Московский государственный университет им. М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математической статистики
Лаборатория статистического анализа

Выпускная квалификационная работа

Нейросеть для ценообразования опционов и ее сравнение с классическими методами

Харинаев Артём Эдуардович

Научный руководитель:
к.ф.-м.н. ______________________ А.Н. Дойников

Заведующий кафедрой:
д.ф.-м.н., профессор ______________________ В.Ю. Королёв

Москва, 2023

Содержание

1 Введение

Опционы - это финансовые инструменты, дающие право, но не обязательство, купить (call option) или продать (put option) базовый актив по заранее установленной цене и в определенное время. Опционы широко используются на финансовых рынках в целях хеджирования, спекуляции и арбитража. Их можно покупать и продавать, как и любые другие финансовые активы, а цены на них определяются различными факторами, включая цену базового актива, волатильность рынка, время до истечения срока действия и безрисковую процентную ставку.

Опционы являются важными финансовыми инструментами, поскольку они дают инвесторам и трейдерам возможность управлять рисками портфеля активов (*хеджирование*): предположим, инвестор владеет портфелем акций, которые, по его мнению, будут расти в цене в течение следующих нескольких месяцев. Однако его беспокоит возможность рыночного спада, который может негативно повлиять на стоимость его портфеля. Чтобы застраховаться от этого риска, инвестор может приобрести опционы put на индекс, который отслеживает динамику рынка. Если на рынке действительно произойдет спад, стоимость опционов put возрастет, компенсируя потери в портфеле инвестора.

Для такого использования инвесторам необходимо точное определение цены опционов. Модель ценообразования опционов Блэка-Шоулза является одной из наиболее широко используемых моделей. Она предполагает, что базовый актив следует геометрическому броуновскому движению, и учитывает такие факторы, как волатильность, время до истечения срока действия и процентные ставки.

Но у данной модели есть ограничения - она предполагает постоянную волатильность, а на практике возникает явление, при котором волатильность зависит от цен исполнения (страйков). Это явление получило название *улыбка волатильности*.

В данной работе предлагается способ моделирования улыбки волатильности с помощью нейронной сети и проводится его сравнение с более классическими моделями ценообразования опционов.

2 Модель Блэка-Шоулза

2.1 Описание модели

Модель ценообразования опционов Блэка-Шоулза впервые была выведена Фишером Блэком и Майроном Шоулзом в 1973 году в статье «The Pricing of Options and Corporate Liabilities» [1].

В то время существовал растущий рынок опционных контрактов, но не было общепринятого метода определения цены этих контрактов. Фишер Блэк и Майрон Шоулз пытались разработать модель, которая обеспечила бы более точную оценку справедливой цены опционов.

Их модель была основана на нескольких предположениях, включая предположение геометрическом броуновском движении цены базового актива, отсутствие транзакционных издержек и предположение риск-нейтральности инвесторов. Модель использует дифференциальные уравнения для описания динамики цены опциона в зависимости от различных факторов, таких как цена базового актива, время и волатильность.

Модель Блэка-Шоулза: При предположении броуновского движения цены базового актива теоретическая цена на европейские опционы определяется по формулам

$$V_{call} = f\mathcal{N}(d_1) - Ke^{-rt_{ex}}\mathcal{N}(d_2) \tag{2.1}$$

$$V_{put} = Ke^{-rt_{ex}}\mathcal{N}(-d_2) - f\mathcal{N}(-d_1) \tag{2.2}$$

$$d_1 = \frac{\ln\frac{f}{K} + (r + \frac{1}{2}\sigma^2)t_{ex}}{\sigma\sqrt{t_{ex}}} \tag{2.3}$$

$$d_2 = d_1 - \sigma\sqrt{t_{ex}} \tag{2.4}$$

Обозначения:

- V_{call}, V_{put} - цены европейский call и put опционов
- f - текущая цена базового актива
- K - цена исполнения опциона (страйк)
- $\mathcal{N}(x)$ - функция распределения стандартного нормального распределения
- r - безрисковая процентная ставка
- t_{ex} - время до даты исполнения (в годах)
- σ - волатильность цены базового актива

2.2 Улыбка волатильности

Обладая информацией об установившихся ценах опционов на рынке - V_{mrkt}, можно с помощью численных методов восстановить подразумеваемую волатильность, которая должна соответствовать такой цене в формуле Блэка-Шоулза $BS(\sigma, f, K, t_{ex}, r) = V_{mrkt}$.

$$\sigma = BS^{-1}(V_{mrkt}, f, K, t_{ex}, r) \tag{2.5}$$

Тогда будет наблюдаться следующее явление: волатильность не константна, а зависит от цены исполнения - *улыбка волатильности* (Рис. 1).

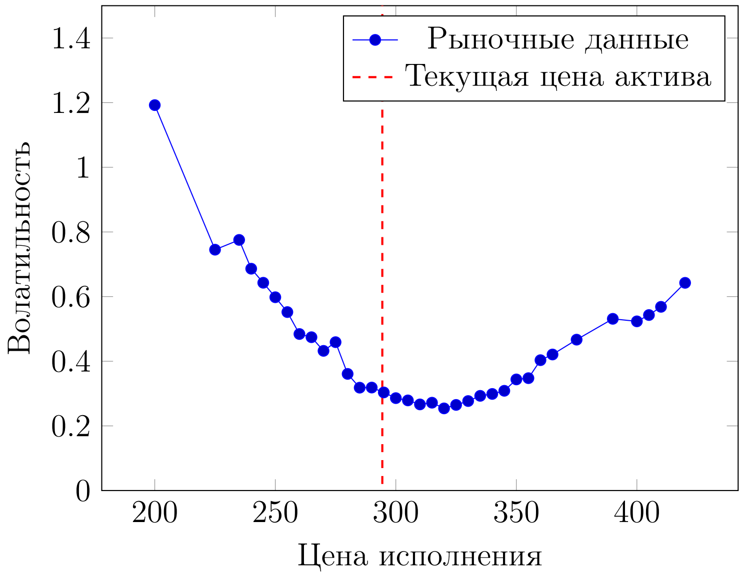


Рис. 1: Улыбка волатильности

2.3 Недостатки модели

У модели Блэка-Шоулза есть несколько недостатков

1. **Предположение о константной волатильности**. Модель Блэка-Шоулза предполагает, что волатильность базового актива одинакова для опционов, отличающихся только ценой исполнения. В реальности волатильность может значительно изменяться в зависимости от соотношения цены базового актива к цене исполнения опциона, что может привести к неточностям в прогнозах модели. Возникает, так называемая, *улыбка волатильности* (см Рис. 1).

2. **Отсутствие рыночных ограничений**. Модель игнорирует различные факторы, такие как транзакционные издержки, спрэды на покупку и ликвидность, которые могут влиять на цену опционных контрактов на реальных финансовых рынках.

3. **Нереалистичные предположения о поведении инвесторов**. Модель предполагает, что инвесторы нейтральны к риску, что означает, что они не требуют более высокой прибыли за несение риска. В реальности инвесторы, как правило, не являются нейтральными к риску и требуют более высокую прибыль в качестве компенсации за принятие риска.

4. **Ограниченная применимость к нестандартным опционам**. Модель Блэка-Шоулза разработана для ценообразования опционов европейского типа, которые могут быть исполнены только по истечении срока действия. Она может быть не столь точной для определения цены опционов американского типа, которые могут быть исполнены в любое время до истечения срока, или других более сложных типов опционов.

В данной работе рассматриваются подходы к решению первого недостатка - отсутствия учета зависимости подразумеваемой волатильности от цены исполнения опциона (страйка) - улыбки волатильности (implied volatility smile).

3 SABR

Модель SABR (Stochastic Alpha, Beta, Rho) была разработана в 2002 году Патриком Хаганом, Дипом Кумаром, Эндрю Лесневски и Дианой Вудворд [2]. Модель была разработана для устранения ограничений модели Блэка-Шоулза.

Название SABR происходит от трех параметров, которые модель использует для описания поведения волатильности базового актива. α представляет собой начальный уровень волатильности, β - чувствительность волатильности к изменениям цены базового актива, а ρ - корреляцию между ценой базового актива и его волатильностью.

Одним из ключевых преимуществ модели SABR является ее способность точно отражать эффект улыбки волатильности, наблюдаемый в реальных рыночных ценах.

Цена базового актива S и его волатильность α удовлетворяют уравнениям

$$dS = \alpha S^{\beta} dW_1, \quad S(0) = f \tag{3.1}$$

$$d\alpha = \nu \alpha dW_2, \quad \alpha(0) = \alpha_0 \tag{3.2}$$

$$dW_1 dW_2 = \rho dt \tag{3.3}$$

где W_1 и W_2 винеровские процессы с коэффициентом корреляции ρ.

Волатильность в модели SABR: Ценообразование подчиняется формулам Блэка-Шоулза (2.1) - (2.4), где предполагаемая волатильность $\sigma_B(f, K)$ вычисляется по формуле

$$\begin{aligned}\sigma_B(f, K) = {} & \frac{\alpha}{(fK)^{(1-\beta)/2} \cdot \left(1 + \frac{(1-\beta)^2}{24} \log^2 \frac{f}{K} + \frac{(1-\beta)^4}{1920} \log^4 \frac{f}{K} + \dots\right)} \cdot \left(\frac{z}{x(z)}\right) \cdot \\ & \cdot \left(1 + \left(\frac{(1-\beta)^2 \alpha^2}{24 (fK)^{1-\beta}} + \frac{\rho \beta \nu \alpha}{4 (fK)^{(1-\beta)/2}} + \frac{2 - 3\rho^2}{24} \nu^2\right) t_{ex} + \dots\right)\end{aligned} \tag{3.4}$$

где

$$z = \frac{\nu}{\alpha} (fK)^{(1-\beta)/2} \log \frac{f}{K} \tag{3.5}$$

$$x(z) = \log\left(\frac{\sqrt{1 - 2\rho z + z^2} + z - \rho}{1 - \rho}\right) \tag{3.6}$$

Для опционов at-the-money достигается равенство $f = K$, при подстановке в формулу (3.4) получается

$$\sigma_{ATM} = \frac{\alpha}{f^{1-\beta}} \left(1 + \left(\frac{(1-\beta)^2}{24}\frac{\alpha^2}{f^{2(1-\beta)}} + \frac{\rho\beta\nu\alpha}{4f^{(1-\beta)}} + \frac{2-3\rho^2}{24}\nu^2\right) t_{ex} + \dots\right) \tag{3.7}$$

4 Heston

Модель Хестона - это модель стохастической волатильности, разработанная Стивеном Хестоном в 1993 году [3]. Она была разработана для устранения некоторых ограничений модели Блэка-Шоулза и других традиционных моделей ценообразования опционов, в частности, предположения о постоянной волатильности.

Модель Хестона - это двухфакторная модель, которая предполагает, что волатильность базового актива следует стохастическому процессу. Это позволяет модели отражать эффект улыбки волатильности, наблюдаемый в реальных рыночных данных.

Допускается предположение, что цена базового актива и её волатильность удовлетворяют процессам

$$dS(t) = \mu S dt + \sqrt{v(t)} S dW_1(t), \tag{4.1}$$

$$dv(t) = k[\theta - v(t)]dt + \sigma\sqrt{v(t)} dW_2(t) \tag{4.2}$$

где $W_1(t), W_2(t)$ - винеровские процессы с коэффициентом корреляции ρ. Параметры:

- θ - долгосрочная дисперсия цен
- k - скорость возврата к долгосрочной дисперсии цен
- σ - волатильность волатильности
- ρ - коэффициент корреляции между винеровскими процессами
- v_0 - начальная дисперсия цены

Предположение отсутствия арбитража и стремление инвесторов к безрисковым портфелям дают дифференциальное уравнение в частных производных на стоимость деривативов $V(S, v, t)$

$$\begin{split}\frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\sigma v S\frac{\partial^2 V}{\partial S \partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + rS\frac{\partial V}{\partial S} + \\ +(k[\theta - v(t)] - \lambda(S, v, t))\frac{\partial V}{\partial v} + \frac{\partial V}{\partial t} = 0\end{split} \tag{4.3}$$

Цена опциона в модели Heston: Решение уравнения 4.3 дает формулу цены опциона по модели Heston:

$$
\begin{aligned}
V(S,v,t) = \frac{1}{2}S(t) + \frac{e^{-rt_{ex}}}{\pi}\int\limits_0^{\infty} Re\left[\frac{K^{-i\phi}f(i\phi+1)}{i\phi}\right]d\phi - \\
-Ke^{-rt_{ex}}\left(\frac{1}{2}+\frac{1}{\pi}\int\limits_0^{\infty} Re\left[\frac{K^{-i\phi}f(i\phi)}{i\phi}\right]d\phi\right)
\end{aligned}
\tag{4.4}
$$

где $f(i\phi)$ - характеристическая функция модели Хестона, удовлетворяющая условиям

$$f(i\phi) = \exp\left[A(\tau) + C(\tau)V_t + i\phi X_t\right] \tag{4.5}$$

$$A(\tau) = ri\phi\tau + \frac{k\theta}{\sigma^2}\left[-(\rho\sigma i\phi - k - M)\tau - 2\ln\frac{1-Ne^{M\tau}}{1-N}\right] \tag{4.6}$$

$$C(\tau) = \frac{(e^{M\tau}-1)(\rho\sigma i\phi - k - M)}{\sigma^2(1-Ne^{M\tau})} \tag{4.7}$$

$$M = \sqrt{(\rho\sigma i\phi - k)^2 + \sigma^2(i\phi + \phi^2)} \tag{4.8}$$

$$N = \frac{\rho\sigma i\phi - k - M}{\rho\sigma i\phi - k + M} \tag{4.9}$$

5 Variance-Gamma

Модель Variance-Gamma (VG) - это модель стохастической волатильности, которая была предложена Дилипом Маданом в 1998 году [4], [5], [6].

В модели Variance-Gamma цена базового актива определяется следующим образом

$$S_t = S_0 \exp\left[(\mu+\omega)t + X(t,\sigma,\nu,\theta)\right] \tag{5.1}$$

где S_0 - начальная цена актива, $X(t,\sigma,\nu,\theta)$ - variance-gamma процесс с параметром дрейфа θ и коэффициентом отклонения ν (см. формулы (1)-(5) в [4]), μ - ожидаемый уровень прибыли, σ - волатильность в моменте, ω определяется через характеристическую функцию variance-gamma процесса:

$$\omega = -\frac{1}{t}ln\left[\phi_X(u,t)|_{u=1/i}\right] \tag{5.2}$$

,

где i - мнимая единица, $\phi_X(u,t)$ - характеристическая функция variance-gamma процесса:

$$\phi_X(u,t) = \mathbb{E}\{iuX(t)\} = \left(\frac{1}{1-i\theta\nu u + \sigma^2u^2\nu/2}\right)^{\frac{t}{\nu}} \tag{5.3}$$

Цена опциона в модели Variance-Gamma определяется следующим образом

$$C_{VG}(S(0),K,t) = S(0)\Psi(d\sqrt{\frac{1-c_1}{\nu}},(\alpha+s)\sqrt{\frac{\nu}{1-c_1}},\frac{t}{\nu}) \\ -Ke^{-rt}\Psi(d\sqrt{\frac{1-c_2}{\nu}},\alpha s\sqrt{\frac{\nu}{1-c_2}},\frac{t}{\nu}) \tag{5.4}$$

где:

$$d = \frac{1}{d}\left[ln\frac{S(0)}{K} + rt + \frac{t}{\nu}ln\frac{1-c_1}{1-c_2}\right], \tag{5.5}$$

$$c_1 = \nu(\alpha+s)^2/2 \tag{5.6}$$

$$c_2 = \nu\alpha^2/2 \tag{5.7}$$

$$\alpha = -\frac{\theta s}{\sigma^2} \tag{5.8}$$

$$s = \frac{\sigma}{\sqrt{1+\left(\frac{\theta}{\sigma}\right)^2\frac{\nu}{2}}} \tag{5.9}$$

Ψ - модифицированная функция Бесселя второго рода (см (A11) в [4]).

6 Искусственная нейросеть

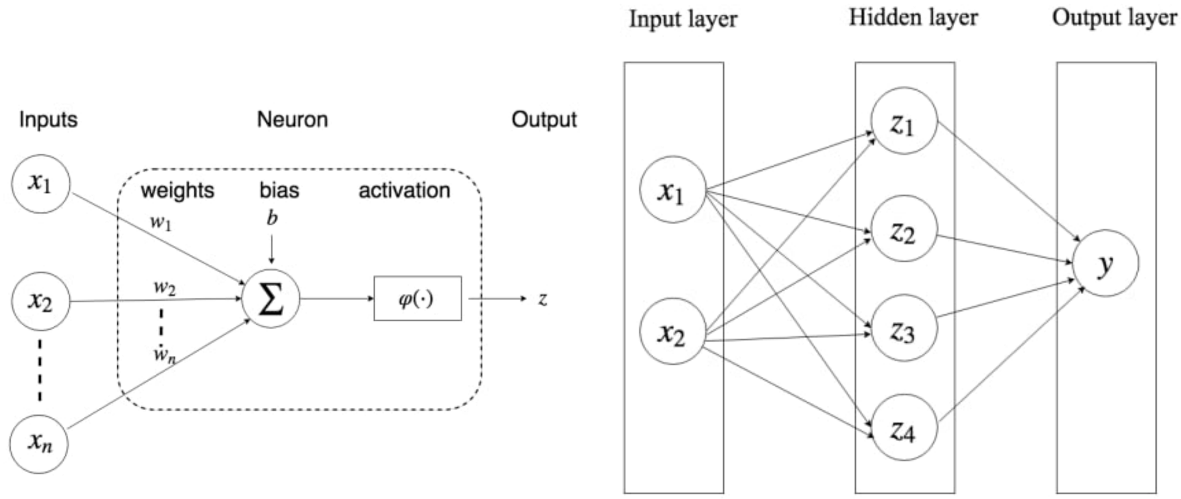
Искусственная нейросеть (ANN/NN - Artificial Neural Network) - универсальный аппроксиматор $\forall$ функции: $\mathbb{R}^m \to \mathbb{R}^n$, состоящий из нейронов, объединенных в слои, нейроны получают входные данные, обрабатывают их с помощью набора нелинейных преобразований и генерируют выходной сигнал. Слои обычно делятся на входной, один или несколько скрытых, и выходной слой (Рис. 2).

ANN может быть представлена математически с помощью матричных операций и функций активации, таких как сигмоидная функция, функция ReLU или функция softmax. Веса (weights), смещения (biases) нейронов и другие параметры ANN подбираются в процессе обучения с использованием набора пар вход/выход, где предсказания сети сравниваются с истинными значениями целевой переменной, а ошибки минимизируются с помощью алгоритма оптимизации, например, градиентного спуска и алгоритма обратного распространения ошибки.

Для решения задачи предсказания подразумеваемой волатильности предлагается два варианта нейросети, отличающихся только входными данными:

A - На вход нейросети подается 4 значения:

1. $\log(\frac{f}{K})$ - логарифм отношения цены актива к цене исполнения



(a) Нейрон

(b) Слои нейросети

Рис. 2: Пример искусственной нейросети

2. t - число лет до исполнения опциона
3. r - безрисковая процентная ставка
4. $\log(\frac{V}{K})$ - логарифм отношения цены опциона к цене исполнения

В - На вход нейросети подается 3 значения - пункты 2-4 из предыдущего варианта (кроме $\log(\frac{V}{K})$)

В качестве целевой переменной, которую предсказывает нейросеть, выступает подразумеваемая волатильность (implied volatility). В каждой из нейросетей 4 скрытых слоя с числом нейронов на каждом из них - 256, 512, 256 и 64. В качестве функции активации выступает функция $ReLU(x) = max\{x, 0\}$.

Компоненты $\log(\frac{f}{K})$ и $\log(\frac{V}{K})$ имеют такой вид по двум причинам:

1. Нормализация на цену исполнения - для применимости на опционах, базовые активы которых обладают разными масштабами цен.
2. Логарифм - для численной стабильности обучения нейросети

Вариант А применим, когда необходимо ретроспективно оценить уровень волатильности актива по установившимся рыночным ценам на производный от него опцион. Однако, если необходимо дать оценку опциону, по которому еще нет заявок, или они недоступны, то есть недоступна компонента $\log(\frac{V}{K})$, применить нейросеть А нельзя. В таком случае применима нейросеть В, обладающая, очевидно, меньшей точностью в случаях, когда компонента $\log(\frac{V}{K})$ доступна.

7 Обучение на рыночных данных

7.1 Данные

Данные загружаются с помощью библиотеки *yfinance* [7] для языка Python от компании Yahoo. С ее помощью можно получить такую информацию по опционам на различные активы, как дата исполнения, цена исполнения (страйк - strike), минимальная предлагаемая цена продажи и максимальная предлагаемая цена покупки в данный момент (bid и ask) и рассчитанная рыночная подразумеваемая волатильность. Цену опциона будем вычислять, как среднее арифметическое цены продажи и цены покупки.

Для сравнения методов используем опционы на акции компании Microsoft с тикером MSFT с различными датами и ценами исполнения. Дата расчета - 16.03.2022.

Т.к. информация о покупках и продажах есть не для всех дат и цен исполнения, то для калибровки моделей используется усеченное подмножество опционов: берутся только те даты исполнения, для которых есть достаточное количество данных для калибровки, затем для этих дат производится пересечение множества доступных цен исполнения. В результате получается два множества: даты исполнения $\mathbb{T}$ и цены исполнения $\mathbb{K}$, такие, что $\forall t \in \mathbb{T}, K \in \mathbb{K} : \exists \sigma_{mrkt}(t, K), V_{mrkt}(t, k)$, где $\sigma_{mrkt}(t, K)$ - подразумеваемая рыночная волатильность, соответствующая дате исполнения t и цене исполнения K, $V_{mrkt}(t, k)$ - рыночная цена опциона, соответствующая дате исполнения t и цене исполнения K.

Для обучения нейросетей используем более широкий набор данных - опционы на акции всех компаний, входящих в индекс S&P 500. Дата расчета опционов - 29.04.2023.

7.2 Подход к обучению

Модели SABR, Heston и Variance-Gamma обладают параметрами, которые необходимо подобрать, минимизируя сумму квадратов ошибок восстановления поверхности волатильности.

$$RSS(t, K) = \sum_{i=1}^{n} (\sigma_{mrkt}(t, K) - \sigma_{model_p}(t, K))^2 \to \min_p \tag{7.1}$$

где

- $\sigma_{mrkt}(t, K)$ - рыночная подразумеваемая волатильность, соответствующая опциону с исполнением через t лет и ценой исполнения K
- $\sigma_{model_p}(t, K)$ - волатильность, полученная с помощью модели с параметрами $p \in \mathbb{R}^{p_{dim}}$, соответствующая опциону с исполнением через t лет и ценой исполнения K. $p_{dim} = 3$ для модели SABR, 5 для модели Heston и 3 для модели Variance-Gamma.

Т.к. параметры моделей обладают физическими границами значений (например, коэффициент корреляции в модели SABR $\rho \in [-1, 1]$), то минимизация осуществляется с помощью алгоритма L-BFGS-B (Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm with boundaries) [8], который умеет работать с параметрами, возможные значения которых ограничены.

7.2.1 SABR

Модель SABR дает явную формулу для вычисления $\sigma_{model_p}(t, K)$ - (3.2).
Параметры: $p = (\alpha, \beta, \rho, \nu) \in \mathbb{R}^4$

- $\alpha \in (0, +\infty)$
- $\beta \in [0, 1]$
- $\rho \in [-1, 1]$
- $\nu \in (0, +\infty)$

7.2.2 Heston

Для данной модели $\sigma_{model_p}(t, K)$ вычисляется с помощью формулы (2.5) с предварительным расчетом цены опциона (4.4).
Параметры: $p = (\theta, \kappa, \sigma, \rho, v_0) \in \mathbb{R}^5$

- $\theta \in (0, +\infty)$
- $\kappa \in (0, +\infty)$
- $\sigma \in (0, +\infty)$
- $\rho \in [-1, 1]$
- $v_0 \in (0, +\infty)$

7.2.3 Variance-Gamma

Для данной модели $\sigma_{model_p}(t, K)$ вычисляется с помощью формулы (2.5) с предварительным расчетом цены опциона (5.4).
Параметры: $p = (\theta, \nu, \sigma) \in \mathbb{R}^3$

- $\theta \in (-\infty, +\infty)$
- $\nu \in (0, +\infty)$
- $\sigma \in (0, +\infty)$

7.2.4 Нейросетевые модели

Нейросети обучаются с помощью стохастической мини-батч модификации градиентного спуска - алгоритма Adam [9], использующего адаптивный темп обучения. Обучение ведется на протяжении 100 эпох, где одна эпоха - полный проход всей обучающей выборке.

7.3 Результаты

Для каждой модели и двух типов опционов (call и put) построены поверхности волатильности, кривые зависимости цены опциона от цены исполнения для разных дат исполнения (см. Рис. 3-18 в приложении A), рассчитано качество предсказания моделей, а также время калибровки и применения моделей.

Качество предсказания измеряется по нескольким критериям:

1. **Train MSE vol.** - среднеквадратичное отклонение предсказаний модели от рыночных волатильностей для множеств $\mathbb{T}$ и $\mathbb{K}$ из пункта 7.1.

2. **All MSE vol.** - среднеквадратичное отклонение предсказаний модели от рыночных волатильностей для всех доступных данных по опционам на акции компании Microsoft.

3. **MSE prices** - среднеквадратичное отклонение цен опционов, рассчитанных на основе предсказаний модели, от рыночных цен волатильностей для всех доступных данных по опционам на акции компании Microsoft.

4. **MSE prices near** - среднеквадратичное отклонение цен опционов, рассчитанных на основе предсказаний модели, от рыночных цен волатильностей для опционов на акции компании Microsoft, до исполнения которых осталось не более 1 года.

5. **MSE prices far** - среднеквадратичное отклонение цен опционов, рассчитанных на основе предсказаний модели, от рыночных цен волатильностей для опционов на акции компании Microsoft, до исполнения которых осталось более 1 года.

Результаты работы моделей представлены в таблицах 1 - 4. Время калибровки и применения моделей представлено в таблице 5.

Model	Train MSE vol.	All MSE vol.
SABR	0,000190	0,023799
Heston	0,001006	0,024879
VG	0,001271	0,026898
NN A	0,000871	0,006513
NN B	0,001997	0,026678

Таблица 1: СКО поверхности волатильности для опционов CALL

Model	MSE prices	MSE prices near	MSE prices far
SABR	163,95	48,88	394,08
Heston	44,30	8,34	116,23
VG	33,14	9,02	81,38
NN A	152,42	47,97	361,32
NN B	251,67	65,74	623,53

Таблица 2: СКО цен опционов CALL

Model	Train MSE vol.	All MSE vol.
SABR	0,000211	0,011350
Heston	0,000975	0,014497
VG	0,000338	0,012986
NN A	0,000922	0,002667
NN B	0,003077	0,016381

Таблица 3: СКО поверхности волатильности для опционов PUT

Model	MSE prices	MSE prices near	MSE prices far
SABR	16,87	3,16	44,30
Heston	20,67	7,50	46,99
VG	22,09	4,88	56,50
NN A	11,28	2,08	29,70
NN B	7,53	3,45	15,70

Таблица 4: СКО цен опционов PUT

Модель	Время калибровки, сек	Время применения, милисек.
SABR	0,09	0,37
Heston	15,40	27,90
VG	16,60	144,00
NN A	-	4,52
NN B	-	4,66

Таблица 5: Время калибровки и применения моделей

8 Сравнение моделей

8.1 Качество предсказаний

Рассмотрим результаты по каждой метрике

- **Train MSE vol.** - наилучшая модель - SABR, т.к. она дает явную формулу, для определения волатильности, тогда как Heston и Variance-Gamma дают формулы для расчета цены опциона, а волатильность определяется уже исходя из цены опциона

- **All MSE vol.** - наилучшая модель - Нейросеть A, т.к. она обучена на широком наборе данных, тогда как модели SABR, Heston и Variance-Gamma калибруются на более узком.

- **MSE prices, MSE prices near, MSE prices far** - по данным метрикам наилучшие модели Heston и Variance-Gamma, т.к. они дают явные формулы для нахождения цены опционов, а для остальных моделей расчет цены опциона производится с помощью модели Блэка-Шоулза с использованием рассчитанных подразумеваемых волатильностей (2.1, 2.2). Также, заметно, что все модели лучше предсказывают цены опционов с более близкими датами исполнения, чем с более дальними, это объяснимо меньшим интересом участников рынка к более дальним опционам, и как следствие - большей случайности в данных.

8.2 Время работы

Для нейросетей не указано время калибровки, т.к. для повторного использования на других опционах их не нужно переобучать.

С точки зрения времени калибровки и применения наилучшая модель - SABR, т.к. в ней используется простая формула, не требующая больших вычислительных ресурсов. Нейросетевые модели близки по времени применения к модели SABR. А модели Heston и Variance-Gamma на порядок медленнее, т.к. для нахождения волатильности с помощью этих моделей необходимо численно решать оптимизационную задачу 2.5.

Однако, при варьировании параметров опциона (дата исполнения, соотношение цены акции и страйка, безрисковой ставки и т.д.) модель SABR необходимо перекалибровывать перед обучением, поэтому время ее применения увеличивается до 9,37 милисекунд, что почти в 2 раза больше, чем время применения нейросетей, которые при таком изменении параметров переобучать не нужно.

8.3 Преимущества и недостатки нейросетей

Главным достоинством нейросетевых моделей является отсутствие необходимости в переобучении для применения к другим опционам, на которых модель не обучалась.

В сравнении с моделями Heston, Variance-Gamma и, в отдельных случаях (при смене параметров опциона), SABR нейросети выигрывают с точки зрения быстроты применения, т.к. их применение заключается в матричных произведениях и применении функций активации, что позволяет производить расчеты параллельно на графических чипах (GPU). А быстрота применения - важный аспект на современном фондовом рынке, где транзакции производятся за доли секунд.

Нейросети можно применять в разных сценариях: если необходимо ретроспективно оценить уровень волатильности активов по ценам производных от них опционов, то можно применить нейросеть A, если же необходимо оценить "новый"опцион, информации о цене которого еще нет, то в данном случае применима нейросеть B.

Главным недостатком нейросетей является потребность в бóльших объемах данных для обучения, чем у классических моделей.

9 Заключение

Задача ценообразования опционов важна для многих участников фондового рынка. Она необходима для точного хеджирования риска - благодаря этому финансовые институты, например, инвестиционные фонды банков и государств, могут обеспечивать высокую надежность своих вложений, что защищает многие слои населения от влияния финансовых кризисов и крахов.

Для более точного учета рыночной ситуации были разработаны модели SABR, Heston и Variance-Gamma, которые, в отличие от модели Блэка-Шоулза, учитывают изменение волатильности.

В дополнение к данным моделям в данной работе были предложены модели на основе нейросетей. На основе проделанных экспериментов можно сделать вывод, что нейросетевые модели дают сравнимое с классическими моделями качество восстановления поверхности волатильности. Такие модели обладают рядом достоинств перед классическими моделями, главные из которых это отсутствие необходимости переобучения модели при изменении параметров оцпиона и быстрота применения моделей за счет возможности параллелизации вычислений на графических ускорителях (GPU).

Список литературы

[1] F. Black, M. Scholes, The Pricing of Options and Corporate Liabilities, Journal of Political Economy 81, no. 3 (1973): 637–54.

[2] P. S. Hagan, D. Kumar, A. S. Lesniewski and D. E. Woodward, Managing smile risk, Wilmott Magazine, no. 1 (2002): 84-108.

[3] Steven L. Heston, (1993), A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options, Review of Financial Studies, 6, issue 2, p. 327-43.

[4] Madan, Dilip B., and Eugene Seneta. The Variance Gamma (V.G.) Model for Share Market Returns. The Journal of Business 63, no. 4 (1990): 511–24.

[5] Dilip B. Madan, Peter P. Carr, Eric C. Chang. The Variance Gamma Process and Option Pricing (1998), Review of Finance, Volume 2, Issue 1, 1998, Pages 79–105

[6] Elton A. Daal, Dilip B. Madan. An Empirical Examination of the Variance-Gamma Model for Foreign Currency Options, The Journal of Business, vol. 78, no. 6, 2005, pp. 2121–52. JSTOR,

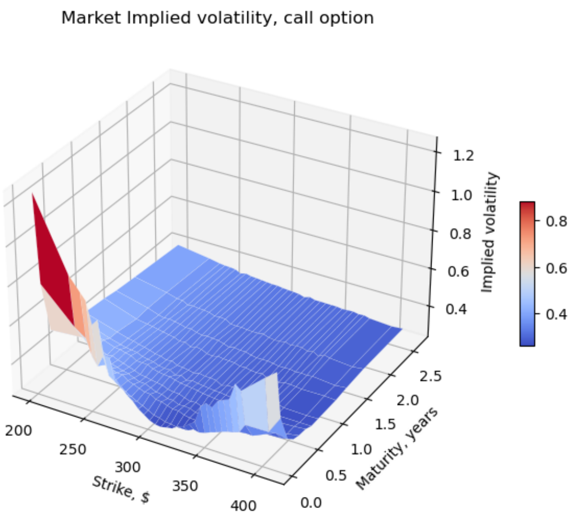
[7] Ran Aroussi, yfinance Python package, `https://pypi.org/project/yfinance/`

[8] R. H. Byrd, P. Lu and J. Nocedal. A Limited Memory Algorithm for Bound Constrained Optimization, (1995), SIAM Journal on Scientific and Statistical Computing , 16, 5, pp. 1190-1208.

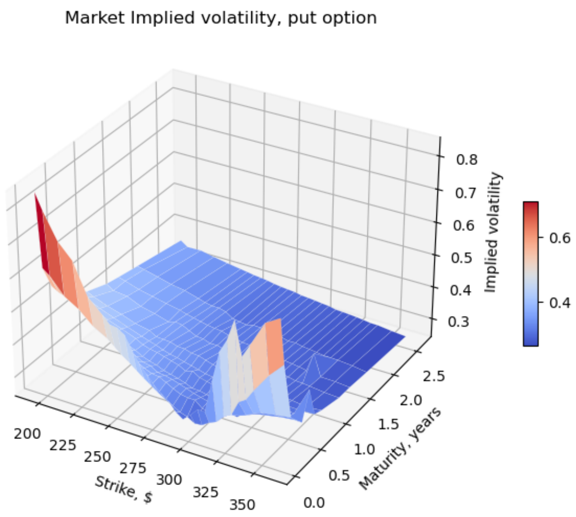
[9] Kingma, D. P., Ba, J. 2014. Adam: A Method for Stochastic Optimization. arXiv e-prints. doi:10.48550/arXiv.1412.6980

A Графики

A.1 Поверхности волатильности

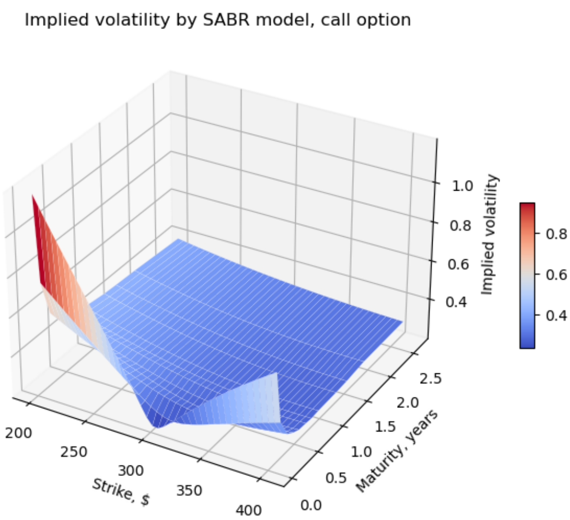


(a) Для оцпиона CALL

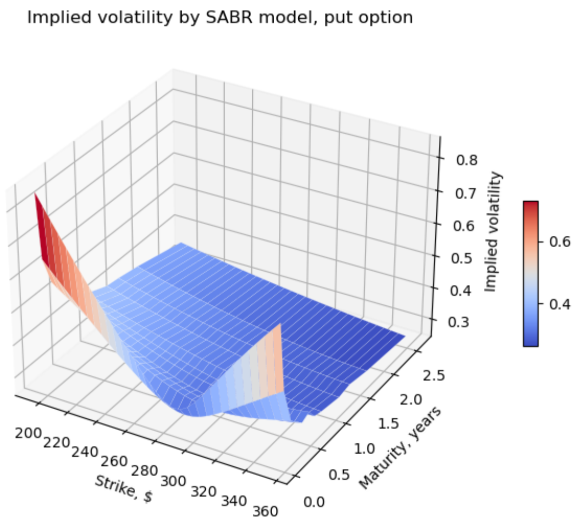


(b) Для оцпиона PUT

Рис. 3: Рычночные поверхности волатильности

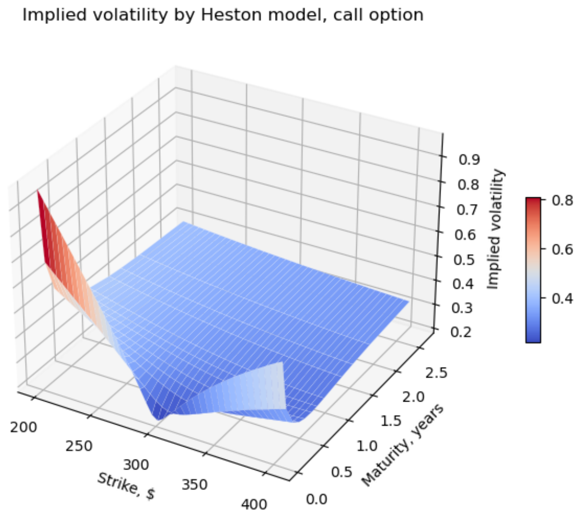


(a) Для оцпиона CALL

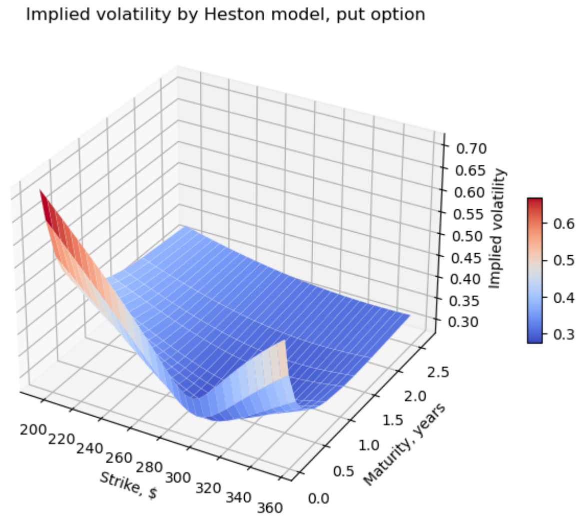


(b) Для оцпиона PUT

Рис. 4: Поверхности волатильности модели SABR

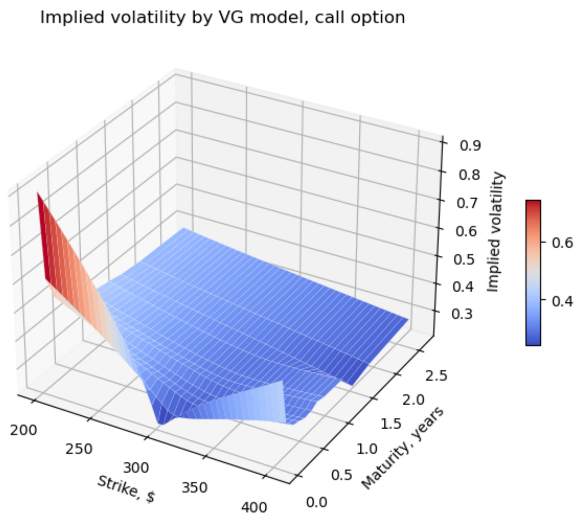


(a) Для опциона CALL

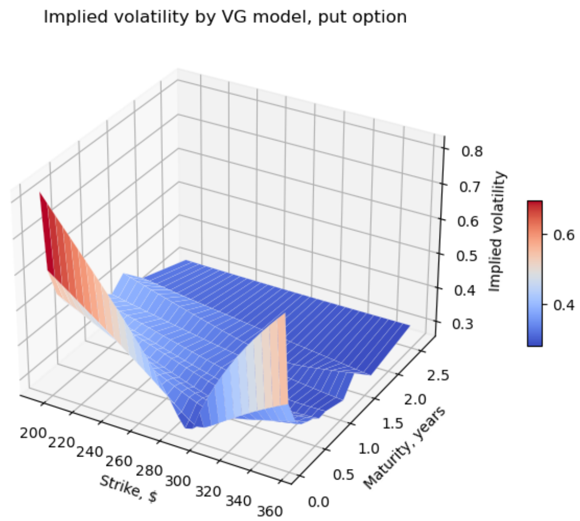


(b) Для опциона PUT

Рис. 5: Поверхности волатильности модели Heston

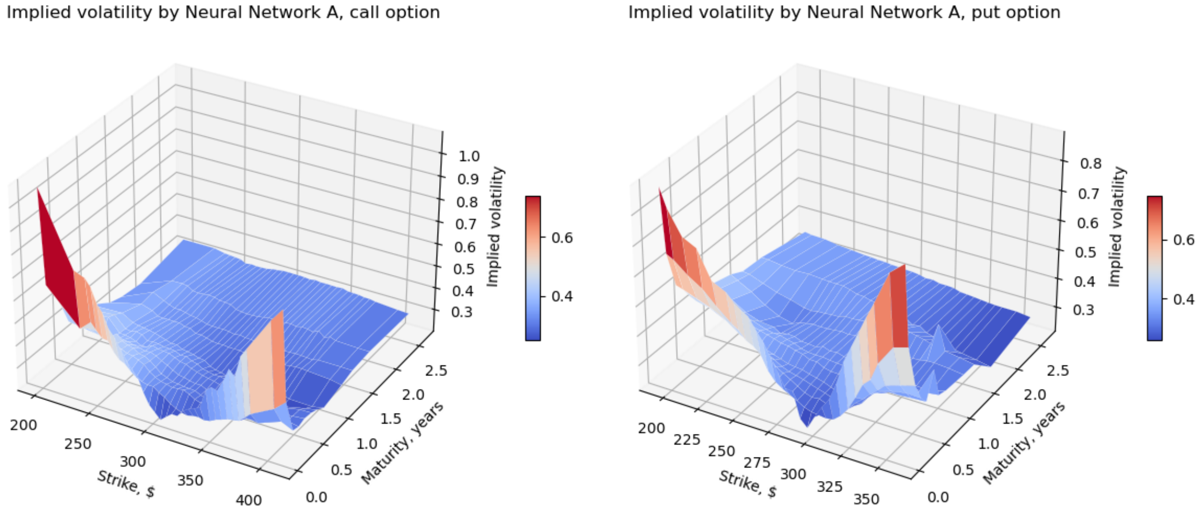


(a) Для опциона CALL



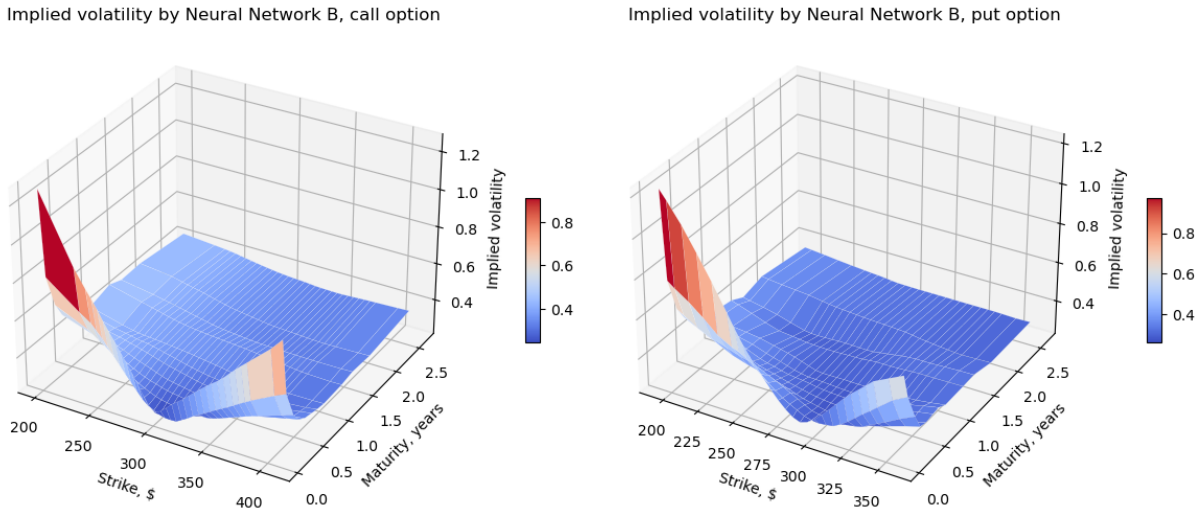
(b) Для опциона PUT

Рис. 6: Поверхности волатильности модели Variance-Gamma



(a) Для оцпиона CALL (b) Для оцпиона PUT

Рис. 7: Поверхности волатильности нейросети A



(a) Для оцпиона CALL (b) Для оцпиона PUT

Рис. 8: Поверхности волатильности нейросети B

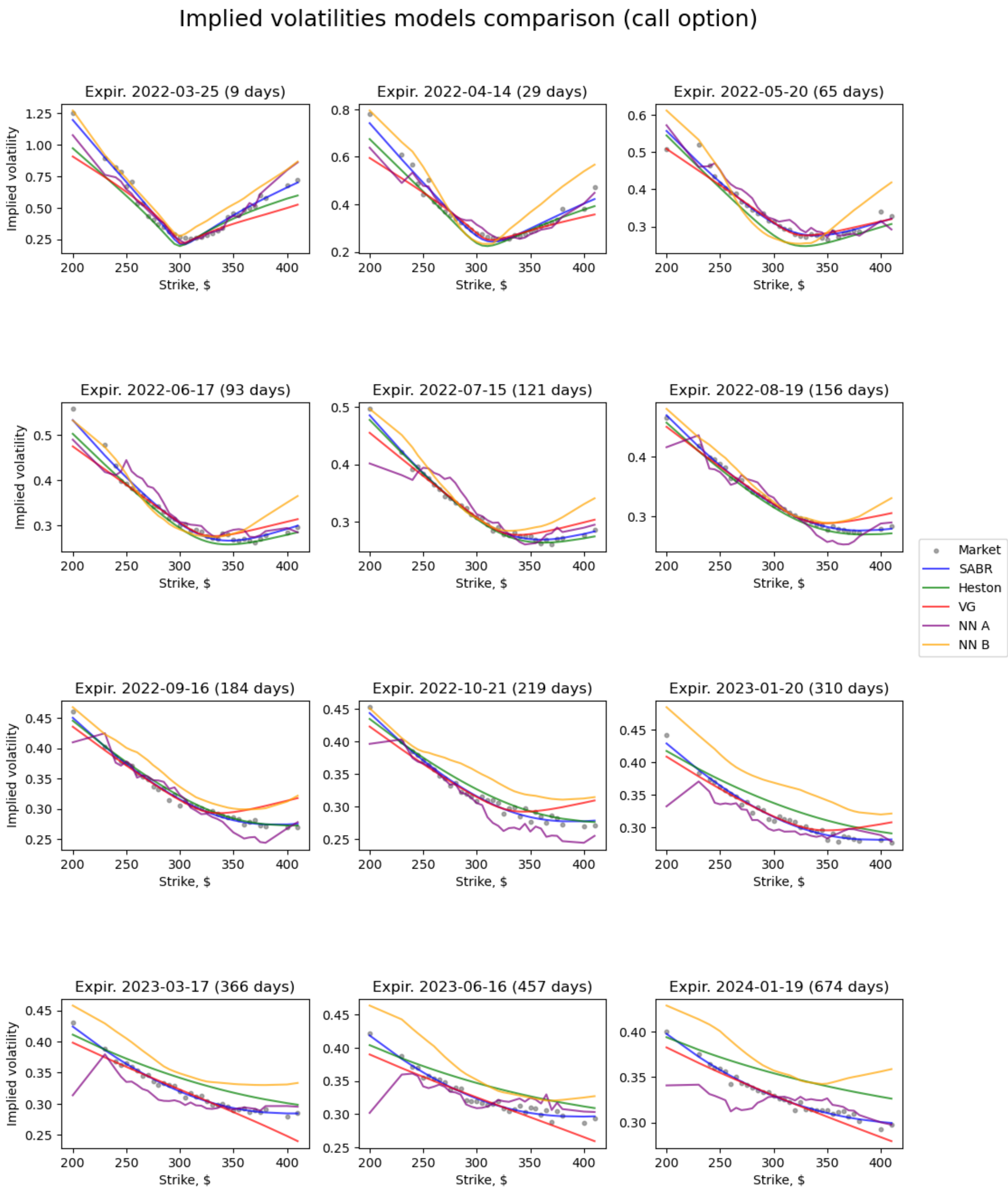


Рис. 9: Сравнение волатильностей для опциона CALL

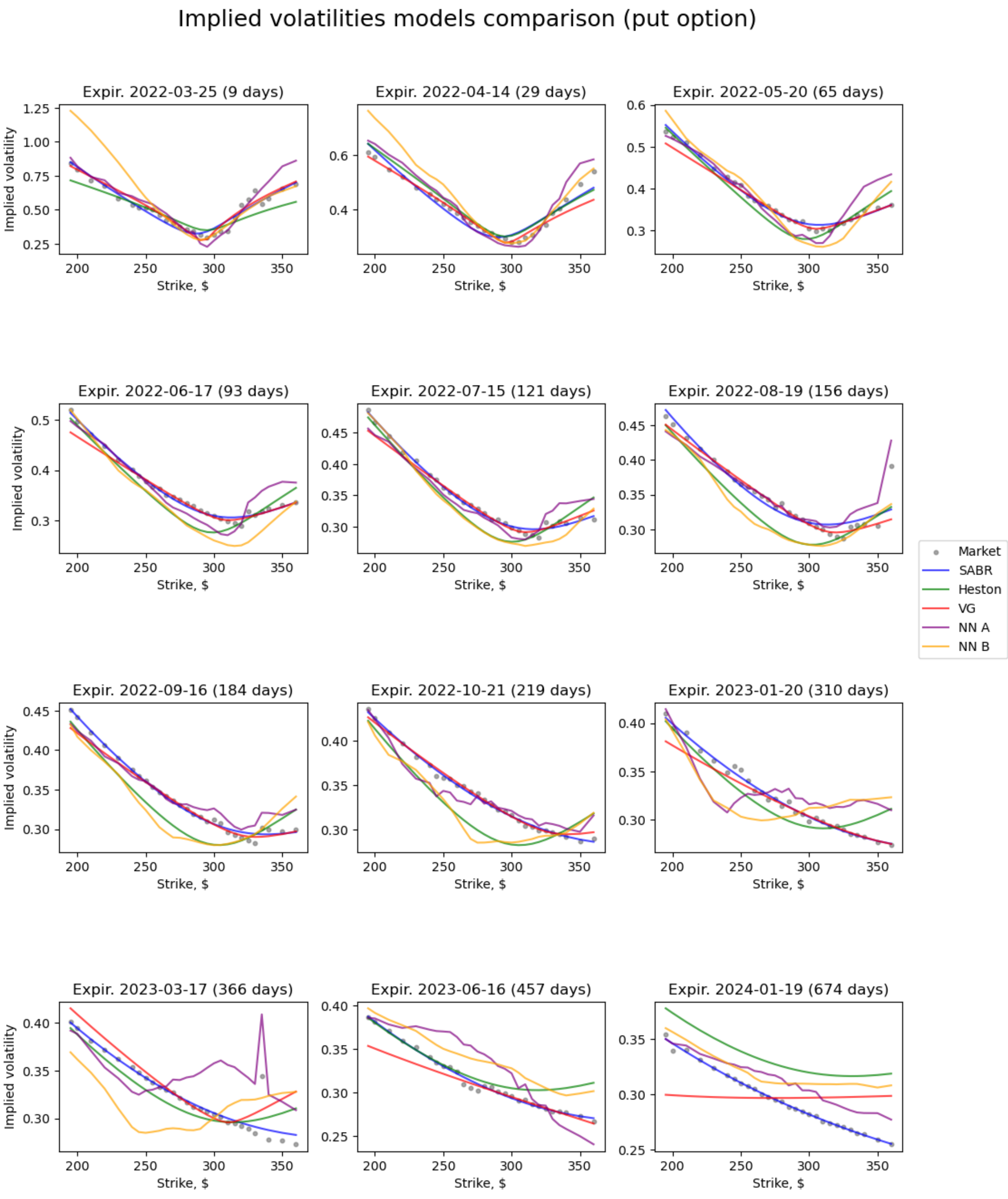
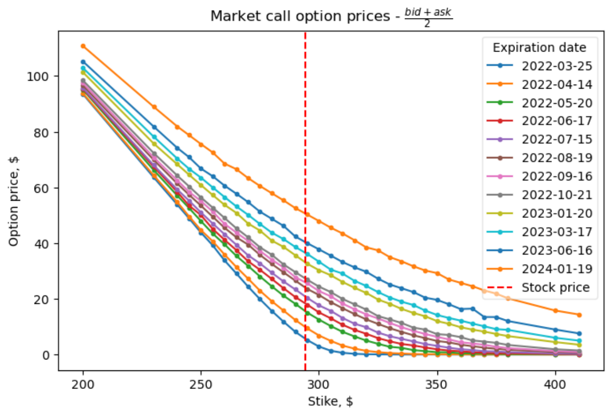
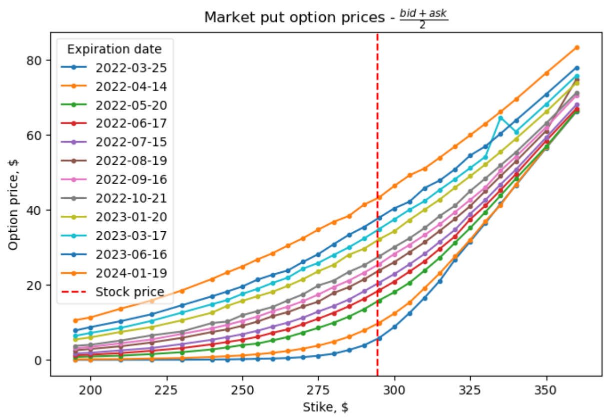


Рис. 10: Сравнение волатильностей для опциона PUT

A.2 Цены опционов

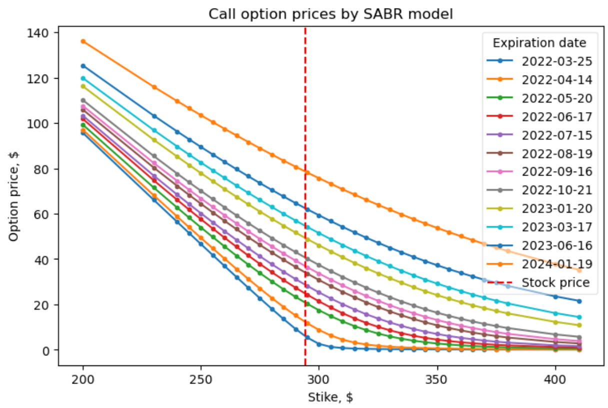


(a) Для опциона CALL

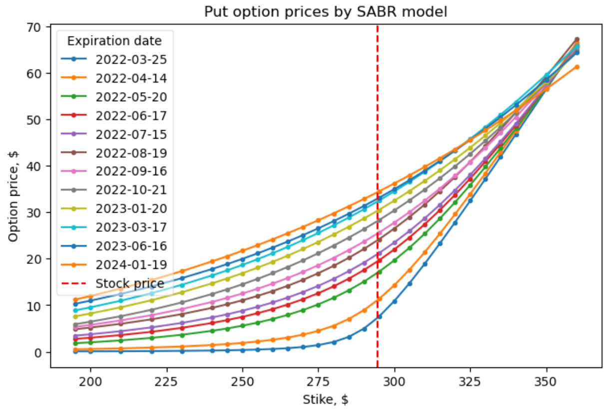


(b) Для опциона PUT

Рис. 11: Рыночные цены опционов

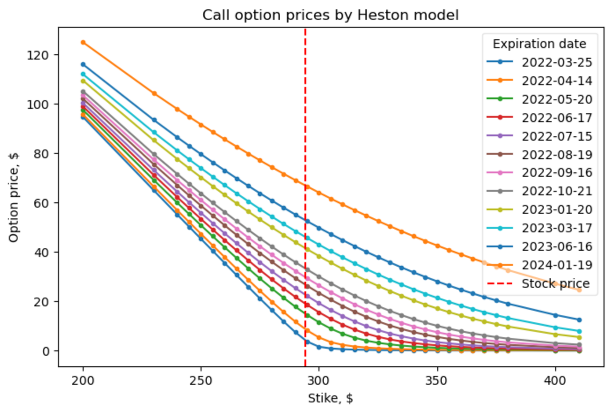


(a) Для опциона CALL

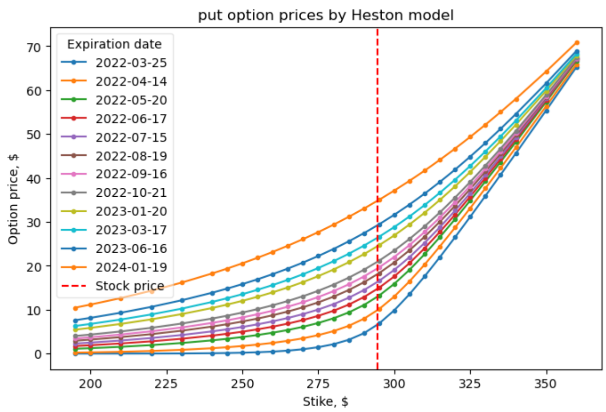


(b) Для опциона PUT

Рис. 12: Цены опционов модели SABR

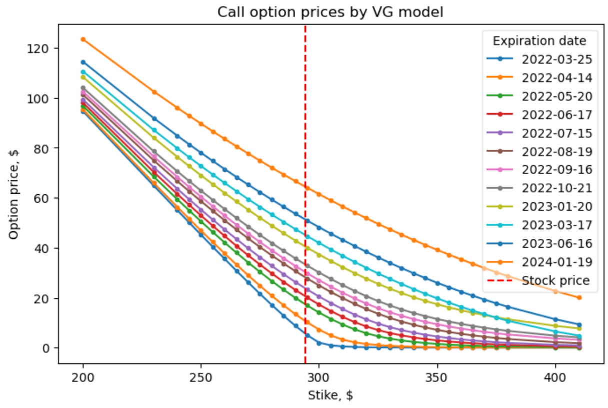


(a) Для опциона CALL

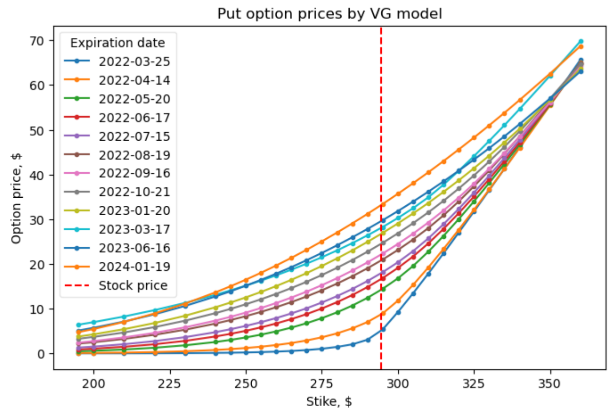


(b) Для опциона PUT

Рис. 13: Цены опционов модели Heston

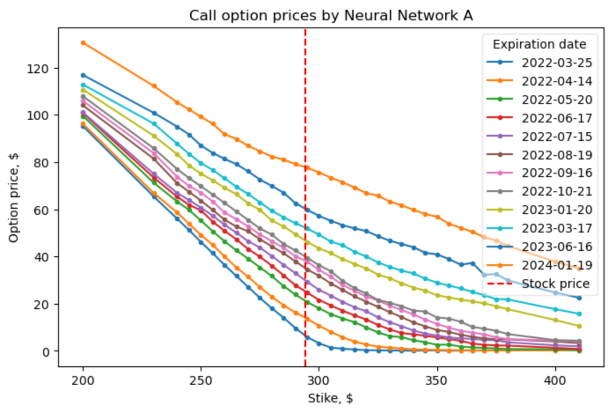


(a) Для оцпиона CALL

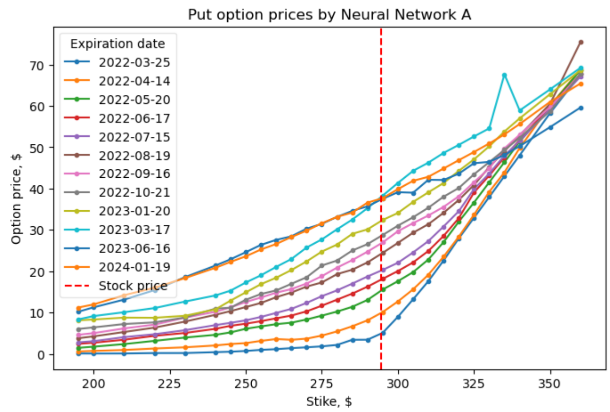


(b) Для оцпиона PUT

Рис. 14: Цены опционов модели Variance-Gamma

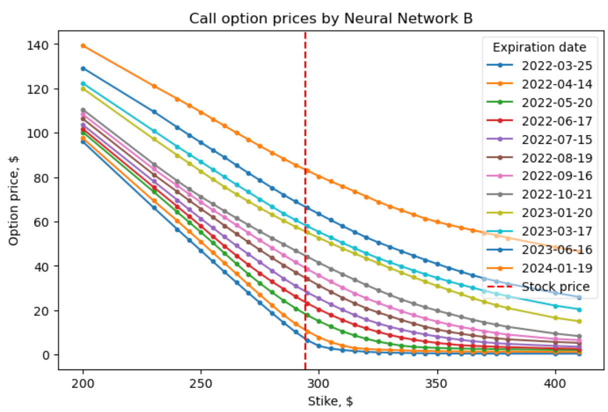


(a) Для оцпиона CALL

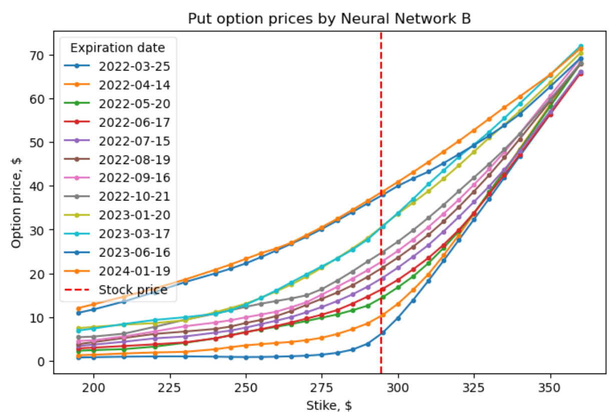


(b) Для оцпиона PUT

Рис. 15: Цены опционов нейросети A



(a) Для оцпиона CALL



(b) Для оцпиона PUT

Рис. 16: Цены опционов нейросети B

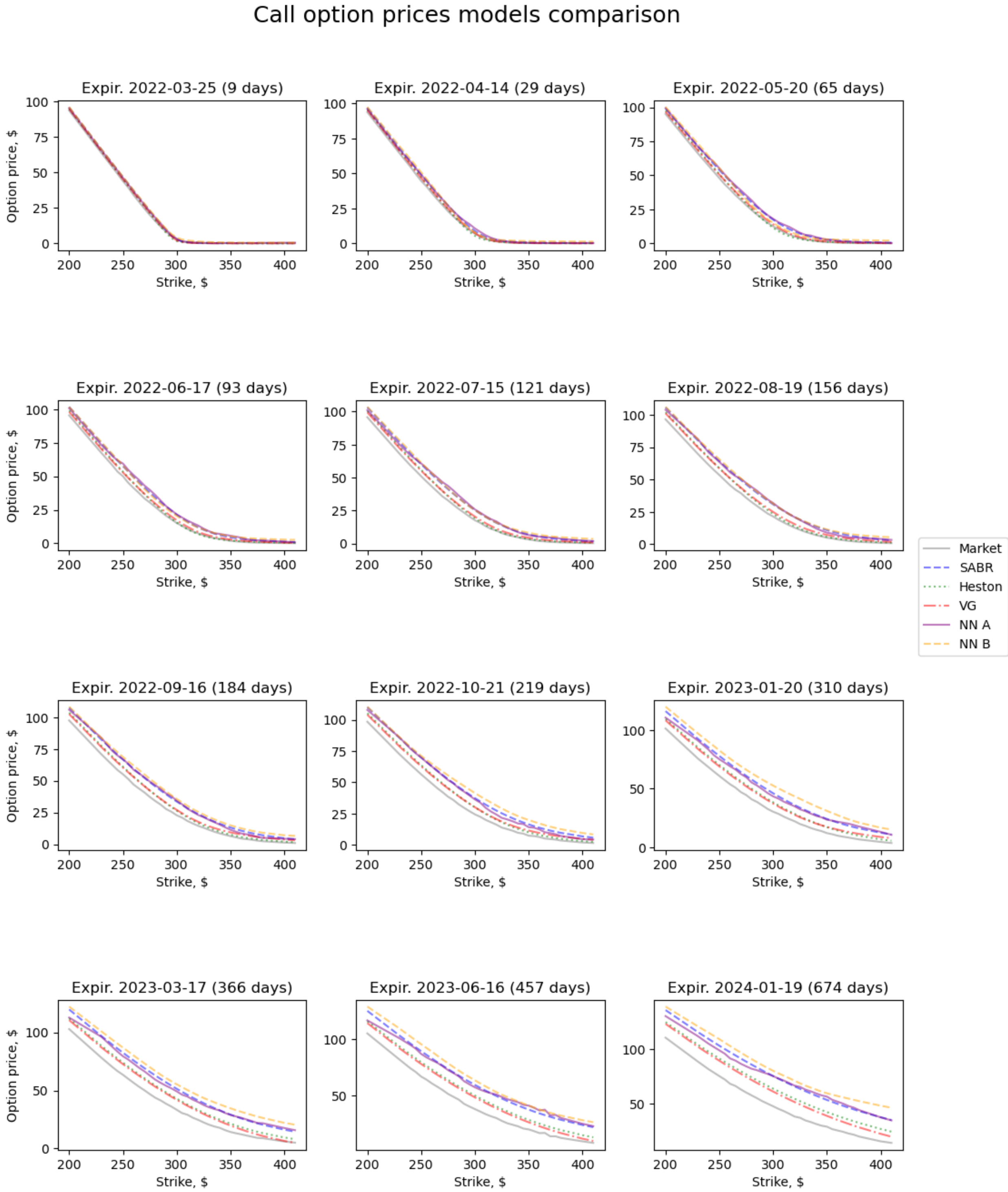


Рис. 17: Сравнение цен опционов CALL

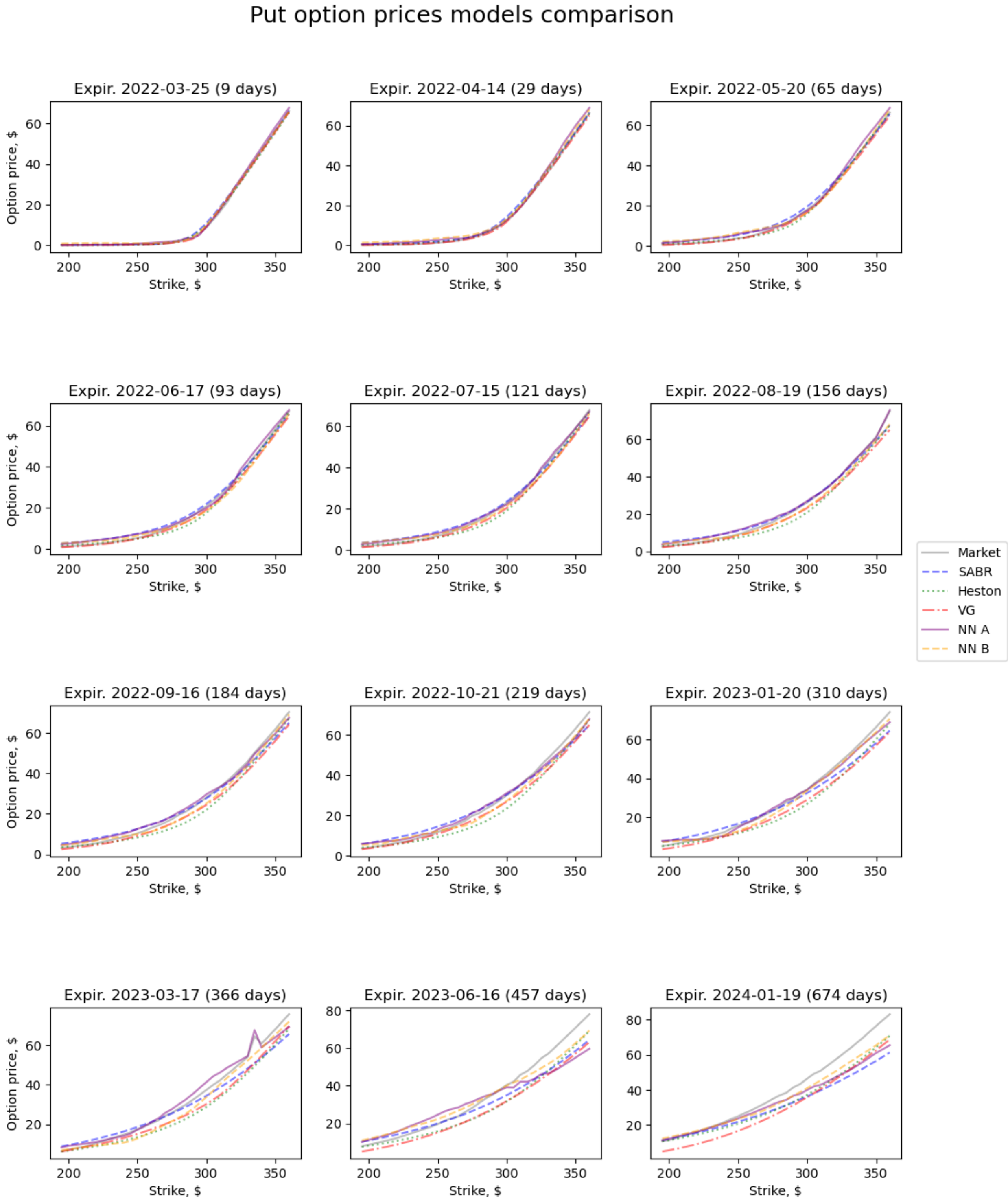


Рис. 18: Сравнение цен опционов PUT

B Программный код

```
class SABR:

    def __init__(self, stock_price=msft_stock_price):
        self.stock_price = stock_price
        self.params = []

    @staticmethod
    def volatility(strikes, stock_price, t, alpha, beta, rho, nu):
        f = stock_price
        K = strikes
        z = nu / alpha * (f*K)**((1-beta)/2) * np.log(f/K)
        x = np.log(((1-2*rho*z+z**2)**0.5 + z - rho)/(1-rho))
        first = alpha / ((f*K)**((1-beta)/2) * (1+(1-beta)**2 / 24 *
                (np.log(f/K))**2 + (1-beta)**4 / 1920 * (np.log(f/K))**4))
        second = z/x
        third = 1 + t*((1-beta)**2 / 24 *  alpha**2 / ((f*K)**(1-beta)) +
                (rho*beta*nu*alpha)/(4*(f*K)**((1-beta)/2)) + nu**2*(2-3*rho**2)/24)
        return first * second * third

    def fit(self, strikes, mrkt_vols, expiration:float, frac_ootm=1, frac_itm=1):

        def RSS_SABR(params, t, stock_price):
            sabr_vols = self.volatility(strikes, stock_price, t, *params)
            return np.sum(np.square(sabr_vols - mrkt_vols))

        strikes = np.array(strikes)
        mrkt_vols = np.array(mrkt_vols)

        if frac_ootm < 1 and frac_itm < 1:
            nearest = np.argmin(np.abs(strikes-self.stock_price))

            mask_relevant = np.repeat(False, len(strikes))
            rel_ootm = int(frac_ootm * len(strikes) / 2)
            rel_itm = int(frac_itm * len(strikes) / 2)
            mask_relevant[nearest-rel_itm : nearest+rel_ootm+1] = True

            strikes = strikes[mask_relevant]
            mrkt_vols = mrkt_vols[mask_relevant]

        # alpha, beta, rho, nu
        min_args = (expiration, self.stock_price)
        x0 = [0.001, 1, -0.999, 0.001]
        bounds = ((0.001, None), (1, 1), (-0.999, 0.999), (0.001, None))
        res = minimize(RSS_SABR, x0, min_args, bounds=bounds)
        self.params.append(res.x)

    def iv_surface(self, strikes, expirations, params=None):
        if params is None:
            params = self.params
        surface = np.zeros((len(expirations), len(strikes)), dtype=float)
        for i, (t,p) in enumerate(zip(expirations, params)):
            surface[i,:] = self.volatility(strikes, self.stock_price, t, *p)

        return surface

    def MSE_all(self, df_list, expirations, params=None):
        if params is None:
            params = self.params

```

```python
        sse = 0
        cnt = 0
        for i, (t,df,p) in enumerate(zip(expirations, df_list, params)):
            cnt += len(df)
            sabr_vols = self.volatility(df.strike, self.stock_price, t, *p)
            sse += np.sum(np.square(sabr_vols - df.impliedVolatility))

        return sse / cnt
```

Листинг 1: Модель SABR

```python
class Heston(ql.HestonModel):

    def __init__(
        self,
        init_params=(0.01, 1, 0.5, 0, 0.1),
        bounds=[(0,1), (0.01,15), (0.01,None), (-1,1), (0,1.0) ],
        q=0.0,
        r=0.05,
        evaluation_date='2022-03-16',
        spot=msft_stock_price
    ):
        self.init_params = init_params
        self.bounds = bounds
        theta, kappa, sigma, rho, v0 = self.init_params
        self.spot = spot
        self.calculation_date = ql.Date(evaluation_date, '%Y-%m-%d')
        day_count = ql.Actual365Fixed()
        self.calendar = ql.UnitedStates(1)
        self.yield_ts = ql.YieldTermStructureHandle(
            ql.FlatForward(self.calculation_date, r, day_count)
        )
        self.dividend_ts = ql.YieldTermStructureHandle(
            ql.FlatForward(self.calculation_date, q, day_count)
        )
        process = ql.HestonProcess(
            self.yield_ts, self.dividend_ts,
            ql.QuoteHandle(ql.SimpleQuote(spot)),
            v0, kappa, theta, sigma, rho
        )
        super().__init__(process)
        self.engine = ql.AnalyticHestonEngine(self)
        self.vol_surf = ql.HestonBlackVolSurface(ql.HestonModelHandle(self), self.engine.
    Gatheral)
        # self.build_helpers()


    def build_helpers(self, expirations, strikes, vols):
        maturities = [
            ql.Period(ql.Date(expir, '%Y-%m-%d') - self.calculation_date, ql.Days)
             for expir in expirations
        ]
        temp=[]
        for m,v in zip(maturities,vols):
            for i,s in enumerate(strikes):
                temp.append(
                    ql.HestonModelHelper(
                        m, self.calendar, self.spot, s,
                        ql.QuoteHandle(ql.SimpleQuote(v[i])),
                        self.yield_ts, self.dividend_ts
                    )
                )
```

```
        for x in temp: x.setPricingEngine(self.engine)
        self.helpers=temp
        self.loss= [x.calibrationError() for x in self.helpers]


    def f_cost(self, params, strikes, mrkt_vols, expirations, norm=False):
        self.setParams( ql.Array(list(params)) )
        self.build_helpers(expirations, strikes, mrkt_vols)
        if norm == True:
            loss = np.array(self.loss)
            mask = np.isinf(loss) | np.isnan(loss)
            self.loss = np.mean(np.square(loss[~mask]))
        return self.loss


    def fit(self, strikes, mrkt_vols, expirationsm, method='L-BFGS-B'):
        # self.build_heplers(expirations, strikes, mrkt_vols)
        if method == 'L-BFGS-B':
            min_args = (strikes, mrkt_vols, expirations, True)
            res = minimize(
                self.f_cost, self.init_params,
                args=min_args,
                method='L-BFGS-B',
                bounds=self.bounds
            )
            self.params = res.x
        elif method == 'LM':
            self.build_helpers(expirations, strikes, mrkt_vols)
            self.calibrate(
                self.helpers,
                ql.LevenbergMarquardt(1e-8, 1e-8, 1e-8),
                ql.EndCriteria(500, 300, 1.0e-8,1.0e-8, 1.0e-8)
            )


    def iv_surface(self, strikes, maturities):
        surface = np.zeros((len(maturities), len(strikes)), dtype=float)
        for i,t in enumerate(maturities):
            surface[i,:] = np.array([self.vol_surf.blackVol(t,s) for s in strikes])

        return surface

    def MSE_all(self, df_list, maturities):
        sse = 0
        cnt = 0
        for i, (t, df) in enumerate(zip(maturities, df_list)):
            cnt += len(df)
            heston_vols = np.array([self.vol_surf.blackVol(t,s) for s in df.strike])
            sse += np.sum(np.square(heston_vols - df.impliedVolatility))

        return sse / cnt
```

Листинг 2: Модель Heston

```
class Variance_Gamma:

    def __init__(self, evaluation_date='2022-03-16', stock_price=msft_stock_price, q=0.0, r
    =0.05):
        self.evaluation_date = evaluation_date
        ql.Settings.instance().evaluationDate = ql.Date(evaluation_date, '%Y-%m-%d')
        self.params = []
        self.stock_price = stock_price
```

```
        self.q = q
        self.r = r

    def get_option_obj(
        self,
        expiration_date : str,
    ):
        """Build `fftoptionlib` Option object

        Args:
            expiration_date (str): expiration date in format '%Y-%m-%d'
            q (float): dividend rate
            r (float): risk-free rate

        Returns:
            vanilla_option : `fftoptionlib` Option object
        """
        vanilla_option = fft.BasicOption()
        (vanilla_option.set_underlying_close_price(self.stock_price)
            .set_dividend(self.q)
            .set_maturity_date(expiration_date)
            .set_evaluation_date(self.evaluation_date)
            .set_zero_rate(self.r))

        return vanilla_option


    def get_vg_prices(
        self,
        theta : float,
        v : float,
        sigma : float,
        option_obj,
        strikes,
    ):
        """Calculates fair option prices using Variance-Gamma model

        Args:
            theta (float): VG model parameter - drift
            v (float): VG model parameter - variance rate
            sigma (float): VG model parameter - instantaneous volatility
            option_obj : `fftoptionlib` Option object
            strikes : strikes for which needs to calculate prices

        Returns:
            vg_prices : VG-calculated fair option prices
        """
        N=2**15
        d_u = 0.01
        alpha = 1
        ft_pricer = fft.FourierPricer(option_obj)
        (ft_pricer.set_pricing_engine(fft.FFTEngine(N, d_u, alpha, spline_order=2))
            .set_log_st_process(fft.VarianceGamma(theta=theta, v=v, sigma=sigma))
        )

        strikes = np.array(strikes)
        put_call = 'call'

        return ft_pricer.calc_price(strikes, put_call)


    def get_iv_params(
```

```
        self,
        expiration_date,
        strikes,
        sigma_0=0.2,
    ):
        calendar = ql.UnitedStates(1)
        exercise = ql.EuropeanExercise(ql.Date(expiration_date, '%Y-%m-%d'))

        S = ql.QuoteHandle(ql.SimpleQuote(self.stock_price))
        q = ql.YieldTermStructureHandle(ql.FlatForward(0, calendar, self.q, ql.Actual365Fixed()))
        r = ql.YieldTermStructureHandle(ql.FlatForward(0, calendar, self.r, ql.Actual365Fixed()))
        sigma = ql.BlackVolTermStructureHandle(
            ql.BlackConstantVol(0, calendar, sigma_0, ql.Actual365Fixed())
        )

        return exercise, S, q, r, sigma

    def calc_iv(
        self,
        opt_prices,
        strikes,
        iv_params,
    ):
        """Calculates implied volatilities from option prices

        Args:
            iv_params (tuple) : exercise, S, q, r, sigma
        """
        ivs = []
        for V, K in zip(opt_prices, strikes):
            payoff = ql.PlainVanillaPayoff(ql.Option.Call, K)
            option = ql.EuropeanOption(payoff, iv_params[0])
            process = ql.BlackScholesMertonProcess(*iv_params[1:])
            ivs.append(option.impliedVolatility(V, process))

        return np.array(ivs)


    def fit(
        self,
        expiration_date : str,
        strikes,
        mrkt_vols,
        frac_ootm : float=0.6,
        frac_itm : float=0.6,
    ):

        def RSS(par, mrkt_vols, option_obj, strikes, iv_params):
            theta = par[0]
            v = par[1]
            sigma = par[2]
            vg_prices = self.get_vg_prices(theta, v, sigma, option_obj, strikes)
            vg_pred_iv = self.calc_iv(vg_prices, strikes, iv_params)

            return np.sum(np.square(mrkt_vols - vg_pred_iv))

        strikes = np.array(strikes)
        mrkt_vols = np.array(mrkt_vols)

        option_obj = self.get_option_obj(expiration_date)
```

```python
        iv_params = self.get_iv_params(expiration_date, strikes)

        if frac_ootm < 1 and frac_itm < 1:
            nearest = np.argmin(np.abs(strikes-self.stock_price))
            mask_relevant = np.repeat(False, len(strikes))
            rel_ootm = int(frac_ootm * len(strikes) / 2)
            rel_itm = int(frac_itm * len(strikes) / 2)
            mask_relevant[nearest-rel_itm : nearest+rel_ootm+1] = True
        else:
            mask_relevant = np.repeat(True, len(strikes))

        min_args = (mrkt_vols[mask_relevant], option_obj, strikes[mask_relevant], iv_params)
        # theta, v, sigma
        x0 = [0, 0.2, 0.3]
        bounds = ((-1,None), (1e-3,None), (1e-3,None))
        res = minimize(RSS, x0, min_args, bounds=bounds, method='L-BFGS-B')
        self.params.append(res.x)
        # return res.x

    def iv_surface(self, strikes, expirations, params=None):
        if params is None:
            params = self.params

        surface = np.zeros((len(expirations), len(strikes)), dtype=float)
        for i, (expir,p) in enumerate(zip(expirations, params)):
            option_obj = self.get_option_obj(expir)
            iv_params = self.get_iv_params(expir, strikes)
            vg_prices = self.get_vg_prices(*p, option_obj, strikes)
            surface[i,:] = self.calc_iv(vg_prices, strikes, iv_params)

        return surface

    def MSE_all(self, df_list, expirations, params=None):
        if params is None:
            params = self.params

        sse = 0
        cnt = 0
        for i, (expir, df, p) in enumerate(zip(expirations, df_list, params)):
            cnt += len(df)

            option_obj = self.get_option_obj(expir)
            iv_params = self.get_iv_params(expir, df.strike)
            vg_prices = self.get_vg_prices(*p, option_obj, df.strike)
            vg_vols = self.calc_iv(vg_prices, df.strike, iv_params)

            sse += np.sum(np.square(vg_vols - df.impliedVolatility))

        return sse / cnt
```

Листинг 3: Модель Variance-Gamma

```python
class ANN(nn.Module):
    def __init__(self, struct, bn=False):
        super().__init__()
        layers = []
        for i,l in enumerate(struct[:-1]):
            if bn:
                layers.append(nn.BatchNorm1d(struct[i]))

            layers += [
                nn.Linear(struct[i], struct[i+1]),
```

```
                nn.ReLU()
            ]

        self.net = nn.Sequential(*layers[:-1])

    def forward(self, x):
        return self.net(x)

struct_ann = [len(columns), 256, 512, 256, 64, 1]
model = ANN(struct_ann, bn=True).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
criterion = nn.MSELoss()

def train(model, optimizer, criterion, dataloader_train, dataloader_test, num_epochs, noise=None, writer=None):
    logs = {
        'train_loss' : [],
        'test_loss' : []
    }
    model.train()
    for epoch in tqdm(range(num_epochs), desc='Epoch'):
        train_running_loss, grad_norm = 0.0, 0.0
        for iter_num, (X,y) in enumerate(dataloader_train):
            X = X.to(device)
            y = y.to(device).unsqueeze(1)
            if noise is not None:
                y = y + torch.rand_like(y) * noise
            optimizer.zero_grad()
            out = model(X)
            loss = criterion(out, y)
            train_running_loss += loss.item()
            loss.backward()

            optimizer.step()

            if writer is not None:
                writer.add_scalars("loss_iter", {'train' : loss}, iter_num + epoch * len(dataloader_train))

        logs['train_loss'].append(train_running_loss/len(dataloader_train))

        running_loss = 0.0
        for iter_num, (X,y) in enumerate(dataloader_test):
            with torch.no_grad():
                X = X.to(device)
                y = y.to(device).unsqueeze(1)
                out = model(X)
                loss = criterion(out, y)
                running_loss += loss.item()
                if writer is not None:
                    writer.add_scalars('loss_iter', {'test' : loss}, iter_num + epoch * len(dataloader_test))

        logs['test_loss'].append(running_loss/len(dataloader_test))

        if writer is not None:
            writer.add_scalars('loss_epoch', {
                'train': train_running_loss/len(dataloader_train),
                'test': running_loss/len(dataloader_test),
            }, epoch)

    if writer is not None:
```

```python
        writer.close()

    return logs

logs = train(model, optimizer, criterion, train_dl, test_dl, num_epochs=100)
```

Листинг 4: Нейросеть