot(['RNM2_1W','RNM2_1M','RNM2_3M','RNM2_6M','RNM2_9M',\
            'RNM2_1Y'],plot_dataset_rnm2[col],label=col,linestyle='--')
        plt.scatter(['RNM2_1W', 'RNM2_1M', 'RNM2_3M', 'RNM2_6M',\
            'RNM2_9M',\'RNM2_1Y'], plot_dataset_rnm2[col])
        plt.legend()
    pdf.savefig(bbox_inches='tight', dpi=400)
    plt.show()
    pdf.close()
```

Code Listing

Listing 13: Plotting

```
#=========================================================================
#%% rnm3 plot
pdf = PdfPages(args.result_save_dir + 'RNM3_manualdates.pdf')
plt.figure(figsize=figsize)
for col in plot_dataset_rnm3.columns:
    plt.plot(['RNM3_1W', 'RNM3_1M', 'RNM3_3M', 'RNM3_6M', 'RNM3_9M',\
        'RNM3_1Y'], plot_dataset_rnm3[col], label=col, linestyle='—')
    plt.scatter(['RNM3_1W', 'RNM3_1M', 'RNM3_3M', 'RNM3_6M', 'RNM3_9M',\
        'RNM3_1Y'], plot_dataset_rnm3[col])
    plt.legend()
pdf.savefig(bbox_inches='tight', dpi=400)
plt.show()
pdf.close()

#%% rnm4 plot
pdf = PdfPages(args.result_save_dir + 'RNM4_manualdates.pdf')
plt.figure(figsize=figsize)
for col in plot_dataset_rnm4.columns:
    plt.plot(['RNM4_1W', 'RNM4_1M', 'RNM4_3M', 'RNM4_6M', 'RNM4_9M',\
        'RNM4_1Y'], plot_dataset_rnm4[col], label=col, linestyle='—')
    plt.scatter(['RNM4_1W', 'RNM4_1M', 'RNM4_3M', 'RNM4_6M', 'RNM4_9M',\
        'RNM4_1Y'], plot_dataset_rnm4[col])
    plt.legend()
pdf.savefig(bbox_inches='tight', dpi=400)
plt.show()
pdf.close()
```

In modern computing environments, efficient parameter management and environment settings are critical steps to ensure that data processing and model training processes run smoothly[5]. The purpose of this code is to describe a piece of code for risk-neutral density extraction that uses a generative machine learning approach and optimises its execution with appropriate environment variable settings and command line parameter parsing.

4 Further directions

4.1 Potential Problem

Static Nature of the Model

The models in the paper are primarily static, meaning they focus on distributions at specific points in time rather than dynamic changes over time. Such static models are more suitable for price prediction and risk assessment at a single moment but fall short in capturing time-related market dynamics.

Optimal transport theory, on the other hand, is particularly well-suited for analyzing distribution changes over different time points, which the models in the paper may not adequately address.

Temporal Discretization

The RN-MLP and RN-DMLP models in the paper use neural network parameterization to handle distributions at different maturities, but these models are discrete in time. Optimal transport theory typically deals with continuous-time distribution shifts, especially relevant in high-frequency trading and continuous market monitoring. Since the paper's models are based on discrete time points, they might not fully leverage optimal transport theory to handle smooth transitions and dynamic changes over time.

Model Complexity and Applicability Optimal transport theory involves complex optimization problems and significant computational effort. This complexity might increase the model's overall difficulty and computational cost. The authors might have chosen not to use optimal transport theory to simplify the model and enhance computational efficiency, ensuring practical feasibility and operability[6]. However, this decision could come at the cost of losing the ability to capture time correlations and market dynamics effectively.

Influence of Traditional Option Pricing Methods

Classical option pricing models (like the Black-Scholes model) usually assume market stability at specific points in time and do not consider time-related distribution changes.[4] This assumption has a broad application base in financial mathematics. The models in the paper might be more influenced by these traditional methods rather than adopting the more complex and cutting-edge optimal transport theory[7]. Neglect of Market Dynamic Changes Optimal transport theory can provide deeper market dynamic analysis, especially when dealing with non-Gaussian distributions and extreme market events. However, the paper might focus more on capturing complex distribution shapes through neural networks and machine learning methods rather than dealing with the dynamic evolution of distributions over time. This focus could limit the models' adaptability during significant market changes or sudden events[2]. Conclusion While the paper demonstrates the ability to capture complex distributions at single points in time, it might lack adequate modeling for temporal dynamics and continuity. The failure to leverage optimal transport theory is indeed a significant limitation, as it might lead to suboptimal performance in handling market dynamics and time correlations.

To enhance the model's robustness and applicability, future research could consider the following points:

- **Incorporating Time Series Models:** Combine time series analysis methods to handle dynamic changes over time.
- **Adopting Optimal Transport Theory:** Utilize optimal transport theory to manage probability distribution shifts between different time points.
- **Enhancing Model Continuity:** Reduce errors from temporal discretization to ensure accuracy when handling continuous-time data.

4.2 A Possible Solution: Optimal Transport Theory

4.2.1 A brief Introduction

Optimal Transport (OT) is a mathematical theory focused on finding the most efficient way to move mass from one distribution to another. This theory has significant applications in various fields, including economics, fluid dynamics, and machine learning.

Mathematical Foundation of Optimal Transport

- **The Monge Formulation** The Monge formulation of the optimal transport problem is based on finding a mapping $T : \mathbb{R}^d \to \mathbb{R}^d$ that pushes forward a probability measure P to another probability measure Q. This is denoted as $T_{\#}P = Q$, where $T_{\#}$ is the push-forward operator. The objective is to minimize the transportation cost:

$$\mathcal{L}_M(T) = \mathbb{E}_x[c(x, T(x))],$$

where $c : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$ is the ground-cost function that measures the cost of moving mass from x to $T(x)$.

- **The Kantorovich Formulation**

The Monge formulation is challenging to analyze due to its restrictive nature. The Kantorovich formulation generalizes this by considering a transport plan γ, which is a probability measure on $\mathbb{R}^d \times \mathbb{R}^d$ with marginals P and Q. The objective is to minimize:

$$\mathcal{L}_K(\gamma) = \mathbb{E}_{(x_1,x_2)\sim\gamma}[c(x_1, x_2)].$$

The set of all such transport plans γ is denoted by $\Gamma(P, Q)$. Thus, the Kantorovich formulation of the optimal transport problem is:

$$\inf_{\gamma\in\Gamma(P,Q)} \mathcal{L}_K(\gamma).$$

- **Duality in Optimal Transport**

The Kantorovich problem has a dual formulation involving Kantorovich potentials $\varphi : \mathbb{R}^d \to \mathbb{R}$ and $\psi : \mathbb{R}^d \to \mathbb{R}$. The dual problem maximizes:

$$\mathcal{L}_K^*(\varphi, \psi) = \mathbb{E}_x[\varphi(x)] + \mathbb{E}_x[\psi(x)],$$

subject to the constraint $\varphi(x_1) + \psi(x_2) \leq c(x_1, x_2)$.

When the ground-cost function is $c(x_1, x_2) = \|x_1 - x_2\|_2^2$, the celebrated Brenier theorem states that the Monge problem is equivalent to the Kantorovich problem with $T = \nabla\varphi$.

- **Kantorovich-Rubinstein Duality**

For the special case where the cost function is the ℓ_1 norm, i.e., $c(x_1, x_2) = \|x_1 - x_2\|_1$, the Kantorovich-Rubinstein duality maximizes:

$$\mathcal{L}_{KR}^*(\varphi) = \mathbb{E}_x[\varphi(x)] - \mathbb{E}_x[\varphi(x)],$$

over all 1-Lipschitz continuous functions φ.

Computational Aspects

- **Entropic Regularization**

Computing the optimal transport distance can be computationally intensive, especially in high dimensions.[8] Entropic regularization introduces a regularization term to the Kantorovich formulation, making it more tractable:

$$\mathcal{L}_K^\epsilon(\gamma) = \mathbb{E}_{(x_1,x_2)\sim\gamma}[c(x_1, x_2)] + \epsilon\mathrm{KL}(\gamma\|P \otimes Q),$$

where ϵ is the regularization parameter and KL denotes the Kullback-Leibler divergence. The Sinkhorn algorithm is often used to solve this regularized problem efficiently.

- **Applications in Machine Learning**

  Optimal transport theory has been extensively applied in machine learning, particularly in generative modeling.[9] [4]It provides a robust framework for comparing probability distributions, which is crucial for tasks such as training generative adversarial networks (GANs), domain adaptation, and clustering.

4.2.2 Generative Model in OT

There are mainly three types of generative models that benefit from OT, namely, GANs, VAEs, and normalizing flows. Here we only talk about GANs[7].

GANs are a neural network architecture proposed for sampling from a complex probability distribution P_0. The principle behind this architecture is building a function g_θ, that generates samples in an input space $\mathcal{X}$ from latent vectors $\mathbf{z} \in \mathcal{Z}$, sampled from a simple distribution Q (e.g., $\mathcal{N}(0, \sigma^2)$). A GAN is composed of two networks: a generator $g_\theta : \mathcal{Z} \to \mathcal{X}$, and a discriminator $h_\xi : \mathcal{X} \to [0, 1]$. $\mathcal{X}$ is called the input spaceand $\mathcal{Z}$ is the latent space (e.g., an Euclidean space $\mathbb{R}^p$).

The GAN objective function is:

$$\mathcal{L}_G(\theta, \xi) = \mathbb{E}_{\mathbf{x}\sim P_0}[\log h_\xi(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim Q}[\log(1 - h_\xi(g_\theta(\mathbf{z})))].$$

This function is minimized w.r.t. θ and maximized w.r.t. ξ. In other words, h_ξ is trained to maximize the probability of assigning the correct label to $\mathbf{x}$ (labeled 1) and $g_\theta(\mathbf{z})$ (labeled 0). g_θ, on the other hand, is trained to minimize $\log(1 - h_\xi(g_\theta(\mathbf{z})))$. As a consequence, it minimizes the probability that h_ξ guesses its samples correctly.

As shown in [25], for an optimal discriminator h^*_ξ, the generator cost is equivalent to the so-called Jensen-Shannon (JS) divergence. [2]In this sense, Arjovsky et al. [91] proposed the Wasserstein GAN (WGAN) algorithm, which substitutes the JS divergence by the Kantorovich-Rubinstein (KR) metric:

$$\min_\theta \max_{h_\xi \in \mathrm{Lip}_1} \mathcal{L}_W(\theta, \xi) = \mathbb{E}_{\mathbf{x}\sim P_0}[h_\xi(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim Q}[h_\xi(g_\theta(\mathbf{z}))].$$

However, the WGAN involves maximizing $\mathcal{L}_W$ over $h_\xi \in \mathrm{Lip}_1$, which is not straightforward for neural networks. Possible solutions include: (i) clipping the weights ξ [91]; (ii) penalizing the gradients of h_ξ [92]; (iii) normalizing the spectrum of the weight matrices [93]. Despite these improvements, several works have confirmed that the WGAN and its variants do not estimate the Wasserstein distance [94, 95, 96].

Another approach is to calculate $\nabla_\theta W_p(P_\theta, P_0)$ using the primal problem. For this, Genevay et al. [43] consider estimating $W_p(P_\theta, P_0)$ through averaging the loss over mini-batches. Calculating derivatives $\nabla_\theta W_p(P_\theta, P_0)$ can be done by back-propagating through Sinkhorn iterations or using the Envelope Theorem [97, 98].

Genevay et al. [43] also propose learning a parametric ground cost $(C_\eta)_{ij} = \|f_\eta(\mathbf{x}_i(P_\theta)) - f_\eta(\mathbf{x}_j(P_0))\|$, where $f_\eta : \mathcal{X} \to \mathcal{Z}$ is a neural network that learns a representation for $\mathbf{x} \in \mathcal{X}$.[6] The overall optimization problem proposed is:

$$\min_\theta \max_\eta S_{c_\eta,\epsilon}(P_\theta, P_0).$$

Engineering or learning the ground cost c_η is crucial for having a meaningful metric between distributions[10], as it serves to compute distances between samples. The Euclidean distance, for example, does not correlate well with perceptual or semantic similarity between images.[4]

Lastly, sliced Wasserstein metrics offer computational advantages and robustness against the curse of dimensionality, making them suitable for high-dimensional data in generative modeling.[1]

References

[1] Yan X. Leung C. H. Wu Q Xian, Z. Risk-neutral generative networks. incorporating machine learning advances, 2024.

[2] Z. Song and D. Xiu. A tale of two option markets: Pricing kernels and volatility risk. journal of econometrics, 2016.

[3] M. et al. Rubinstein. Edgeworth binomial trees, 1998.

[4] McDonald J. B. et al. Bookstaber, R. M. A general distribution for describing security price returns. the journal of business, 1987.

[5] B Bahra. Implied risk-neutral probability density functions from option prices: theory and application. *Bank of England*, 1997.

[6] Zheng Y. Yang, Y. and T Hospedales. Gated neural networks for option pricing: Rationality by design, 2017.

[7] S. L Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options, 1993.

[8] John Wiley Sons. Financial modelling: Theory, implementation and practice with matlab source. *Bank of England*, 2013.

[9] Tan H M Zhang Y, Zhao P. Cost-sensitive portfolio selection via deep reinforcement learning[j], 2022.

[10] Radford A et al Salimans T, Zhang H. Improving gans using optimal transport[j], 2018.