НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2026 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Система калібрування моделі стохастичної волатильності Гестона на основі гібридного нейромережевого підходу»

Виконав:
студент IV курсу, групи КА-21
Шевцов Олександр Олександрович _____
Керівник:
Асистент кафедри ММСА.
Древаль Максим Михайлович _____
Консультант з економічного розділу:
Доцент кафедри ЕК, к.е.н.
Рощина Надія Василівна _____
Консультант з нормоконтролю:
Асистент кафедри ММСА, доктор філософії
технічних наук Канцедал Георгій Олегович _____
Рецензент:
Доцент кафедри СП, к.т.н.
Безносик Олександр Юрійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально–науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту
Шевцову Олександру Олександровичу

1. Тема роботи «Система калібрування моделі стохастичної волатильності Гестона на основі гібридного нейромережевого підходу», керівник роботи Древаль Максим Михайлович, асистент кафедри ММСА, затверджена наказом по університету від «27» травня 2026 р. №1944-с.
2. Термін подання студентом роботи «10» червня 2026 р.х
3. Вихідні дані до роботи: Історичні котирування європейських опціонів на індекс S&P 500 (OptionsDX, 2017-2022 рр.), часові ряди дохідностей казначейських облігацій США (FRED), результати калібрування чотирьох методів на синтетичному та ринковому тестових наборах.
4. Зміст роботи: 1. Дослідження предметної області: огляд поверхонь неявної волатильності та обмежень моделі Блека-Шоулза; класифікація моделей

стохастичної волатильності; огляд класичних методів калібрування афінних моделей у просторі неявної волатильності; огляд нейромережевих підходів до калібрування (амортизовані оцінювачі, точково-параметричні архітектури); огляд фізично–інформованих функцій втрат у фінансовому машинному навчанні. 2. Теоретичні основи моделі Гестона та алгоритмів калібрування: SDE–формулювання та характеристична функція моделі; метод косинусного розкладання Фанг-Остерлі для оцінювання європейських опціонів; калібрування методом Левенберга-Марквардта; архітектура згорткового нейромережевого калібратора; фізично–інформована функція втрат на основі диференційовного оцінювача. 3. Реалізація системи та експериментальне дослідження чотирьох методів калібрування на ринкових поверхнях S&P 500: побудова синтетичного датасету з апріорного розподілу параметрів; реалізація Методу 0 (еталон), Методу 1 (Левенберг-Марквардт), Методу 2 (нейромережа з MSE-втратами) та Методу 3 (нейромережа з фізично–інформованою функцією втрат); перебір вагового коефіцієнта λ; порівняння методів за IV–RMSE та швидкодією на синтетичних і ринкових даних 2010–2023; запровадження фактора розриву синтетика-ринок як діагностичного інструмента. 4. Функціонально-вартісний аналіз програмного продукту: обґрунтування функцій та системи параметрів; експертне ранжування й попарне порівняння; розрахунок коефіцієнта технічного рівня; економічний аналіз варіантів розробки та вибір кращого за техніко-економічним критерієм. 5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Функціонально-вартісний аналіз	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання 25.02.2026.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою роботи.	25.04.2026	Виконано
2	Підготовка першого розділу.	10.05.2026	Виконано
3	Підготовка другого розділу.	17.05.2026	Виконано
4	Реалізація диференційовного оцінювача та чотирьох калібраторів	19.05.2026	Виконано
5	Проведення обчислювальних експериментів	21.05.2026	Виконано
6	Підготовка третього розділу.	25.05.2026	Виконано
7	Підготовка економічного розділу.	01.06.2026	Виконано
8	Оформлення дипломної роботи.	06.06.2026	Виконано

Студент Олександр ШЕВЦОВ

Керівник Максим ДРЕВАЛЬ

РЕФЕРАТ

Дипломна робота: 103 с., 9 рис., 13 табл., 25 джерел, 2 додатки.

МОДЕЛЬ ГЕСТОНА, СТОХАСТИЧНА ВОЛАТИЛЬНІСТЬ, КАЛІБРУВАННЯ, ФІЗИЧНО-ІНФОРМОВАНІ НЕЙРОННІ МЕРЕЖІ, IV–RMSE, ОПЦІОНИ SPX.

Об’єкт дослідження – процес калібрування афінних моделей стохастичної волатильності на спостережуваних ринкових поверхнях опціонів. Предмет дослідження – гібридні нейромережеві методи калібрування моделі Гестона з фізично-інформованою компонентою функції втрат.

Мета роботи: розробити, реалізувати та порівняти чотири методи калібрування моделі Гестона, провести систематичний перебір вагового коефіцієнта фізично–інформованої компоненти функції втрат та запропонувати діагностику якості калібрування на ринкових даних.

Методи дослідження – методи фінансової математики (модель Гестона, формула Блека-Шоулза), методи чисельного інтегрування (косинусне розкладання Фанг-Остерлі), методи оптимізації (Левенберг-Марквардт), методи машинного навчання (згорткові нейронні мережі, фізично-інформовані нейронні мережі), методи статистичного оцінювання.

Розроблено програмний продукт мовою програмування Python з використанням бібліотек PyTorch, NumPy, SciPy. Реалізовано систему калібрування моделі Гестона з чотирма методами на єдиному протоколі IV-RMSE. Виокремлено три внески роботи: методологічний (роль вагового коефіцієнта λ у гібридній функції втрат), емпіричний (валідаційно-ринкова інверсія: ранжування λ за валідаційною та ринковою метриками декорельовані), діагностичний (фактор розриву синтетика-ринок $\approx$ 2,25 як нижня межа похибки для класу моделі Гестона).

ABSTRACT

Bachelor's thesis: 103 pages, 9 figures, 13 tables, 25 sources, 2 appendices.

HESTON MODEL, STOCHASTIC VOLATILITY, CALIBRATION, PHYSICS-INFORMED NEURAL NETWORKS, IV–RMSE, SPX OPTIONS.

Object of research – the process of calibrating affine stochastic volatility models on observed market option surfaces.

Subject of research – hybrid neural-network methods for Heston model calibration with a physics-informed loss component.

Aim of the work – to develop, implement and compare four methods of Heston model calibration, conduct a systematic sweep of the physics–informed loss–component weight coefficient, and propose a calibration–quality diagnostic for market data.

Research methods – financial mathematics methods (Heston model, Black–Scholes formula), numerical integration methods (Fang-Oosterlee COS expansion), optimization methods (Levenberg-Marquardt), machine learning methods (convolutional neural networks, physics-informed neural networks), and statistical estimation methods.

A software product was developed in Python programming language using PyTorch, NumPy, and SciPy libraries. A Heston model calibration system with four methods on a single IV-RMSE protocol was implemented. Three contributions were identified: methodological (the role of the weight coefficient λ in the hybrid loss function), empirical (validation-market inversion: rankings of λ under validation and market metrics are decorrelated), and diagnostic (synthetic-to-market gap factor $\approx$ 2.25 as a lower error bound for the Heston model class).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ATM – навколо-грошовий опціон (at-the-money) – опціон, ціна виконання якого дорівнює поточній ціні базового активу.

CIR – процес Кокса-Інгерсолла-Росса (Cox-Ingersoll-Ross), який описує еволюцію дисперсії у моделі Гестона.

CNN – згорткова нейронна мережа (convolutional neural network).

COS – метод косинусного розкладання Фанг-Остерлі (Fourier-cosine expansion) для оцінювання європейських опціонів через характеристичну функцію.

FRED – Federal Reserve Economic Data – публічна база макроекономічних часових рядів Федерального резерву США, що використовується як джерело кривих безризикових ставок.

ITM – опціон у грошах (in-the-money) – опціон, виконання якого зараз дає прибуток.

IV – неявна волатильність (implied volatility) – значення параметра волатильності, при якому ціна опціона у моделі Блека-Шоулза збігається зі спостережуваною ринковою.

LM – алгоритм Левенберга-Марквардта (Levenberg-Marquardt) розв'язання задач нелінійних найменших квадратів.

ML – машинне навчання (machine learning).

NN – нейронна мережа (neural network).

OTM – позагрошовий опціон (out-of-the-money) – опціон, виконання якого зараз було б збитковим.

PINN – фізично-інформована нейронна мережа (physics-informed neural network) – мережа, у функцію втрат якої входить компонента, що відображає узгодженість виходу з фізичним або прямомодельним оператором задачі.

RMSE – корінь з середньоквадратичної похибки (root mean square error).

SABR – модель стохастичної волатильності Хагана та інших (stochastic alpha-beta-rho).

SLV – стохастична локальна волатильність (stochastic local volatility).

SPX – індекс S&P 500 – біржовий індекс 500 найбільших публічних компаній США, на опціонах якого проводяться обчислювальні експерименти роботи.

TRF – варіант дзеркального відбиття у довірчій підобласті (trust-region-reflective) – модифікація алгоритму Левенберга-Марквардта з обмеженнями на параметри, реалізована у функції scipy.optimize.least_squares.

СДР – стохастичне диференціальне рівняння.

Call-опціон(Європейський) – фінансовий інструмент, що дає право купити базовий актив за ціною K через строк T.

Put-опціон(Європейський) – фінансовий інструмент, що дає право продати базовий актив за ціною K через строк T.

Vega (ν) – чутливість ціни опціона до зміни неявної волатильності.

Грошовість – відношення ціни виконання K до поточної ціни базового активу S.

ВСТУП

Класична модель Блека-Шоулза, побудована на припущенні сталої волатильності базового активу, виявилася непридатною для узгодженого опису ринкових поверхонь неявної волатильності. Феномени «усмішки» та «косої усмішки», які послідовно фіксуються на індексних опціонах від кінця 1980-х років, безпосередньо суперечать припущенням про константність і вимагають моделей, у яких волатильність є випадковим процесом. Серед таких моделей особливе місце посідає модель Гестона [1], запропонована у 1993 році, яка вирізняється двома практичними перевагами: наявністю замкнутого виразу для характеристичної функції розподілу логарифма ціни та обмеженою кількістю параметрів – лише п'ятьма, які підлягають калібруванню до спостережуваних ринкових котирувань.

Калібрування моделі Гестона полягає у пошуку п'яти параметрів – швидкості повернення дисперсії до середнього, довгострокового рівня дисперсії, волатильності, миттєвої кореляції шоків та початкової дисперсії – за умови мінімізації відхилення розрахованих модельних цін опціонів від ринкових. У переважній більшості практичних реалізацій задача формулюється у просторі неявної волатильності з vega-зваженими нев'язками, а її розв'язання виконують методом нелінійних найменших квадратів – найчастіше у варіанті Левенберга-Марквардта [2]. Цей підхід надає високу точність, утім потребує багаторазового виклику оцінювача опціонів усередині оптимізаційного циклу, що накладає обчислювальні витрати порядку десятків мілісекунд на одну поверхню.

За останнє десятиліття у літературі сформувався альтернативний підхід – нейромережеві калібратори [3, 4]. Ідея полягає у тренуванні мережі, що зіставляє поверхню неявної волатильності з вектором параметрів моделі, на синтетичній вибірці, згенерованій за допомогою прямого виклику оцінювача з відомими параметрами. Після одноразового навчання виклик мережі коштує

менше за мілісекунду, тобто прискорення порівняно з ітеративним оптимізатором сягає двох порядків. Однак нейромережеві калібратори страждають від проблеми зміщення розподілу: розподіл параметрів у синтетичній вибірці лише наближено відповідає реальному, тому мережа, що демонструє низьку похибку на синтетичній валідаційній підвибірці, не обов'язково повторює цей результат на ринкових поверхнях.

Один із способів пом'якшення цього розриву полягає у доповненні основної функції втрат компонентою, обчисленою через оцінювач опціонів, так званою фізично-інформованою функцією втрат [5]. У літературі цей прийом представлений під назвами «втрата калібрування» [4] або «втрата неявних цін» [6]; вибір конкретного значення вагового коефіцієнта фізичного складника досліджується у роботах останніх двох років [7, 8] як з адаптивними, так і з фіксованими схемами.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

У розділі проведено аналітичний огляд літератури за темою роботи. Описано емпіричне явище поверхні неявної волатильності та обмеження класичної моделі Блека-Шоулза. Систематизовано основні моделі стохастичної волатильності з акцентом на моделі Гестона як стандарті для індексних опціонів. Окремі підрозділи присвячено двом класам методів калібрування: класичним оптимізаційним та нейромережевим, а також концепції фізично-інформованих функцій втрат як способу пом'якшення розподілового зсуву між синтетичною й ринковою вибірками. У підсумку окреслено конкретну конфігурацію, у якій поточна робота розвиває цю лінію досліджень.

1.1 Поверхні неявної волатильності та обмеження моделі Блека-Шоулза

Класична модель оцінювання європейських опціонів, побудована Блеком та Шоулзом у 1973 році [9], спирається на припущення про геометричний броунівський рух ціни базового активу зі сталою волатильністю. У межах цих припущень одержано замкнений вираз ціни опціона як функцію п'яти аргументів – поточної ціни активу, ціни виконання, строку до виконання, безризикової ставки та волатильності. Розв'язання цієї формули відносно волатильності при фіксованих інших аргументах і спостережуваній ринковій ціні визначає неявну волатильність – таке значення, при якому модельна ціна збігається з ринковою. За умови коректності припущень моделі неявна волатильність для всіх контрактів на один і той самий базовий актив на одну й ту саму торгову дату становила б єдине число.

На практиці спостерігається стійка залежність неявної волатильності від ціни виконання та строку – поверхня неявної волатильності.

Перші систематичні свідчення про неконстантність неявної волатильності зафіксовано на ринку індексних опціонів S&P 500 одразу після обвалу 19 жовтня 1987 року: спостерігалося стійке формування косої усмішки – неявна волатильність позагрошових опціонів “put” помітно перевищувала неявну волатильність позагрошових опціонів “call” на той самий строк. Цей феномен у літературі трактують як непрямий показник того, що ринкові учасники приписують негативним рухам індексу вищу ймовірність, ніж передбачає симетричний логнормальний розподіл. Аналогічно зафіксовано залежність форми поверхні від строку – зростання навколо-грошової волатильності зі збільшенням строку для коротких контрактів та її повільне спадання для довгих, а також зміни всієї поверхні у часі, особливо у періоди ринкового напруження.

Емпіричні спостереження за дохідностями активів утворюють узгоджений набір регулярностей, несумісних з геометричним броунівським рухом зі сталою волатильністю: важкі хвости розподілу дохідностей у короткостроковому горизонті, кластеризація волатильності, ефект важеля (від'ємна кореляція між дохідностями та реалізованою волатильністю), повільне згасання автокореляції абсолютних величин дохідностей. Жодну з цих властивостей класична модель не відтворює: при сталій волатильності розподіл логарифмічних дохідностей є строго нормальним без важких хвостів, а кореляція між дохідностями та зміною волатильності тотожно нульова – самої зміни волатильності у моделі немає.

Розширенням моделі є відмова від припущення сталої волатильності. У літературі сформовано два основні підходи. Локальна волатильність – детермінована функція ціни та часу, яка точно відтворює дану поверхню неявної волатильності у момент калібрування, проте має проблеми зі стабільністю динаміки: майбутні поверхні, передбачені моделлю, можуть істотно відрізнятися від спостережуваних. Стохастична волатильність –

підхід, у якому волатильність є окремим випадковим процесом. Він є гнучкішим стосовно динамічних властивостей: процеси повернення до середнього, від'ємна кореляція між шоками ціни і волатильності та інші регулярності закладаються безпосередньо у структурі рівнянь. У роботі розглядається другий підхід.

Поверхня неявної волатильності – спостережувана статистика ринку, а не модельний об'єкт. Будь-яка модель оцінювання опціонів має пояснювати як ринкова сітка котирувань породжує цю поверхню. Якщо модель є коректною, її параметри, відкалібровані до поверхні, відтворюють її з прийнятною точністю; рівень розбіжності між ринковою та модельною поверхнями у просторі неявної волатильності (IV-RMSE) використовується далі у роботі як стандартна міра якості калібрування.

1.2 Огляд моделей стохастичної волатильності

Перші моделі стохастичної волатильності, запропоновані наприкінці 1980-х – на початку 1990-х років, спиралися на припущення про незалежність шоків ціни та волатильності. До цього класу належить модель Галла-Уайта [10], в якій волатильність описано геометричним броунівським рухом. Моделі такого типу відтворюють деякі емпіричні регулярності, зокрема кластеризацію волатильності, проте неузгоджено пояснюють асиметрію косої усмішки – ефект важеля у них відсутній за конструкцією. Через це моделі цього типу не закріпились як галузевий стандарт.

У 1993 році Гестоном [1] вперше побудовано модель стохастичної волатильності з корельованими шоками та одержано замкнений аналітичний вираз для характеристичної функції розподілу логарифма ціни активу. Стохастичну еволюцію умовної дисперсії задано рівнянням Кокса-Інгерсолла-Росса з поверненням до довгострокового рівня θ, швидкістю κ, волатильністю волатильності σ та шоком, корельованим із шоком ціни через параметр ρ.

Разом із початковим значенням ціни активу S_0 модель містить п'ять параметрів: $\Theta = (\kappa, \theta, \sigma_v, \rho, v_0)$ – мінімальну кількість, необхідну для опису ключових емпіричних регулярностей. Замкнена форма характеристичної функції зробила можливим практичне оцінювання опціонів через метод швидкого перетворення Фур'є (Карра-Мадана [11]) або косинусного розкладання (Фанг-Остерлі [12]); без неї оцінювання довелося б виконувати моделюванням Монте-Карло, що несумісне з ітеративним калібруванням у режимі реального часу.

Модель Гестона належить до класу афінних моделей за термінологією Даффі-Пана-Сінглтона [13], для яких характеристична функція має експоненційно-афінну форму у початкових значеннях факторів. До цього ж класу належать розширення моделі через включення стрибків ціни (для уточнення короткострокового сегмента поверхні), а також версії з немарковською дисперсією. Кожне таке розширення додає параметри і ускладнює калібрування.

Серед альтернативних формулювань – модель SABR (стохастичний альфа-бета-ро) Хагана та співавторів 2002 року [14], побудована як параметричне розкладання поверхні неявної волатильності у замкненій формі; вона закріпилася як стандарт для процентних та валютних опціонів, але на індексних застосовується рідше через специфіку трактування скосу. Окремим активним напрямком є моделі грубої волатильності (rough volatility) – у роботі Гатерала-Жайссона-Розенбаума 2018 року [15] показано, що емпіричні часові ряди реалізованої волатильності найкраще описуються процесом з показником Хьорста приблизно 0.1, що значно нижче за 0.5 (показник стандартного броунівського руху). Моделі цього класу добре відтворюють короткостроковий сегмент поверхні, проте втрачають замкнену форму характеристичної функції; кожне оцінювання опціона у них потребує моделювання, що ускладнює класичне калібрування і робить нейромережеві методи (підрозділ 1.4) альтернативою з підвищеною привабливістю.

У контексті поточної роботи модель Гестона обрано на таких підставах. По-перше, вона є галузевим стандартом для індексних опціонів, основою для більшості еталонних порівнянь у літературі з калібрування [16]. По-друге, замкнена форма характеристичної функції забезпечує швидке та диференційовне оцінювання через метод косинусного розкладання, що необхідно для побудови гібридного нейромережевого підходу. По-третє, обмежена кількість параметрів (п'ять) дозволяє згортковій нейромережі помірного розміру оцінити обернене відображення з поверхні у параметри без надмірного ризику перенавчання. Нарешті, обмеження моделі (відсутність стрибків, лише дифузійна динаміка) добре документовані у літературі та можуть бути кількісно оцінені – це створює основу для введення відношення розриву (підрозділ 1.5).

1.3 Класичні методи калібрування афінних моделей

Калібрування афінної моделі стохастичної волатильності формулюється як задача нелінійних найменших квадратів: знайти вектор параметрів, при якому модельні ціни (або неявні волатильності) найкраще відтворюють сітку спостережуваних котирувань. Формулювання у просторі неявної волатильності, поширене у сучасній літературі [16], має кілька числових переваг над роботою у ціновому просторі: воно нормалізує нев'язки до співвимірних величин незалежно від грошовості та строку, безпосередньо корелює зі сприйняттям якості калібрування у роботі трейдерів та зменшує чутливість до помилок інверсії Блека-Шоулза для глибоко позагрошових контрактів. Стандартним вибором ваги при нев'язках є vega-зважування (похідна ціни Блека-Шоулза за σ), що збільшує внесок АТМ-сегмента і зменшує внесок на межах поверхні, де точність ринкових котирувань нижча.

Серед чисельних методів розв'язання задачі нелінійних найменших квадратів метод Левенберга-Марквардта [17, 2] є одним з найбільш

поширених у фінансовому контексті. Він поєднує дві стратегії – метод Гаусса-Ньютона (на ділянках, де лінеаризація цільової функції добре наближає її поведінку) та метод найшвидшого спуску (поблизу складних точок або в умовах поганої обумовленості Якобіана) – через параметр демпфування μ, який автоматично адаптується між ітераціями. Бренч, Коулмен і Лі [18] запропонували варіант методу під назвою «дзеркальне відбиття у довірчій підобласті» (trust-region-reflective) – модифікацію, яка вміє утримувати параметри в межах заданих діапазонів (наприклад, кореляцію в інтервалі [–1, 1] або волатильність строго додатньою). Цей варіант реалізовано у функції scipy.optimize.least_squares бібліотеки SciPy як метод 'trf'.

Близькою до проблематики поточної роботи є праця Мразека та Поспішила 2017 року [16], у якій проведено систематичне дослідження калібрування моделі Гестона на історичних даних індексу DAX. Автори зіставили кілька схем зважування у просторі неявної волатильності (uniform, vega, спред купівля-продаж) та кілька оптимізаційних алгоритмів (lsqnonlin у різних конфігураціях, симульований відпал, диференціальна еволюція). Серед висновків: vega-зважування покращує якість калібрування на краях поверхні; стратегія з 5-10 випадкових початкових наближень необхідна для надійного уникнення локальних мінімумів у дослідницькій конфігурації; ініціалізація з результатів калібрування попереднього торгового дня прискорює робочий сценарій без помітної втрати точності.

Альтернативні підходи до глобальної оптимізації, що досліджувалися у літературі – диференціальна еволюція, симульований відпал, генетичні алгоритми – забезпечують вищу впевненість у знаходженні глобального мінімуму за рахунок значно більших обчислювальних витрат (від десятків секунд до хвилин на одну поверхню). У щоденному оперативному використанні, де перерахунок виконується по цілій книзі контрактів на десятки індексів у режимі реального часу, ці методи є непрактичними. Стандартний компроміс – локальний Метод LM з початковим наближенням, який обрано як еталон у плануваній роботі.

Принциповим обмеженням класичного підходу є обчислювальна складність. Один цикл LM–калібрування виконує десятки ітерацій, кожна з яких обчислює Якобіан нев'язок за п'ятьма параметрами методом скінченних різниць або аналітично; кожне обчислення Якобіана у свою чергу потребує викликів оцінювача, де n – кількість активних котирувань у поверхні (зазвичай 50–80). Сумарно одне калібрування – десятки мілісекунд на одне ядро процесора. Для розгортання у режимі оперативної перекалібровки на десятки індексів одночасно це створює істотне обмеження пропускної здатності та мотивує пошук швидших альтернатив.

1.4 Нейромережеві підходи до калібрування

Заміна ітеративного оптимізатора нейромережевим калібратором розглядається у літературі щонайменше з 2016 року [3], коли Ернандес запропонував використати багатошаровий персептрон для прямого відображення ринкових даних у вектор параметрів моделі (у першій роботі – модель Галла-Уайта для процентних ставок на навколо–грошових своп–опціонах). Підхід є природним розв'язком оберненої задачі: якщо існує диференційований оператор, який відображає Θ у поверхню, то задача калібрування полягає у пошуку Θ за заданою поверхнею. Замість того, щоб обчислювати це обернення ітеративно для кожної нової поверхні, нейромережа наближає функцію один раз і застосовується прямо. Дані для навчання генеруються синтетично – через прямий виклик тієї самої моделі, що калібрується. Це створює необмежене джерело пар з відомими істинними параметрами.

Згорткові архітектури для обробки поверхні неявної волатильності з'явилися дещо пізніше і закріпилися як стандарт у задачах калібрування афінних моделей. Фундаментальною роботою, що сформулювала сучасну методологію калібрування Гестона та споріднених моделей, є праця Байера та

Стемпера 2018 року [4] про глибоке калібрування моделей грубої волатильності. У ній детально описано протокол: генерація синтетичних пар через емпіричний апріорний розподіл параметрів; побудова згорткової архітектури, що обробляє поверхню як двовимірне зображення (грошовість x строк); стандартизація параметрів через z- або log-перетворення; тренування з ранньою зупинкою за валідаційною метрикою. Одним з ключових експериментальних висновків є те, що нейромережа на синтетичній валідаційній підмножині досягає RMSE у просторі неявної волатильності на порядок нижчого, ніж класичний LM, проте на реальних ринкових даних результати лише конкурентоспроможні, а не однозначно кращі.

Подальший розвиток підходу простежується у роботах Лю, Боровиха, Гржелака та Остерлі 2019 року [19] та Хорвата, Мугурузи й Томаса 2021 року [6]. У першій запропоновано загальну схему нейромережевого калібрування фінансових моделей; обговорено питання якості синтетичного апріорного розподілу та вибору архітектури під різні моделі стохастичної волатильності. У другій наведено ґрунтовний емпіричний аналіз нейромережевого калібрування моделей грубої волатильності; показано, що глибокі мережі здатні наближати функції оцінювання rough Bergomi та rough Heston з точністю порядку 10^{-3} у просторі неявної волатильності на синтетичній підмножині. Окрему увагу автори приділяють швидкісним характеристикам: один прохід прямого поширення через мережу коштує менш ніж 1 мс, тоді як симуляція Монте–Карло для цих моделей – десятки секунд.

Поряд із перевагами – однопрохідне обчислення, паралелізованість за поверхнями, відсутність проблеми локальних мінімумів – нейромережеві калібратори мають кілька обмежень. Перше з них – розподіловий зсув між синтетичною та ринковою вибірками. Емпіричний апріорний розподіл, з якого генерується тренувальна вибірка, лише наближено відповідає тому, що спостерігається на ринку: реальні поверхні можуть мати структури (зокрема, злам у косій усмішці після стресових днів), які не породжуються самою моделлю Гестона. У результаті нейромережа, що демонструє низьку похибку

на синтетичній валідаційній підмножині, може давати значно гіршу якість на реальних поверхнях. Друге обмеження – деградація якості у стресових ринкових режимах, для яких частка тренувальної вибірки може бути малою або відсутньою. Третє – нелінійна неіндентифікованість: різні комбінації параметрів Гестона можуть породжувати близькі поверхні (зокрема, у площині $\kappa-\theta$), і мережа, тренована з функцією втрат за параметрами, може не виявляти цю невизначеність у якості на поверхні.

Період 2024-2025 років позначений активним розвитком цього напрямку. Аміці, Морандотті та Чжан [20] запропонували глибокі диференціальні мережі (Deep Differential Networks) – архітектуру, що навчається не лише відображенню параметрів Гестона у ціну опціона, а й частинним похідним за параметрами; роботу опубліковано в *Decisions in Economics and Finance* у 2025 році. Сонг 2025 року [8] запропонував наскрізну архітектуру на основі тривимірної згорткової мережі у поєднанні з трансформером (3D CNN–Transformer) для прямого відображення панелі опціонів у параметри Гестона з диференційовним оцінювачем на основі швидкого перетворення Фур'є та шаром штрафу за порушення відсутності арбітражу, реалізовано та оцінено на S&P 500 і EURO STOXX 50. Хошісаші, Фелан та Барукка опублікували серію робіт [7, 21] про адаптивне зважування компонент функції втрат у фізично–інформованому калібруванні поверхні неявної волатильності S&P 500 (WamOL) та стохастичної локальної волатильності на даних валютного ринку (PDC–PINN). Шьон і Вальтер [22] у *Journal of Risk* провели порівняльний аналіз стохастичних моделей волатильності на ринку S&P 500, що охоплює як звичайні ринкові умови, так і період стресу пандемії COVID–19.

Серед цих робіт особливе значення для поточного дослідження мають [7], [8] та [22]. У WamOL Хошісаші-Фелан-Барукки [7] представлено самоадаптивну схему автоматичного балансування для зважування кількох компонент функції втрат (нев'язка диференціального рівняння та диференціальні нерівності відсутності арбітражу), що динамічно

оновлюються через градієнтні кроки протягом тренування. Сонг [8] для тих самих індексних опціонів проводить фіксований перебір значень λ для ваги штрафу за порушення відсутності арбітражу (вибір кількох дискретних значень із подальшим порівнянням якості). Шьон-Вальтер [22] проводять порівняльний аналіз базових моделей стохастичної волатильності (Гестон, моделі грубої волатильності, SLV) і виявляють підвищені флуктуації параметрів Гестона під час пандемії COVID-19.

Поточна дипломна робота розвиває цю лінію досліджень у конкретній конфігурації, що дещо відрізняється від наявних робіт за трьома вимірами. По-перше, ваговий коефіцієнт λ у роботі застосовано не до штрафу за порушення відсутності арбітражу (як у Сонга [8]) і не до адаптивно балансованої нев'язки диференціального рівняння (як у WamOL [7]), а до прямомодельно–узгодженого переоцінювання у сенсі «втрата калібрування» Байера–Стемпера [4]; досліджується фіксований перебір λ за чотирма значеннями. По-друге, для кількісної інтерпретації розподілового зсуву між синтетикою та ринковими даними використовується просте відношення СКП (співвідношення RMSE на ринкових поверхнях до RMSE на синтетичній валідаційній підмножині), що не є новинкою у загальному ML-контексті, але дозволяє інтерпретовно порівнювати чотири методи. По-третє, виконується стратифікація результатів за п'ятьма ринковими режимами 2017–2023 років з акцентом на порівняльну таблицю усіх чотирьох методів – у форматі, ближчому до [22], ніж до аналізів одного методу [7, 8].

Серед інших суміжних робіт, що пропонують нейромережеві підходи до моделювання нестаціонарних фінансових процесів, варто згадати генеративну модель Недашківської та Андросова [23] на основі варіаційного автокодувальника, рекурентних мереж GRU і нейронних звичайних диференціальних рівнянь – методологічно близька задача побудови нейромережевого наближення для нестаціонарного процесу, що природно описується стохастичним диференціальним рівнянням. У роботі Романенка та Канцедала [24] досліджуються інструменти моделювання динаміки

криптовалютних ринків – суміжна предметна область з тими самими методичними викликами розподілового зсуву та режимної стратифікації. Багатокритеріальна модель кредитного скорингу Писарчука та інших [25] ілюструє ширший контекст застосування підходів на основі даних до обернених фінансових задач.

1.5 Фізично-інформовані функції втрат у фінансовому машинному навчанні

Концепція фізично-інформованих нейронних мереж (Physics-Informed Neural Networks, PINN) у строгому формулюванні запропонована Райссі, Пердікарісом та Карніадакісом у 2019 році [5]. Базова ідея: якщо мережа має наближати функцію, що задовольняє диференціальне рівняння виду$\mathcal{N}[u] = 0$ (де $\mathcal{N}$– диференціальний оператор), то у функцію втрат додається штрафний член, який вимагає, щоб мережа задовольняла це рівняння на наборі контрольних точок:

$$L_{total} = L_{data} + \lambda \cdot \|\mathcal{N}[u_{NN}]\|^2,$$

де L_{data} – стандартна помилка узгодження з даними;

а другий член – нев'язка диференціального рівняння.

Ваговий коефіцієнт контролює, наскільки сильно мережа має слідувати фізичному закону у порівнянні з точковим узгодженням даних. У PINN-літературі прийнято, що мале λ діє як м'яка регуляризація, що сприяє узагальненню без жорсткого зміщення, тоді як велике λ перетворює задачу на майже чистий розв'язок диференціального рівняння [5, 7].

Перенесення цієї парадигми у фінансове машинне навчання потребує уточнення. У задачі калібрування афінної моделі строге диференціальне рівняння Гестона не входить безпосередньо у функцію втрат – нев'язки виду $\|\mathcal{N}[u_{NN}]\|^2$ відсутня, оскільки нейромережа оцінює параметри, а не функцію самого процесу. Натомість у літературі калібрування афінних моделей

використовується інша стратегія, методологічно споріднена з PINN. Її ідея – додати до функції втрат компоненту, що обчислюється через прямий оператор моделі (тобто оцінювач): з оцінених мережею параметрів обчислюється реконструйована поверхня, яка порівнюється з вхідною ринковою поверхнею. У літературі ця компонента називається «втрата калібрування» (Байер–Стемпер [4]), «втрата неявних цін» (Хорват та інші [6]), «втрата узгодженості» або «втрата прямомодельної узгодженості». У роботі прийнято термін «фізично–інформована функція втрат» у ширшому сенсі, в якому він використовується у фінансовому машинному навчанні: будь-яка компонента функції втрат, що походить від прямого оператора моделі.

Технічною передумовою такого підходу є диференційовність оцінювача. Для класичних реалізацій оцінювача Гестона методом косинусного розкладання, що історично писалися у середовищах без автоматичного диференціювання (C++, стандартний NumPy), пропускання градієнту крізь оцінювач було нетривіальною задачею. Розвиток середовищ автоматичного диференціювання (PyTorch, JAX) у середині 2010-х років зняв цю перепону, що уможливило поширення гібридних архітектур з компонентами прямомодельної узгодженості.

Питання вибору вагового коефіцієнта, що балансує параметричну та компоненту переоцінювання у гібридній функції втрат, є активним напрямом досліджень. Сучасна література пропонує два основні підходи. Перший – адаптивне автоматичне балансування під час тренування: λ–коефіцієнти оновлюються за градієнтною схемою на кожній епосі. Цей підхід реалізовано у WamOL Хошісаші–Фелан–Барукки [7] для шести компонент функції втрат у PINN-калібруванні поверхні неявної волатильності SPX та у PDC–PINN [21] для стохастичної локальної волатильності з дворівневою–оптимізацією. Другий підхід – фіксована сітка λ: набуває кілька дискретних значень, для кожного з яких проводиться окреме тренування з подальшим порівнянням якості. Цей підхід застосовано Сонгом [8] для ваги штрафу за порушення

відсутності арбітражу у задачі калібрування Гестона на SPX–даних за сіткою $\lambda \in \{0,1,10,100,1000\}$.

У поточній роботі застосовується другий, методологічно простіший підхід – фіксований перебір λ, але до іншої компоненти функції втрат порівняно з [8]: до втрати прямомодельної узгодженості (переоцінювання) у сенсі Байера-Стемпера [4]. Вибір на користь дискретного перебору, а не адаптивної схеми, обумовлений двома обставинами. По-перше, інтерпретовність: дискретна сітка дозволяє безпосередньо спостерігати залежність якості на ринкових даних від відносної ваги фізичної компоненти, без плутанини з динамікою адаптивної схеми. По-друге, контрольованість експерименту: усі чотири значення λ тренуються з однаковими гіперпараметрами оптимізатора, без впливу адаптивної процедури, що дозволяє чітко атрибутувати спостережувані ефекти ваговому коефіцієнту. Це робить підхід методологічно простішим за WamOL, але цілеспрямованим для навчальних цілей дипломної роботи.

1.6 Висновки до розділу 1

У розділі систематизовано стан досліджень у предметній області. Емпіричне явище неконстантної поверхні неявної волатильності, несумісне з моделлю Блека-Шоулза, мотивує використання моделей стохастичної волатильності. Серед них модель Гестона займає позицію стандарту для індексних опціонів – завдяки замкнутій формі характеристичної функції, обмеженій кількості параметрів і добре документованим обмеженням.

Класичне калібрування моделі Гестона методом Левенберга-Марквардта у варіанті TRF з vega-зваженими нев'язками є усталеною практикою, ретельно дослідженою у працях Мразека-Поспішила та інших. Принципове обмеження – обчислювальна складність порядку десятків мілісекунд на одну поверхню мотивує пошук швидших альтернатив.

Нейромережеві калібратори, започатковані працями Ернандеса, Байера-Стемпера, Лю та Хорвата у 2016-2021 роках, забезпечують прискорення на два порядки за рахунок одноразового тренування на синтетичних даних. Період 2024-2025 років позначений якісним розвитком цього напрямку: WamOL Хошісаші–Фелан–Барукки з адаптивним балансуванням компонент функції втрат, наскрізна архітектура Сонга з диференційовним оцінювачем на основі перетворення Фур'є та явним перебором значень λ, глибокі диференціальні мережі Аміці–Морандотті–Чжана, а також порівняльний аналіз Шьон-Вальтера у *Journal of Risk*. Поле активно досліджується, а основні методологічні питання вже значною мірою опрацьовані.

Фізично-інформовані функції втрат у формі компоненти прямомодельної узгодженості є природним механізмом пом'якшення розподілового зсуву між синтетикою та ринковими даними через введення компоненти, що оцінюється у просторі неявної волатильності через диференційовний оцінювач. Сучасні роботи досліджують як адаптивне λ-балансування ([7, 21]), так і фіксований перебір окремих значень [8], однак для різних компонент функції втрат (нев'язка диференціального рівняння, штраф за порушення відсутності арбітражу, обмеження–нерівності на похідні), а не для компонента переоцінювання у класичному форматі Байера–Стемпера.

У плановій конфігурації перебір λ застосовано до компоненти переоцінювання (на відміну від адаптивних схем [7, 21] та конфігурації Сонга [8]), а порівняльна таблиця чотирьох методів – “тривіальний” метод, LM–метод, чистий CNN–калібратор та гібридний CNN з компонентою переоцінювання – будується за п'ятьма ринковими режимами на поверхнях SPX 2017–2023 років. У другому розділі побудовано теоретичну основу методу.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ОСНОВИ МОДЕЛІ ТА АЛГОРИТМІВ КАЛІБРУВАННЯ

Модель Гестона [1] описує спільну еволюцію ціни базового активу та її миттєвої дисперсії у нейтральній до ризику мірі системою двох стохастичних диференціальних рівнянь:

$$\mathrm{d}S_t = (r - q)S_t\,dt + \sqrt{v_t}S_t\,\mathrm{d}W_t^S \tag{2.1}$$

$$\mathrm{d}v_t = \kappa(\theta - v_t)\,dt + \sigma\sqrt{v_t}\,\mathrm{d}W_t^v \tag{2.2}$$

де $S_t -$ позначає ціну базового активу;

v_t – миттєву (умовну) дисперсію;

r – безризикову ставку;

q – ставку дивідендів;

W_t^S та W_t^v – корельовані вінерівські процеси, що задовольняють співвідношення:

$$\mathrm{d}\langle W^S, W^v\rangle_t = \rho\,dt. \tag{2.3}$$

Модель містить п'ять параметрів, які підлягають калібруванню: швидкість повернення дисперсії до середнього κ, довгостроковий рівень дисперсії θ, волатильність волатильності σ, миттєву кореляцію між шоками ціни і дисперсії ρ та початкову дисперсію v_0. Параметри формують вектор $\Theta = (\kappa, \theta, \sigma, \rho, v_0) \in R^5$, що є об'єктом оцінювання у задачі калібрування. У роботі S_0 – ціна закриття індексу SPX на торговий день; $r(T)$ – лінійно інтерпольована крива Treasury constant maturity ставок з даних FRED; $q = 0$ за стандартним припущенням для індексних опціонів. Допустима область: $\kappa > 0$, $\theta > 0$, $\sigma > 0$, $\rho \in (-1, 1)$, $v_0 > 0$.

Процес v_t описується рівнянням Кокса-Інгерсолла-Росса(CIR). За виконання умови Феллера:

$$2\kappa\theta > \sigma^2 \tag{2.4}$$

траєкторії залишаються строго додатними з імовірністю 1. У ринкових у котируваннях індексів акцій, зокрема S&P 500, ця умова стандартно порушується для типових калібрувальних значень параметрів, оскільки ринкова поверхня вимагає високого σ для відтворення ексцесу косої усмішки коротких строків. У плануваному дослідженні умова Феллера не накладається ані під час навчання нейромережі, ані під час класичного калібрування – таке рішення відповідає галузевій практиці [16].

Модель Гестона належить до класу афінних моделей за термінологією Даффі-Пана-Сінглтона [13]: характеристична функція логарифма ціни активу до строку погашення T має експоненційно–афінну форму у початковій дисперсії v_0 та log-ціні $x = ln\, S_0$. Запишемо її у вигляді:

$$\varphi(u; T, \Theta) = exp\{C(u,T) + D(u,T)v_0 + iu(ln\, S_0 + (r - q)T)\} \qquad (2.5)$$

де комплекснозначні функції C і D задаються виразами:

$$D(u,T) = \frac{b - d}{\sigma^2} \cdot \frac{1 - e^{-dT}}{1 - ge^{-dT}} \qquad (2.6)$$

$$C(u,T) = \frac{\kappa\theta}{\sigma^2} \cdot \left[(b - d)T - 2\, ln\left(\frac{1 - ge^{-dT}}{1 - g}\right)\right] \qquad (2.7)$$

де $b = \kappa - i\rho\sigma u$ – зміщений коефіцієнт дрейфу;

$d = \sqrt{b^2 + \sigma^2(iu + u^2)}$ – дискримінант характеристичного рівняння;

$g = \frac{b - d}{b + d}$ – відношення характеристичних коренів.

Формулювання коефіцієнтів C і D через різницю $(b - d)$ у чисельнику, а не через суму $(b + d)$, відповідає варіанту Little Trap [12]: воно уникає перетину гілок комплексного логарифма, що виникає у класичному формулюванні Гестона при великих T, та забезпечує числову стабільність на всьому діапазоні строків до п'яти років.

Замкнутий вираз для характеристичної функції є ключовим обчислювальним фактом, що робить калібрування моделі практично можливим: оцінювання опціонів зводиться до одноразового обчислення

скінченої суми за частотами (метод косинусного розкладання, підрозділ 2.2) без симуляції Монте-Карло на кожному кроці оптимізатора.

2.2 Метод косинусного розкладання для оцінювання європейських опціонів

Серед чисельних методів оцінювання опціонів через характеристичну функцію – методу швидкого перетворення Фур'є Карра-Мадана [11], методу Льюїса, методу напівпрямого квадратного розв'язання – у роботі обрано метод косинусного розкладання (COS-метод) Фанг та Остерлі [12]. Цей вибір обумовлений трьома обставинами: експоненційна швидкість збіжності за кількістю членів розкладання для гладких характеристичних функцій; відсутність необхідності обчислювати невласний інтеграл та проблеми вибору контуру інтегрування; сумісність з автодиференціюванням у середовищі PyTorch, що необхідно для побудови гібридної функції втрат (підрозділ 2.5).

Метод базується на тому, що щільність розподілу логарифма ціни активу на скінченному інтервалі $[a, b]$ може бути розкладена у ряд Фур'є за косинусами:

$$f(y) \approx \sum_{k=0}^{N-1}{}' F_k \cos\left(k\pi \frac{y-a}{b-a}\right) \tag{2.8}$$

у якій штрих $\sum'$ означає, що нульовий коефіцієнт F_0 береться з вагою $\frac{1}{2}$. Коефіцієнти F_k обчислюються через дійсні частини характеристичної функції на дискретних частотах:

$$F_k = \frac{2}{b-a} \cdot \mathrm{Re}\left\{\varphi\left(\frac{k\pi}{b-a}; T\right) \cdot exp\left(-ik\pi \frac{a}{b-a}\right)\right\} \tag{2.9}$$

Оскільки виплата європейського опціона залежить тільки від кінцевого значення активу, її очікувана величина у нейтральній до ризику мірі обчислюється як інтеграл від виплати, перемноженої на щільність. Підставляння розкладання (2.8) зводить інтеграл до скінченної суми:

$$V(t=0) = e^{-rT} \sum_{k=0}^{N-1} F_k V_k \,, \tag{2.10}$$

у якій коефіцієнти V_k – інтеграли виплати опціона за косинусами; для стандартних put/call-виплат вони обчислюються аналітично через елементарні функції.

Межі інтегрування обираються через кумулянти процесу: $[a, b] = \left[c_1 - L\sqrt{c_2 + \sqrt{c_4}}, c_1 + L\sqrt{c_2 + \sqrt{c_4}}\right]$, де c_n – кумулянти n-го порядку розподілу логарифма ціни, а L – константа, що визначає запас на хвости розподілу. У реалізації планується використати $L = 10$, $N = 128$, що достатньо для покриття хвостів розподілу на горизонтах до п'яти років без втрати точності оцінювання.

2.3 Калібрування методом Левенберга-Марквардта у просторі неявної волатильності

Класичне калібрування моделі Гестона до спостережуваних ринкових котирувань формулюється як задача нелінійних найменших квадратів. Нехай задано множину з n опціонних контрактів $\left\{K_i, T_i, \sigma_i^{\text{market}}\right\}_{i=1}$ (де K_i – ціна виконання, T_i – строк, σ_i^{market} – спостережувана неявна волатильність). Цільову функцію формують у просторі неявної волатильності з vega–зваженими нев'язками [16]:

$$r_i(\Theta) = \nu_i \cdot \left(\sigma_i^{\text{model}}(\Theta; K_i, T_i) - \sigma_i^{\text{market}}\right), \tag{2.11}$$

$$\Theta^* = \operatorname{argmin}_{\Theta \in \Omega} \frac{1}{2} \sum_{i=1}^{n} r_i(\Theta)^2 \,, \tag{2.12}$$

де $\sigma_i^{\text{model}}(\Theta; K_i, T_i)$ – модельна неявна волатильність для параметрів Θ, обчислена через інверсію Блека-Шоулза з модельної ціни (одержаної через оцінювач за методом косинусного розкладання), ν_i – vega-вага, Ω – допустимий простір параметрів з обмеженнями $\kappa \in [0{,}1, 20]$, $\theta \in [10^{-3}, 0{,}5]$,

$\sigma \in [0{,}05, 2]$, $\rho \in (-0{,}99, 0{,}99)$, $v_0 \in [10^{-3}, 0{,}5]$. Обмеження виключають чисельно вироджені області ($\kappa \to 0$ чи $\sigma \to 0$).

Метод Левенберга-Марквардта [17, 2] об'єднує два підходи до розв'язання задачі (2.12): метод Гаусса-Ньютона (через лінеаризацію Якобіана цільової функції) та метод найшвидшого спуску. На кожній ітерації крок $\Delta \in R^5$ обчислюється з розв'язання лінійної системи

$$(J^T J + \mu I) \cdot \Delta = -J^T r \tag{2.13}$$

де J – матриця Якобіана нев'язок за параметрами;

μ – параметр демпфування.

У граничному випадку $\mu \to 0$ метод вироджується у Гаусса-Ньютона, а при $\mu \to \infty$ – у градієнтний спуск зі зменшеним кроком. Балансування за схемою Марквардта (збільшується при невдалих ітераціях, зменшується при вдалих) забезпечує надійну збіжність навіть з посередніх початкових наближень.

У запланованій реалізації використовуватиметься метод TRF [18] з бібліотеки scipy.optimize.least_squares, який враховує обмеження. Передбачається дослідити дві конфігурації методу. Дослідницька конфігурація – з 10 запусками з різних випадкових ініціалізацій, високою точністю зупинки, щедрим обмеженням на кількість ітерацій – призначена для побудови емпіричного апріорного розподілу параметрів на історичній вибірці. Робоча конфігурація – з ініціалізацією за результатами калібрування попереднього торгового дня, обмеженням у 50 ітерацій, одним стартовим наближенням та допуском 10^{-6} – задає реалістичний еталон для розгортання у щоденному використанні.

2.4 Архітектура нейромережевого калібратора моделі Гестона

Нейромережевий калібратор моделі Гестона у плануваній роботі є відображенням $f_w: S \to R^5$, де S – простір поверхонь неявної волатильності, дискретизованих у фіксовану сітку, а w – параметри мережі. На відміну від

класичного LM-калібрування, що розв'язує оптимізаційну задачу окремо для кожної поверхні, нейромережа є амортизованим наближенням оберненої функції Heston-оцінювача: один раз натренована на синтетичних даних, вона видає вектор Θ за один прохід прямого поширення, без ітеративної процедури.

Обрана архітектура – згорткова (CNN) – мотивована структурою вхідних даних. Поверхня неявної волатильності представляється як 2D-сітка з двома осями: грошовість K/S та строк до експірації T (у діапазоні від 7 до 365 днів). Характерні структури поверхні – форма усмішки навколо ATM–точки, нахил скосу для коротких строків, пом'якшення цих структур із зростанням строку – мають локальний характер, що добре відображається згортковими шарами.

Деталі архітектури планованого калібратора: вхід – 2D-тензор розміру 11×8 (грошовість × строк) з одним каналом, далі чотири блоки виду Conv2d → BatchNorm → ReLU → Dropout з кількістю каналів від 32 до 128, далі flatten та MLP–голова з прихованим шаром 64 нейрони і виходом у 5 параметрів. Сумарно мережа містить близько 450 тисяч ваг.

Стандартне формулювання нейромережевого калібрування [4, 19] полягає у мінімізації середньоквадратичного відхилення передбачених параметрів $\hat{\Theta}$ від істинних Θ у стандартизованому просторі на синтетичній навчальній вибірці. Цю функцію втрат позначимо

$$L_{\text{param}}(w) = E_{(\Theta,\, S)\, \sim\, D_{\text{synth}}} \left\| z\left(\hat{\Theta}(w; S)\right) - z(\Theta) \right\|^2 \qquad (2.14)$$

де z – стандартизуюче перетворення;

D_{synth} – синтетичний розподіл (поверхня, параметри).

Принципова проблема використання тільки (2.14) – оцінювання помилки відбувається у просторі параметрів, тоді як практично значущою є помилка у просторі ринкових цін (еквівалентно, у просторі неявної волатильності). Спостерігається часткова неідентифікованість: різні комбінації Θ можуть породжувати близькі поверхні. Це мотивує гібридну функцію втрат, що поєднує параметричну та поверхневу помилки.

2.5 Фізично-інформована функція втрат на основі диференційовного оцінювача

Для пом'якшення часткової неідентифікованості та розподілового зсуву між синтетикою та ринковими даними у роботі планується дослідити гібридну функцію втрат, що включає компоненту переоцінювання через диференційовний оцінювач за методом косинусного розкладання:

$$L_{\text{hybrid}}(w;\lambda) = L_{\text{param}}(w) + \lambda \cdot L_{\text{repr}}(w) \tag{2.15}$$

де $$L_{\text{repr}}(w) = E_{(\Theta,\,S)\,\sim\,D_{\text{synth}}} \left\| \sigma\left(P\left(\hat{\theta}(w;S)\right)\right) - \sigma(S) \right\|_{\text{RMSE}} \tag{2.16}$$

де P – диференційовний оцінювач за методом косинусного розкладання (підрозділ 2.2);

σ – оператор інверсії Блека–Шоулза для одержання неявної волатильності з модельної ціни;

$\|\cdot\|_{\text{RMSE}}$ – корінь з середнього квадрату по сітці.

Послідовність операцій: мережа видає $\hat{\theta}$ із вхідної поверхні S; далі оцінювач P відновлює реконструйовану ціну з $\hat{\theta}$; інверсія σ дає реконструйовану IV–поверхню; RMSE обчислюється між реконструйованою та вхідною поверхнями. Градієнт протікає назад крізь усі кроки до ваг мережі w завдяки автодиференціюванню; необхідною умовою є диференційовність оцінювача P, забезпечена реалізацією у PyTorch.

Ваговий коефіцієнт λ контролює відносну важливість фізичного компонента. У роботі планується перевірити стандартну PINN-інтуїцію («мале λ корисне як регуляризація») систематичним експериментом – перебір по $\lambda \in \{0{,}05,\ 0{,}20,\ 0{,}70,\ 2{,}00\}$ з оцінкою узагальнювальної здатності і на синтетичній валідаційній підвибірці, і на ринкових поверхнях.

2.6 Висновки до розділу 2

У розділі побудовано теоретичну основу методу. Записано стохастичні диференціальні рівняння моделі Гестона у нейтральній до ризику мірі, виведено замкнутий вираз для характеристичної функції у формулюванні Little Trap та обговорено умову Феллера, яку у плануваному дослідженні не накладено з мотивів узгодженості з ринковими даними.

Описано метод косинусного розкладання Фанг-Остерлі для оцінювання європейських опціонів через характеристичну функцію. Обґрунтовано його вибір серед альтернатив: експоненційна збіжність на гладких характеристичних функціях, відсутність невласного інтеграла та сумісність з автодиференціюванням.

Сформульовано задачу класичного калібрування Гестона у vega-зваженому просторі неявної волатильності та розглянуто метод Левенберга-Марквардта у варіанті TRF з обмеженнями. Введено дві конфігурації методу: дослідницьку для побудови емпіричного апріорного розподілу і робочу для практичного розгортання.

Наведено архітектуру згорткової нейромережі для оберненої задачі відображення поверхонь у параметри Гестона. Обґрунтовано вибір згорткової структури з огляду на просторову природу поверхні неявної волатильності. Описано стандартизатор параметрів, що приводить простір цілей навчання до зручної для оптимізатора форми.

Сформульовано дві функції втрат: чисто параметричну та гібридну з компонентою переоцінювання через диференційовний оцінювач. Виокремлено λ–залежність як центральну змінну для систематичного дослідження.

Теоретична основа, викладена у розділі, забезпечує математичну та алгоритмічну базу для проведення обчислювальних експериментів у третьому розділі дипломної роботи – порівняльного аналізу чотирьох методів калібрування на історичних поверхнях індексу SPX, перебір за ваговим

коефіцієнтом фізичного складника та аналізу фактора розриву між синтетикою та ринковими даними як кількісної міри розподілового зсуву.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

У розділі викладено реалізацію інформаційно-аналітичної системи калібрування моделі стохастичної волатильності Гестона на основі гібридного нейромережевого підходу та результати її експериментального дослідження. Описано архітектуру системи, методику підготовки ринкових даних, реалізацію диференційовного оцінювача, чотирьох методів калібрування та протоколу оцінювання, а також наведено порівняльний аналіз методів за точністю та швидкодією.

3.1 Огляд системи та номенклатура методів калібрування

Реалізовану систему калібрування моделі Гестона побудовано як інформаційно-аналітичну систему з функціонально автономними модулями. Обчислювальний процес проходить таку послідовність: попередня обробка ринкових котирувань S&P 500, побудова поверхні неявної волатильності на фіксованій сітці грошовість × строк, калібрування одним із чотирьох методів та оцінювання якості за єдиним протоколом IV–RMSE переоцінювання.

Технічним ядром системи є диференційовний оцінювач опціонів – реалізація методу косинусного розкладання, що дозволяє наскрізне обчислення градієнтів модельної ціни за параметрами Гестона засобами автоматичного диференціювання. Саме ця диференційовність робить можливим введення фізично–інформованої компоненти у функцію втрат нейромережі та забезпечує єдиний механізм переоцінювання для всіх методів – класичного і нейромережевих.

У роботі реалізовано і досліджено чотири методи калібрування, що утворюють номенклатуру для всього подальшого викладу:

Метод 0 – наївний еталон. На кожен новий торговий день використовуються параметри, одержані на попередній день, без жодної оптимізації. Метод не аналізує поточну поверхню, а просто екстраполює розв'язок попереднього дня. Його єдина роль – встановити нижню межу якості, нижче якої опускатися неприпустимо для будь–якого нетривіального методу.

Метод 1 – класичний калібратор Левенберга-Марквардта. На кожній поверхні окремо запускається ітеративна нелінійна оптимізація за методом найменших квадратів з обмеженнями на параметри. У роботі реалізовано дві конфігурації: дослідницька (широкий обчислювальний бюджет, багатопочаткова схема, жорсткі критерії збіжності) для побудови емпіричного апріорного розподілу та робоча (вузький бюджет, теплий старт від параметрів попереднього дня) для еталонного оцінювання якості. Метод 1 не має налаштовуваних параметрів, і працює безпосередньо у просторі моделі.

Метод 2 – амортизований нейромережевий калібратор. Згорткова нейронна мережа приймає на вхід вже побудовану поверхню неявної волатильності та за один прямий прохід повертає вектор параметрів Гестона. Тренування виконується на синтетичних поверхнях, для яких істинні параметри відомі; функція втрат – класичне MSE у просторі стандартизованих параметрів. Метод 2 не використовує жодної інформації про модельну ціну в момент тренування – мережа сприймає задачу як чистий регресор «поверхня – параметри».

Метод 3 – гібридний фізично–інформований калібратор. Архітектура мережі, стандартизатор, оптимізатор і протокол тренування повністю збігаються з Методом 2. Принципова відмінність – у функції втрат, яка тепер є зваженою сумою параметричної компоненти та компоненти переоцінювання у просторі неявної волатильності. Ваговий коефіцієнт цієї компоненти – параметр λ – контролює, наскільки сильно тренування мережі враховує узгодженість передбачених параметрів з модельними цінами. У роботі

систематично перебрано чотири значення λ – 0.05, 0.2, 0.7 та 2.0 – що охоплюють діапазон від слабкої регуляризації до її домінування.

Така номенклатура використовується наскрізно у викладі. Основна дослідницька гіпотеза роботи стосується Методу 3: чи дає додавання фізично–інформованої компоненти у функцію втрат покращення якості порівняно з чистим нейромережевим калібратором Методу 2, і як саме ваговий коефіцієнт λ впливає на цей результат.

3.2 Дані, протокол оцінювання та диференційовний оцінювач

Ринкові дані. Джерелом ринкових даних обрано безкоштовний рівень провайдера OptionsDX з добовими файлами котирувань європейських опціонів на індекс S&P 500 за період 2010-2023 років. Безризикові ставки одержано з бази FRED Федерального резерву США за сімома часовими рядами дохідностей казначейських облігацій (DGS1MO - DGS10). Для індексу S&P 500 ставку дивідендів покладено нульовою, оскільки SPX є безперервно реінвестованим індексом.

Очищення даних виконано за стандартними критеріями для калібрування індексних опціонів: грошовість K/S у діапазоні від 0.8 до 1.2, строк до експірації від 7 до 365 днів, ненульові ціни попиту та пропозиції з відносним спредом до 50%, неявна волатильність у діапазоні від 0.01 до 5.0, OTM–конвенція (call для опціонів вище грошей, put – нижче). Очищена множина котирувань кожної дати інтерполяється на фіксовану сітку 11×8 (одинадцять рівнів грошовості, вісім строків) лінійною апроксимацією за розсіяними точками. Середнє покриття сітки становить 89.4%; порожні комірки залишаються невизначеними і нехтуються у всіх метричних обчисленнях.

Тестова ринкова вибірка. Для оцінювання методів сформовано тестову вибірку з 178 ринкових поверхонь SPX за період квітень 2017 - грудень 2022,

стратифіковану за п’ятьма ринковими режимами: calm_2017 (35 поверхонь), covid_2020 (35), calm_2021 (35), stress_2022 (34) та transition (39 перехідних дат). Така стратифікація дозволяє оцінити поведінку методів окремо у спокійних, кризових та перехідних умовах ринку.

Синтетичний набір даних. Тренування і первинне валідаційне оцінювання нейромережевих методів виконано на синтетичних поверхнях, для яких істинні параметри Гестона відомі. Синтетичний набір згенеровано з емпіричного апріорного розподілу: до приблизно 660 історичних торгових дат застосовано Метод 1 у дослідницькій конфігурації, після відсіювання незбіжних та граничних розв’язків на отриманому емпіричному наборі параметричних векторів оцінено багатовимірний нормальний розподіл у трансформованому просторі. З цього розподілу згенеровано 500 тис. параметричних векторів, для кожного через виклик диференційовного оцінювача побудовано штучну поверхню неявної волатильності на тій самій сітці 11×8. Тренувальна частина мережі – детермінована підмножина зі 100 тис. поверхонь, поділена на тренувальну, валідаційну та тестову у пропорції 90/5/5 (90 тис., 5 тис., 5 тис. відповідно).

Метрика та протокол оцінювання. Основною метрикою якості калібрування є IV–RMSE переоцінювання – корінь з середньоквадратичного відхилення між вхідною поверхнею та реконструйованою через диференційовний оцінювач від каліброваних параметрів. Усереднення виконується за всіма валідними комірками без vega–зважування, тобто кожна комірка має однакову вагу. Такий вибір метрики забезпечує природне порівняння методів: жоден з них не оптимізує саме vega–незважену метрику безпосередньо, тому метрика є зовнішньою для всіх методів і коректно порівнює їх. Швидкодію виміряно окремим стандартизованим протоколом за медіаною 90 послідовних викликів калібратора на одній обчислювальній платформі.

Диференційовний оцінювач. Технічною передумовою формулювання Методу 3 є наявність диференційовного оцінювача опціонів. У роботі

реалізовано метод косинусного розкладання Фанг–Остерлі [12] для моделі Гестона у формулюванні Little Trap [1], що уникає розривів комплексного логарифма у виразах для коефіцієнтів. Усі обчислення виконуються засобами PyTorch у режимі автоматичного диференціювання, що забезпечує наскрізне обчислення градієнтів модельної ціни та неявної волатильності за параметрами Гестона.

Ціна європейського опціону у моделі Гестона записується через характеристичну функцію у вигляді скінченної суми тригонометричних коефіцієнтів (повний вивід формули наведено у підрозділі 2.2 Розділу 2):

$$V(S,T) = exp(-r \cdot T) \cdot \Sigma Re(\varphi(u_k, T) \cdot U_k) \quad (3.1)$$

де φ – характеристична функція моделі Гестона;

u_k – аналітично обчислювані коефіцієнти косинусного розкладання, що залежать лише від типу опціону та його ціни виконання, а сума береться за фіксованим числом мод (у реалізації – 64).

Перехід від модельної ціни до неявної волатильності виконується ітераційно методом Ньютона, який також реалізовано у диференційовній формі.

3.3 Чотири методи калібрування

У підрозділі описано концептуальну побудову чотирьох методів калібрування, що зіставляються у роботі. Деталі реалізації, специфічні для відтворюваності експериментів, винесені у додатки до роботи.

Метод 0 (наївний еталон). Калібрування зведено до тривіальної екстраполяції: параметри, одержані на (n–1)-у торгову дату, використовуються як розв’язок для n-ої. Жодна інформація про поточну поверхню не аналізується. Цей метод не призначений для практичного використання, але його роль як еталона критично важлива: якщо нетривіальний метод не перевершує наївну екстраполяцію, це є ознакою того,

що він не виконує реальної роботи з калібрування, а лише відтворює інерцію ринку.

Метод 1 (LM). Класичний калібратор реалізує метод Левенберга-Марквардта у варіанті дзеркального відбиття у довірчій підобласті з обмеженнями на параметри. Вектор нев'язок утворено vega–зваженими різницями неявної волатильності між ринковою та модельною поверхнями. Цільова функція мінімізується ітеративно на кожній поверхні окремо:

$$F(\theta) = \sum_{i,j} w_{ij} \cdot (IV_{\text{ринок}} - IV_{\text{модель}}(\theta))^2 \quad (3.2)$$

Параметричні обмеження покладено штучно широкими (κ у діапазоні [0.01, 20], θ – $[10^{-4}, 1]$, σ_v– [0.01, 5], ρ – [-0.999, 0.999], v_0– $[10^{-4}, 1]$), щоб уникнути зупинки оптимізатора на границі. Умова Феллера не накладається, що відповідає сучасній практиці калібрування індексних опціонів.

У роботі реалізовано дві конфігурації Методу 1. Дослідницька конфігурація використовує широкий обчислювальний бюджет (до 200 викликів функції нев'язок), жорсткі критерії збіжності та багатопочаткову схему з чотирма точками старту – вона потрібна для високоякісного калібрування історичних дат, з якого далі будується синтетичний апріорний розподіл. Робоча конфігурація моделює оперативне промислове розгортання: 50 викликів функції, послаблені критерії та теплий старт від параметрів попереднього дня. Саме робоча конфігурація використовується як еталон у подальших порівняннях. На тестовій вибірці 178 поверхонь робоча конфігурація досягає збіжності у 98.9% випадків за середньої кількості викликів функції 8.2 – значно менше дозволеного бюджету.

Метод 2 (нейромережевий калібратор). Архітектура HestonCNN складається з трьох послідовних згорткових блоків та двошарової повнозв'язної голови. Згорткові блоки локально агрегують інформацію про форму поверхні з ядром 3×3 та послідовним розширенням простору ознак до 64 каналів; шари об'єднання не використовуються, оскільки кожна комірка сітки несе інформацію про окремий квадрант поверхні. Голова мережі – двошаровий персептрон з 64 прихованими нейронами та виходом у п'ять

параметрів Гестона. Загальна кількість параметрів мережі – приблизно 420 тис., що дозволяє повне тренування без апаратного прискорення.

Параметри Гестона на виході мережі представлено у стандартизованому просторі: на додатних компонентах виконано логарифмування з нижньою обмежувальною константою, на кореляції ρ – atanh-перетворення, що відображає (–1, 1) на R, а у трансформованих координатах застосовано z-стандартизацію. Стандартизатор реалізовано як диференційовний у обох напрямках модуль – це дозволить у Методі 3 вбудувати обернений перехід у функцію втрат, де знадобиться відновлення фізичних параметрів для виклику диференційовного оцінювача.

Функція втрат Методу 2 – середньоквадратичне відхилення передбачених параметрів від істинних у стандартизованому просторі:

$$L_2 = E[(z_{pred} - z_{true})^2] \tag{3.3}$$

Тренування виконано оптимізатором AdamW з графіком швидкості навчання за схемою косинусного відпалювання та параметром терплячості ранньої зупинки 3. Принциповим є вибір метрики для ранньої зупинки: у роботі обрано не валідаційну функцію втрат, а валідаційну IV–RMSE переоцінювання на синтетичних поверхнях. Це узгоджує тренування з кінцевою метою методу – точною реконструкцією поверхні, а не самою точністю відновлення параметрів, які через часткову неідентифікованість моделі Гестона можуть давати близькі поверхні при відмінних значеннях.

Метод 3 (фізично-інформований калібратор). Метод 3 розширює Метод 2 додаванням компоненти переоцінювання у функцію втрат. Архітектура мережі, стандартизатор, оптимізатор та критерій ранньої зупинки повністю ідентичні Методу 2. Принципова відмінність – функція втрат є зваженою сумою параметричної компоненти та компоненти переоцінювання:

$$L_3 = L_2 + \lambda \cdot R_{reprice} \tag{3.4}$$

де компонента переоцінювання $R_{reprice}$ обчислюється як IV–RMSE реконструйованої поверхні відносно вхідної. Шлях обчислення такий: передбачений нейромережею стандартизований вектор параметрів проходить

через обернений перехід стандартизатора у природний простір параметрів Гестона; через диференційовний оцінювач обчислюються модельні ціни на сітці грошовість × строк; через інверсію Ньютона будується реконструйована поверхня неявної волатильності; нарешті, обчислюється середньоквадратичне відхилення від цільової поверхні. Завдяки наскрізній диференційовності градієнт компоненти $R_{reprice}$ автоматично входить у крок оптимізатора.

Ваговий коефіцієнт λ контролює відносну важливість компоненти переоцінювання. У границі λ = 0 Метод 3 збігається з Методом 2. У границі λ дуже великого параметрична компонента стає незначною і мережа оптимізується безпосередньо за якістю реконструкції поверхні. У роботі досліджено чотири значення λ: 0.05, 0.2, 0.7 та 2.0, що охоплюють діапазон приблизно у півтори декади. Малі значення відповідають режиму слабкої фізичної регуляризації, великі – режиму домінування фізичної компоненти. Для кожного значення λ мережа тренується незалежно з однаковими налаштуваннями.

Вибір остаточного значення λ. Для вибору найкращого значення застосовано стандартну ML–дисципліну: чотири моделі натреновано, обрано ту, що дає найменшу валідаційну IV–RMSE на синтетичній валідаційній частині. Цей вибір принципово виконується без врахування ринкової вибірки – тестова вибірка призначена винятково для остаточного оцінювання якості, не для налаштування гіперпараметрів. Поведінка лямбда–залежності та обрана модель детально аналізуються у наступних підрозділах.

3.4 Якість калібрування на синтетичних даних

Першим етапом експериментального дослідження є оцінювання якості методів на синтетичному тестовому наборі з 500 поверхонь, побудованих з тієї ж генеративної моделі, на якій тренувалися нейромережі. У цій умові розподіловий зсув між тренуванням і тестуванням відсутній: усі чотири

методи оцінюються на поверхнях, що породжені моделлю Гестона з параметрами того ж розподілу, з якого витягувалися тренувальні дані. Результати наведено у табл. 3.1.

Таблиця 3.1 – Якість калібрування на синтетичному тестовому наборі (500 поверхонь): середня IV-RMSE та середні абсолютні відхилення параметрів Гестона відносно істинних значень.

Метод	**IV-RMSE**	**MAE κ**	**MAE θ**	**MAE σ_v**	**MAE ρ**	**MAE v_0**
Метод 0	–	4.720	0.098	0.330	0.080	0.060
Метод 1	0.0075	0.065	0.001	0.013	0.0005	$4 \cdot 10^{-5}$
Метод 2	0.0089	0.283	0.054	0.566	0.063	0.010
Метод 3 (λ=0.05)	0.0080	0.112	0.052	0.624	0.065	0.013

Класичний Метод 1 у дослідницькій конфігурації досягає найменшої IV–RMSE 0.0075 і одночасно відновлює параметри з високою точністю: MAE параметра κ становить 0.065, інших компонент – на порядок менші. Це характеризує Метод 1 як близький до точного відновлювача на синтетиці – результат очікуваний, оскільки синтетика побудована саме з модельних поверхонь.

Чисто нейромережевий Метод 2 досягає IV–RMSE 0.0089 – на 19% вище за Метод 1. Параметричні похибки істотно більші, особливо для κ та σ_v (MAE відповідно 0.283 та 0.566). Цей результат відображає відому особливість моделі Гестона: часткова неідентифікованість у напрямку κ-σ_v означає, що різні комбінації цих параметрів можуть давати майже однакові поверхні. Нейромережа, оптимізуючи MSE у просторі параметрів, не отримує прямого зворотного зв’язку від поверхні і знаходить розв’язок, який задовольняє параметричній метриці, але не обов’язково відповідає істинному поєднанню параметрів.

Гібридний Метод 3 з валідаційно обраним значенням λ = 0.05 досягає IV-RMSE 0.0080 – покращення на 10% відносно Методу 2 і наближення до

показника класичного Методу 1. Цей результат підтверджує методологічну гіпотезу: додавання компоненти переоцінювання у функцію втрат, навіть з малою вагою, покращує якість калібрування у середовищі, для якого мережу натреновано. Параметричні похибки Методу 3 також помітно менші за похибки Методу 2 (MAE к зменшується з 0.283 до 0.112), хоча для σ_v покращення відсутнє, що свідчить про обмеження поточної форми регуляризації при невеликому значенні вагового коефіцієнта.

Наївний Метод 0 не призначений для оцінювання на синтетиці і у відповідному стовпці IV-RMSE замінено риску, оскільки екстраполяція з попередньої дати не має сенсу у незалежному наборі поверхонь. Параметричні похибки наведено для повноти картини: вони характеризують типову відстань між сусідніми параметричними векторами у згенерованому наборі і є природним фоном для оцінки досягнень інших методів.

На синтетичних даних усі нетривіальні методи демонструють якість, близьку до нижньої межі шуму. Метод 3 з валідаційно обраним λ кращий за Метод 2 на 10%, що підтверджує очікуване методологічне покращення від додавання фізично-інформованої компоненти. Цей результат відповідає інтуїції, поширеній у літературі фізично-інформованих калібраторів [4, 6, 7, 8]: відомості про модельну ціну, введені у функцію втрат, утворюють додатковий регуляризатор, що покращує узагальнення на тестовому наборі тієї ж природи.

3.5 Перебір вагового коефіцієнта λ для Методу 3 на синтетиці

Систематичний перебір вагового коефіцієнта λ дозволяє дослідити, як змінюється якість калібрування Методу 3 при переході від слабкої до сильної фізичної регуляризації. У таблиці 3.2 наведено результати чотирьох конфігурацій Методу 3 на синтетичному валідаційному наборі – середовищі, у якому тренувалась мережа і у якому правомірне порівняння за валідаційною метрикою.

Таблиця 3.2 – Результати перебору вагового коефіцієнта λ Методу 3 на синтетичних даних: епоха ранньої зупинки, валідаційна IV–RMSE та ранг конфігурацій за валідаційною метрикою.

Конфігурація	**Епоха зупинки**	**Валідаційна IV-RMSE**	**Ранг**
Метод 3 (λ=0.05)	10	0.00909	1
Метод 3 (λ=0.20)	8	0.00968	3
Метод 3 (λ=0.70)	4	0.00958	2
Метод 3 (λ=2.00)	5	0.00984	4

На рис. 3.1 показано, як валідаційна IV–RMSE Методу 3 на синтетичному наборі поводиться як функція вагового коефіцієнта λ (на рисунку відображено обидві криві – синтетичну та ринкову; у цьому підрозділі обговорюється лише синтетична вісь, ринковий аспект буде розглянуто у підрозділі 3.7).

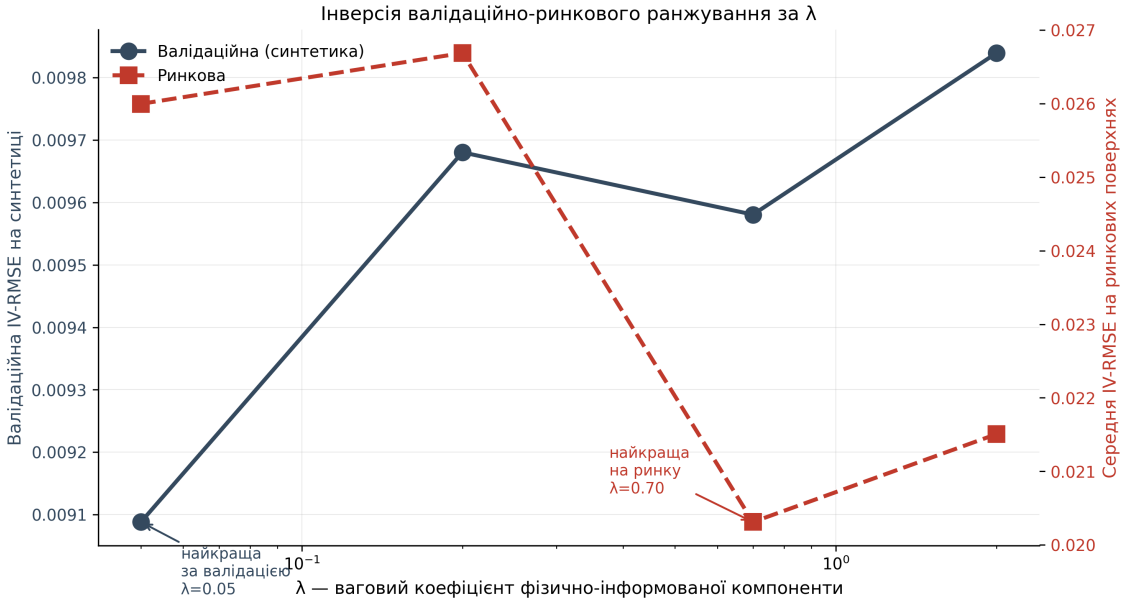


Рисунок 3.1 – Залежність IV–RMSE Методу 3 від вагового коефіцієнта λ: валідаційна IV-RMSE на синтетиці (ліва вісь, темна крива) та середня IV-RMSE на ринковій вибірці (права вісь, червона крива).

На синтетичній валідації найкращою є конфігурація з λ = 0.05 (IV-RMSE 0.00909), що на 8% краще за найгірше досліджене значення λ = 2.0 (IV-RMSE 0.00984). Конфігурації з λ = 0.7 та 0.2 займають проміжні позиції з IV-RMSE

0.00958 та 0.00968 відповідно. Відмінності між сусідніми значеннями λ є помірними (близько 0.001 IV-RMSE), але систематичними у тому сенсі, що валідаційний ранг 1 чітко належить найменшому з досліджених значень.

Мала вага компоненти переоцінювання (λ = 0.05) у тренуванні дозволяє мережі зосередитися насамперед на параметричній компоненті MSE – тобто на якомога точнішому відновленні саме параметрів Гестона, з яких генеровано тренувальну поверхню. Оскільки тестовий синтетичний набір походить з того ж розподілу, відмінне відновлення параметрів природним чином дає малу IV-RMSE під час переоцінювання. Велика вага компоненти переоцінювання (λ = 2.0), навпаки, спрямовує мережу шукати параметри, що відтворюють вхідну поверхню, не обов'язково істинні. На синтетичному тестовому наборі, де істинні параметри і дають істинну поверхню, така переорієнтація не дає переваги і помітно погіршує валідаційну метрику.

Епохи ранньої зупинки додають важливий контекст. Конфігурації з λ = 0.7 та 2.0 зупиняються після 4 і 5 епох відповідно – тобто додавання значної фізичної компоненти прискорює досягнення локального оптимуму у функції втрат, але одночасно обмежує досяжний рівень валідаційної метрики через зміщення цілі тренування. Конфігурації з малим λ тренуються довше (8 – 10 епох) і досягають кращої валідаційної якості.

Підрозділ підтверджує робочу гіпотезу щодо фізично-інформованих калібраторів: ваговий коефіцієнт λ є справді функціональним важелем налаштування. Він не є формальною константою, а контролює конкурентний баланс між двома компонентами функції втрат – точністю відновлення параметрів та узгодженістю передбачень з модельними цінами. На синтетиці, де ці дві мети узгоджені, оптимум зміщений до малого λ. Як зміщується баланс під час переходу до ринкових даних – окреме питання, що визначає висновки підрозділу 3.7.

3.6 Порівняння методів за швидкодією

Швидкодія калібратора є самостійним критерієм оцінювання поряд із точністю – особливо у застосуваннях потокового калібрування (інтрадейних оновлень, реакція на новинні події, обчислення на великих портфелях). Швидкодію виміряно окремою стандартизованою процедурою: для кожного методу виконано 100 послідовних викликів калібратора на однаковій підмножині з 10 поверхонь, перші 10 викликів відсічено як холодний старт, медіану обчислено за 90 замірами. Результати наведено у табл. 3.3.

Таблиця 3.3 – Медіана часу виконання чотирьох методів калібрування та прискорення відносно класичного Методу 1.

Метод	**Інференс (мс)**	**Повний конвеєр (мс)**	**Прискорення відн. Методу 1**
0	0.001	3.86	-
1	49.07	52.93	1×
2	0.27	4.14	179×
3	0.27	4.13	180×

Класичний Метод 1 у робочій конфігурації виконується за медіаною 49.1 мс інференсу. Нейромережеві методи показують істотну перевагу: медіана інференсу обох становить 0.27 мс – у 179 разів швидше за Метод 1. На повному конвеєрі, що включає завантаження поверхні та переоцінювання, медіана становить 4.1 мс – лише у 13 разів швидше за Метод 1, оскільки завантаження поверхні (~1.1 мс) та переоцінювання (~2.7 мс) є спільними для всіх методів і не залежать від власне калібратора.

Парето-фронт «швидкодія – точність» (рис. 3.2) ілюструє це практичне розділення сфер застосування. На обох панелях видно, що Метод 1 і нейромережеві методи займають різні точки Парето-фронту: Метод 1 досягає кращої точності ціною у багато разів більшого часу, нейромережі забезпечують істотну перевагу за швидкістю ціною помірного погіршення

точності. Метод 0 не дає реальної переваги: найшвидший, але одночасно найгірший, що домінується будь-яким нетривіальним методом.

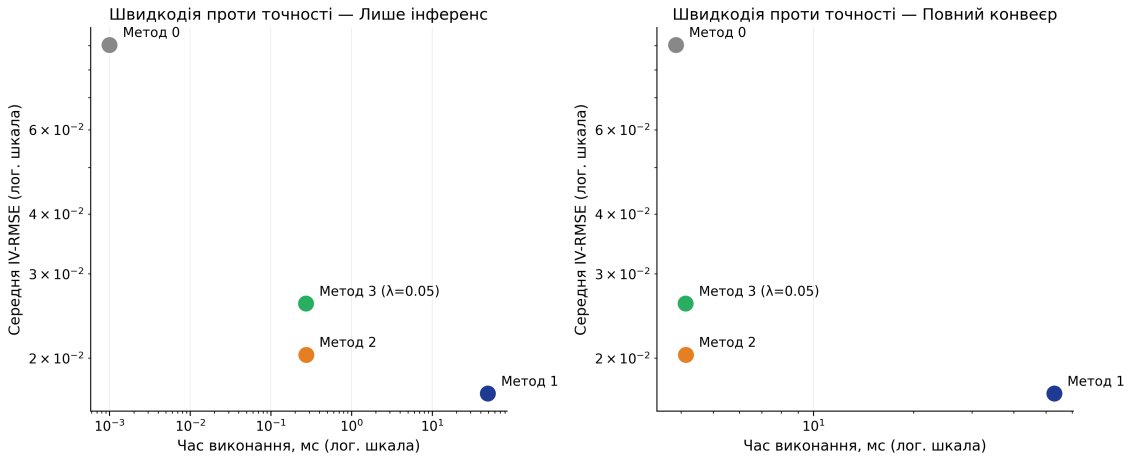


Рисунок 3.2 – Фронт швидкодія – точність для чотирьох методів на логарифмічній шкалі: інференс (ліворуч) та повний конвеєр (праворуч).

Практичний висновок щодо швидкодії формує природне розділення сфер застосування двох сімейств методів. Для пакетних дослідницьких задач, у яких якість калібрування є визначальним критерієм, а час виконання на одну поверхню не є вузьким місцем, раціональним вибором залишається Метод 1. Для онлайн-систем потокового калібрування, інтрадейних оновлень амортизована нейромережа дає виграш у швидкодії на два порядки, що часто виправдовує помірне погіршення точності.

3.7 Узагальнення на ринкових даних та обмеження моделі Гестона

Усі попередні результати – якість Методів 1, 2 та 3, покращення Методу 3 над Методом 2, обрана конфігурація $\lambda = 0.05$ – стосуються синтетичних поверхонь, породжених моделлю Гестона. Зараз ці методи застосовано до 178 реальних ринкових поверхонь S&P 500. Цей перехід виявляє три структурні явища, що мають спільну природу і вказують не на недоліки розглянутих методів калібрування, а на принципове обмеження самої моделі Гестона як моделі ринку.

Якість на ринкових поверхнях. Середні IV-RMSE методів на ринковій вибірці помітно вищі, ніж на синтетиці (табл. 3.4 та рис. 3.3). Метод 1 досягає 0.0169, Метод 2 – 0.0203, Метод 3 з валідаційно обраним λ = 0.05: 0.0260. Усі три значення у 2.2-3.3 раза перевищують відповідні синтетичні показники, хоча на синтетиці ці методи демонстрували якість, близьку до нижньої межі шуму.

Таблиця 3.4 – Зіставлення середньої IV-RMSE на синтетичному тестовому наборі та на ринковій вибірці S&P 500.

Метод	Синтетика	Ринок	Фактор розриву
1	0.0075	0.0169	2.25x
2	0.0089	0.0203	2.28x
3 (λ=0.05)	0.0080	0.0260	3.25x
3 (λ=0.20)	0.0088	0.0267	3.03x
3 (λ=0.70)	0.0087	0.0203	2.33x
3 (λ=2.00)	0.0092	0.0215	2.35x

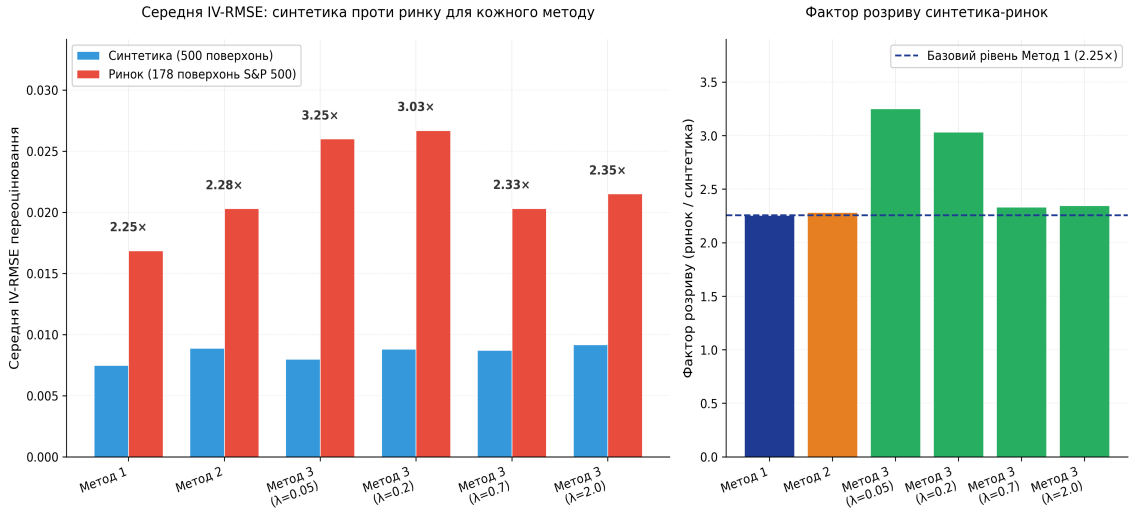


Рисунок 3.3 – Порівняння IV-RMSE на синтетиці та на ринку для шести методів.

Фактор розриву синтетика-ринок – відношення ринкової IV-RMSE до синтетичної – є цільовим діагностичним показником цього підрозділу. Базовий рівень, спостережуваний для Методу 1, становить 2.25x. Цей рівень

принципово важливо інтерпретувати правильно. Метод 1 не залежить від тренувального розподілу і не має параметрів, що могли б підлаштуватися під синтетику, – він просто шукає найкраще наближення у класі поверхонь Гестона до кожної конкретної ринкової поверхні. Тому 2.25x не є мірою якості методу, а є мірою принципової відстані між класом поверхонь, які модель Гестона здатна породити, і класом реальних ринкових поверхонь.

Іншими словами, базовий фактор розриву 2.25x є нижньою межею для будь-якого калібратора моделі Гестона на реальних ринкових поверхнях. Незалежно від того, наскільки досконалим буде нейромережевий або класичний оптимізатор, він не може переоцінити ринкову поверхню точніше, ніж її дозволяє апроксимувати сама модель Гестона. Цей рівень похибки спричинений структурними особливостями ринку, які модель Гестона не здатна відтворити: виразніший перекіс короткострокової волатильності, поодинокі дислокації у строковій структурі, ефекти грубої волатильності у сенсі Гетераля-Жайсона-Розенбаума [15] та інші регулярності, що відсутні у двофакторному дифузійному формулюванні Гестона.

Поведінка нейромережевих методів відносно базового рівня показова. Метод 2 на ринку має фактор розриву 2.28x – практично збігається з базою Методу 1. Це означає, що нейромережа, натренована лише на синтетиці, не вносить додаткової похибки понад ту, що зумовлена обмеженням моделі. Метод 3 з валідаційно обраним $\lambda = 0.05$ має істотно вищий фактор 3.25x, але великі значення λ (0.7 та 2.0) знову повертають фактор до базового рівня (2.33x та 2.35x). Тобто сильна фізична регуляризація змушує нейромережу триматися класу поверхонь, що модель Гестона здатна породити, і саме тоді її якість на ринку максимально наближається до якості класичного оптимізатора в межах того самого класу.

Зіставлення синтетичного і ринкового ранжування значень λ (табл. 3.5) демонструє характерну невідповідність. На синтетичній валідації найкращим є $\lambda = 0.05$, на ринку – $\lambda = 0.7$. Коефіцієнт Спірмена між двома ранжуваннями дорівнює нулю, тобто валідаційний рейтинг не несе інформації про ринковий.

Таблиця 3.5 – Зіставлення ранжування значень λ за валідаційною IV-RMSE на синтетиці та середньою IV-RMSE на ринку.

λ	Валідаційна IV–RMSE	Ринкова IV–RMSE	Ранг (синтетика)	Ранг (ринок)
0.05	0.00909	0.0260	1	3
0.20	0.00968	0.0267	3	4
0.70	0.00958	0.0203	2	1
2.00	0.00984	0.0215	4	2

Інтерпретація цієї невідповідності у термінах обмеження моделі є природною. На синтетичних даних поверхні породжуються безпосередньо моделлю Гестона з відомими параметрами; тут тренувальна мета мережі (точно відновити параметри) і кінцева оцінювальна мета (точно реконструювати поверхню) є практично еквівалентними. Тому найкращим є малий λ, що дає мережі максимальну свободу мінімізувати параметричну MSE. На ринкових поверхнях, однак, передбачуваність параметрів через мережу наражається на принципову проблему: ці параметри взагалі неможливо відновити точно, тому що ринкова поверхня не породжується моделлю Гестона з жодним фіксованим набором параметрів. Класичний Метод 1 у такій ситуації знаходить найкращий проєкційний розв’язок у класі Гестона; нейромережевий Метод 3 з великим λ також змушено триматися цього класу і виходить близько до класичного результату; нейромережевий Метод 3 з малим λ оптимізує параметричну метрику без зв’язку з поверхнею і отримує гірший результат, оскільки сама ця метрика на ринкових даних не має чіткої цілі.

Для подальшого підтвердження природи явища корисно розглянути найбільш стресовий ринковий період – кризу березня 2020 року (рис. 3.4). На лівій панелі рисунка видно, що пік IV-RMSE для всіх чотирьох методів припадає на дати 17-25 березня 2020 – момент кризового обвалу, пов’язаного з пандемією COVID-19. У цей період навіть Метод 1 – класичний оптимізатор без жодних залежностей від тренувального розподілу – показує локальне

погіршення якості. Це підтверджує, що погіршення спричинене не специфікою нейромережевих методів, а самою природою ринкової поверхні: у кризові періоди структурні відхилення від класу Гестона стають максимальними і жодна апроксимація у цьому класі не може дати малу похибку.

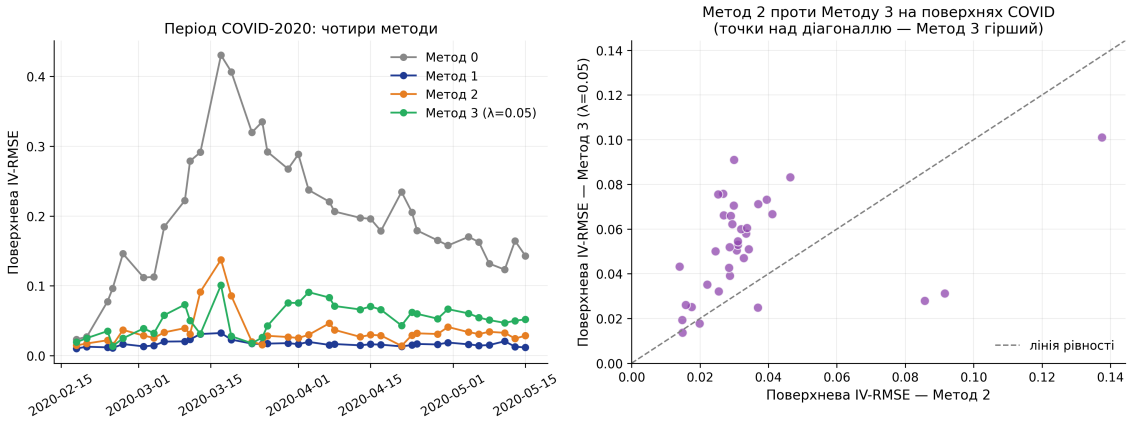


Рисунок 3.4 – часовий ряд IV–RMSE за датами 17–25 березня 2020(ліва панель); права панель порівнює поверхневі похибки Методу 2 і Методу 3 на найскладніших поверхнях кризи.

Усі три структурні явища – підвищений фактор розриву, інверсія валідаційно-ринкового рейтингу λ та спільне погіршення методів у кризові періоди – вказують на одне і те саме: на принципове обмеження моделі Гестона як апроксимації ринкової динаміки. Методологія нейромережевого калібрування з фізично-інформованою компонентою функції втрат на синтетичних даних показує очікуване покращення (підрозділ 3.4) і коректно реагує на ваговий коефіцієнт λ (підрозділ 3.5). Перехід до ринкових даних не виявляє недоліку самої методології, а виявляє обмеження модельного класу, у якому працюють усі чотири розглянуті методи. Звідси випливає природний наступний крок розвитку – перехід до багатших модельних класів (грубої волатильності, моделі зі стрибками, локально-стохастичні розширення), у яких базовий фактор розриву синтетика-ринок буде істотно меншим.

Запропонований у роботі діагностичний інструмент – фактор розриву синтетика-ринок, що повідомляється поряд з агрегованою IV-RMSE, –

дозволяє формально розділити два джерела похибки: обмеження модельного класу (спільна нижня межа для всіх методів) та індивідуальну якість методу понад цю межу. Без такого розділення показники якості на ринковому тесті можна неправильно інтерпретувати як оцінку методу, тоді як вони значною мірою відображають властивості самої моделі. Систематичне використання фактора розриву у літературі нейромережевого калібрування зробило б порівняння методів між собою і з класичними еталонами помітно інформативнішим.

3.8 Висновки до розділу 3

Реалізовано інформаційно-аналітичну систему калібрування моделі стохастичної волатильності Гестона з чотирма методами: наївним еталоном (Метод 0), класичним методом Левенберга-Марквардта (Метод 1), нейромережевим калібратором (Метод 2) та гібридним фізично-інформованим калібратором (Метод 3) з систематичним перебором вагового коефіцієнта λ. Усі методи оцінено за єдиним протоколом IV-RMSE переоцінювання на синтетичному тестовому наборі та на 178 ринкових поверхнях S&P 500.

На синтетичних даних показано очікуване методологічне покращення від додавання фізично-інформованої компоненти у функцію втрат: Метод 3 з валідаційно обраним $\lambda = 0.05$ досягає IV-RMSE 0.0080, що на 10% краще за чисто нейромережевий Метод 2 (0.0089) і наближається до якості класичного Методу 1 (0.0075). Перебір λ підтверджує, що ваговий коефіцієнт є функціональним важелем налаштування і дозволяє контрольовано зміщувати баланс між точністю відновлення параметрів та узгодженістю передбачень з модельними цінами Гестона.

За швидкодією нейромережеві методи на два порядки випереджають класичний: 0.27 мс інференсу проти 49.1 мс. Це формує природне розділення сфер застосування: класичний Метод 1 – у дослідницьких задачах з акцентом

на точність; нейромережеві Методи 2 та 3 – в онлайн-системах потокового та інтрадейного калібрування.

Перехід до ринкових даних виявляє три структурні явища, що мають спільну природу: всі методи показують підвищений фактор розриву синтетика-ринок (базовий рівень 2.25x), ранжування значень λ за валідаційною та ринковою метриками є повністю декорельованим (Спірмен = 0), а локальне погіршення якості у кризовий період covid_2020 є спільним для всіх чотирьох методів. Усі три явища спричинені не недоліком розглянутих методів калібрування, а обмеженням самої моделі Гестона – двофакторне дифузійне формулювання не здатне точно описати ефекти грубої волатильності, виражений перекіс короткострокової поверхні та структурні дислокації, характерні для реальних ринкових даних. Класичний Метод 1, що не залежить від тренувального розподілу, встановлює нижню межу досяжної якості у класі моделі Гестона; нейромережевий Метод 3 з сильною фізичною регуляризацією ($\lambda = 0.7$) досягає тієї ж нижньої межі, що підтверджує коректність методології.

Основним науковим внеском роботи є виявлення явища валідаційно-ринкової інверсії у задачі нейромережевого калібрування з фізично-інформованою компонентою та формулювання фактора розриву синтетика-ринок як діагностичного показника, що дозволяє відокремити обмеження модельного класу від індивідуальної якості методу. Запропоноване діагностичне звітування – фактор розриву поряд з агрегованою IV-RMSE забезпечує коректну інтерпретацію результатів порівняння нейромережевих методів між собою і з класичними еталонами у контексті, де всі вони обмежені властивостями обраного модельного класу.

Природний напрям подальших досліджень випливає з сформульованого пояснення: перехід до багатших модельних класів (грубої волатильності, моделі зі стрибками, локально-стохастичні розширення), у яких базовий фактор розриву синтетика-ринок буде істотно меншим за 2.25x. Розроблена методологія нейромережевого калібрування з фізично-інформованою

компонентою функції втрат, а також запропонований діагностичний показник, переносяться на ці багатші класи без структурних змін – змінюється лише диференційовний оцінювач і апріорний розподіл параметрів.

РОЗДІЛ 4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У розділі проведено техніко-економічне обґрунтування розробленого програмного продукту для калібрування моделі стохастичної волатильності Гестона із застосуванням функціонально-вартісного аналізу порівняно два конкуруючі варіанти реалізації – гібридний нейромережевий калібратор з фізично-інформованою функцією втрат та класичний калібратор Левенберга-Марквардта. За результатами аналізу обчислено коефіцієнти технічного рівня та повну вартість розробки кожного варіанту, виконано обґрунтований вибір варіанту реалізації для подальшого дослідження.

4.1 Постановка задачі проектування

Метою етапу є створення інформаційно-аналітичного програмного комплексу, що автоматизує калібрування моделі стохастичної волатильності Гестона за ринковими поверхнями неявної волатильності індексу S&P 500. Розроблений продукт повинен виконувати наскрізне обчислення: попередню обробку котирувань опціонів, побудову фіксованої сітки грошовість-строк до експірації, оцінку параметрів моделі обраним методом калібрування та оцінювання якості калібрування за єдиним протоколом IV-RMSE переоцінювання.л

Постановка задачі полягає в обґрунтованому виборі архітектури програмного продукту з-поміж конкуруючих варіантів реалізації за критерієм техніко-економічного рівня. Конкурують два підходи до калібрування Гестона: класичний детермінований метод Левенберга-Марквардта з розширеним тюнінгом та амортизований нейромережевий калібратор з фізично-інформованою функцією втрат. Перший потребує повторного

запуску нелінійної оптимізації на кожній новій поверхні; другий вимагає одноразового тренування мережі, після чого калібрування здійснюється за одиничний прямий прохід.

Технічні вимоги до програмного продукту є такими:

1) функціонування у стандартному середовищі мови Python із застосуванням наукових пакетів для лінійної алгебри, оптимізації та автоматичного диференціювання;
2) підтримка диференційовного обчислення модельних цін європейських опціонів, необхідного для роботи градієнтних методів навчання нейромережевих калібраторів;
3) забезпечення повної відтворюваності результатів калібрування на синтетичних і ринкових даних;
4) зручне зберігання та повторне використання навчених моделей разом із відповідною статистикою якості;
5) модульна архітектура, що дозволяє у подальшому замінити модельний клас без переписування калібраційного конвеєра;
6) мінімізація апаратних вимог зі збереженням продуктивності, достатньої для пакетної обробки сотень поверхонь на типовому робочому комп’ютері.

4.2 Обґрунтування функцій програмного продукту

З урахуванням сформульованих технічних вимог, у структурі програмного продукту виділено чотири основні функції, які визначають як технічні характеристики майбутньої системи, так і трудомісткість її розробки. Головна функція F0 – реалізація замкненого конвеєра калібрування моделі Гестона за ринковими поверхнями неявної волатильності – розкладається на чотири підфункції.

F1 – вибір мови програмування:

1) Python з пакетами NumPy, SciPy та PyTorch;
2) C++ з ручною реалізацією автоматичного диференціювання.

F2 – архітектура калібратора моделі Гестона:

1) класичний калібратор Левенберга-Марквардта з обмеженнями, багатопочатковою схемою та теплим стартом від результату попереднього дня;
2) гібридний нейромережевий калібратор з фізично–інформованою функцією втрат, що поєднує параметричну компоненту й компоненту переоцінювання у просторі неявної волатильності.

F3 – реалізація диференційовного оцінювача опціонних цін:

1) власна реалізація методу косинусного розкладання Фанг–Остерлі з модифікацією Little Trap, що забезпечує наскрізну сумісність з автоматичним диференціюванням;
2) інтеграція готового зовнішнього пакету ціноутворення опціонів (QuantLib через Python–обгортку), яка дає виграш у швидкості, але не дозволяє наскрізне обчислення градієнтів.

F4 – тип обчислювальної інфраструктури:

1) локальний робочий комп'ютер з графічним прискорювачем рівня персонального GPU;
2) розподілене середовище з орендованим хмарним кластером GPU.

Запропоновані варіанти реалізації для всіх чотирьох функцій утворюють морфологічну множину, з якої формуються технічно сумісні комбінації. Аналіз кожного варіанту через позитивно–негативну матрицю наведено в таблиці 4.1.

Таблиця 4.1 – Позитивно–негативна матриця варіантів реалізації функцій.

Функція	**Варіант**	**Переваги**	**Недоліки**
F1	1) Python	Розвинена екосистема пакетів для наукових обчислень, машинного навчання й автоматичного диференціювання; швидке прототипування	Інтерпретована мова, нижча швидкодія базових циклів порівняно з компільованими аналогами
F1	2) C++	Найвища швидкодія обчислювального ядра; повний контроль розміщення пам'яті	Тривалий цикл розробки; обмежений вибір готових ML–фреймворків; складність наскрізного автодиференціювання
F2	1) Класичний LM	Детермінований результат; інтерпретовані параметри	Висока обчислювальна вартість на кожній поверхні; необхідність тривалого емпіричного тюнінгу обмежень і вагів нев'язки
F2	2) Гібридний нейромережевий	Амортизована вартість калібрування у режимі інференсу; гнучкість функції втрат	Потреба в тренувальному наборі поверхонь; ризик зміщення на розподілі, відмінному від тренувального
F3	1) Власна COS	Наскрізна підтримка автоматичного диференціювання;	Більший обсяг власного коду; необхідність ретельного тестування збіжності
F3	2) QuantLib	Перевірена бібліотека з широким спектром перевірених схем ціноутворення; стабільні числові результати	Відсутність наскрізного автоматичного диференціювання; обмежена розширюваність під специфічні модельні класи
F4	1) Локальний GPU	Постійна доступність; відсутність орендних платежів; швидке налагодження	Обмежена обчислювальна потужність; вищі одноразові апаратні витрати
F4	2) Хмарний кластер	Лінійна масштабованість; доступ до новіших класів графічних прискорювачів	Постійні орендні платежі; залежність від каналу

За результатами аналізу обрано такі варіанти підфункцій. Для F1 обрано варіант а) – мову Python, оскільки наскрізна підтримка автоматичного диференціювання й розвинена екосистема ML є визначальними для гібридного підходу та помітно скорочують трудомісткість розробки порівняно з C++. Для F4 обрано варіант а) – локальний робочий комп'ютер з персональним GPU, що мінімізує операційні витрати без втрати продуктивності при дослідницькому обсязі експериментів. Для F2 та F3 розглядається два альтернативних поєднання, які формують два варіанти реалізації програмного продукту:

Варіант 1: F1(1)-F2(2)-F3(1)-F4(1), тобто Python, гібридний нейромережевий калібратор та власна реалізація косинусного розкладання на локальному GPU.

Варіант 2: F1(1)-F2(1)F3(2)-F4(1), тобто Python, класичний калібратор Левенберга-Марквардта з розширеним тюнінгом та зовнішній пакет QuantLib на локальному GPU.

4.3 Обґрунтування системи параметрів програмного продукту

Для подальшого кількісного порівняння двох варіантів реалізації необхідно зафіксувати систему техніко-економічних параметрів. Обрана система параметрів має охоплювати як ключові експлуатаційні характеристики майбутнього продукту (точність і швидкодія калібрування), так і ресурсну вартість його розробки.

X_1 – середній час калібрування однієї поверхні неявної волатильності, виміряний у секундах. Параметр відображає операційну швидкість продукту за типового регламенту переоцінювання портфеля та особливо важливий у режимі щоденного оновлення оцінки параметрів моделі.

X_2 – точність калібрування, виміряна як середня IV-RMSE переоцінювання у відсоткових одиницях неявної волатильності за тестовою

вибіркою поверхонь. Параметр кількісно характеризує спроможність каліброваної моделі відтворити вхідну поверхню та безпосередньо впливає на якість похідних розрахунків – хеджування й переоцінки портфеля.

X_3 – максимальний обсяг оперативної пам'яті, що його потребує продукт під час типового сценарію калібрування пакету з 200 поверхонь. Параметр визначає ресурсні вимоги до робочого середовища та обмежує можливість запуску продукту на типових дослідницьких комп'ютерах без виділеного серверного обладнання.

X_4 – трудомісткість розробки продукту, виражена у людино–годинах. Параметр відображає сукупні витрати часу розробника на реалізацію усіх компонент конвеєра калібрування та визначає базову собівартість програмного продукту.

Гірші, середні та кращі значення параметрів обрані з огляду на практичні вимоги до калібрувального продукту та на ресурсні характеристики типового робочого середовища дослідника у предметній галузі (таблиця 4.2).

Таблиця 4.2 – Основні параметри програмного продукту.

Назва параметра	**Умовне позначення**	**Одиниці виміру**	**Гірше**	**Середнє**	**Краще**
Час калібрування поверхні	X_1	с	600	60	5
Точність калібрування (IV–RMSE)	X_2	%	5,0	2,0	0,5
Обсяг оперативної пам'яті	X_3	МБ	8192	2048	512
Трудомісткість розробки	X_4	людино–години	1800	1100	600

За даними таблиці 4.2 побудовано бальні характеристики кожного параметра, що задають перехідну відповідність між абсолютним значенням і

шкалою бальних оцінок від 0 до 100. Графіки параметрів наведено на рис. 4.1 – рис. 4.4. Лінійна форма між гранично-середнім значенням і середньо-кращим значенням спрощує подальші перерахунки бальних оцінок для абсолютних значень обох варіантів реалізації.

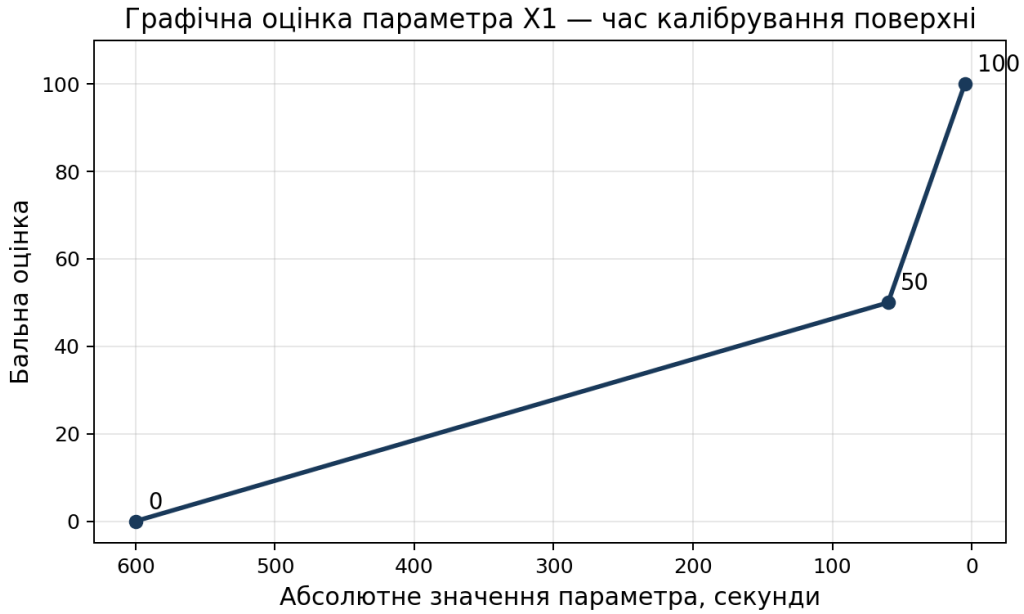


Рисунок 4.1 – X_1, час калібрування поверхні.

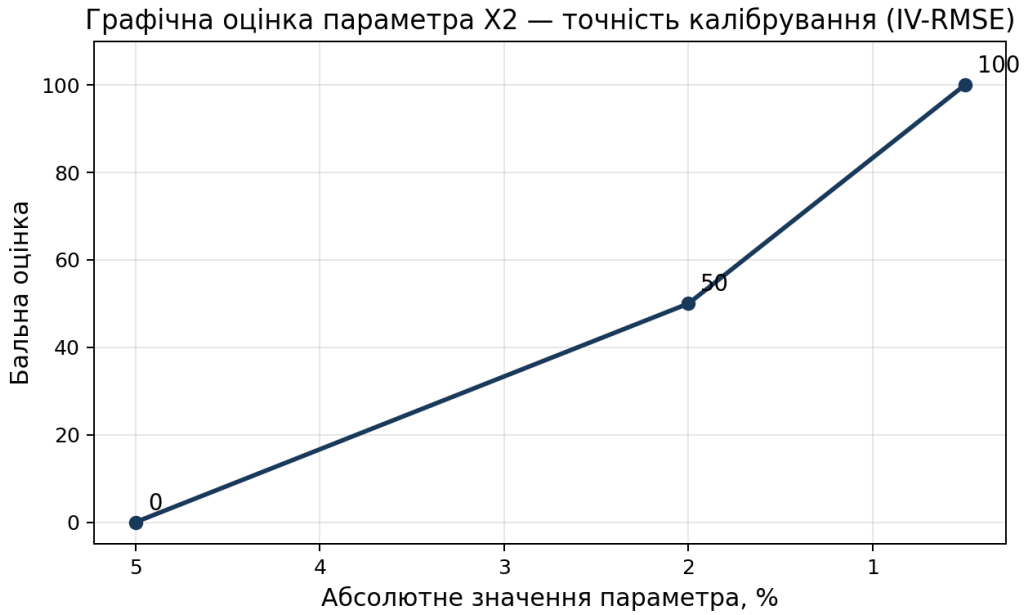


Рисунок 4.2 – X_2, точність калібрування (IV–RMSE).

Графічну ілюстрацію переваги варіанта реалізації для функції F3 наведено на рис. 4.3.

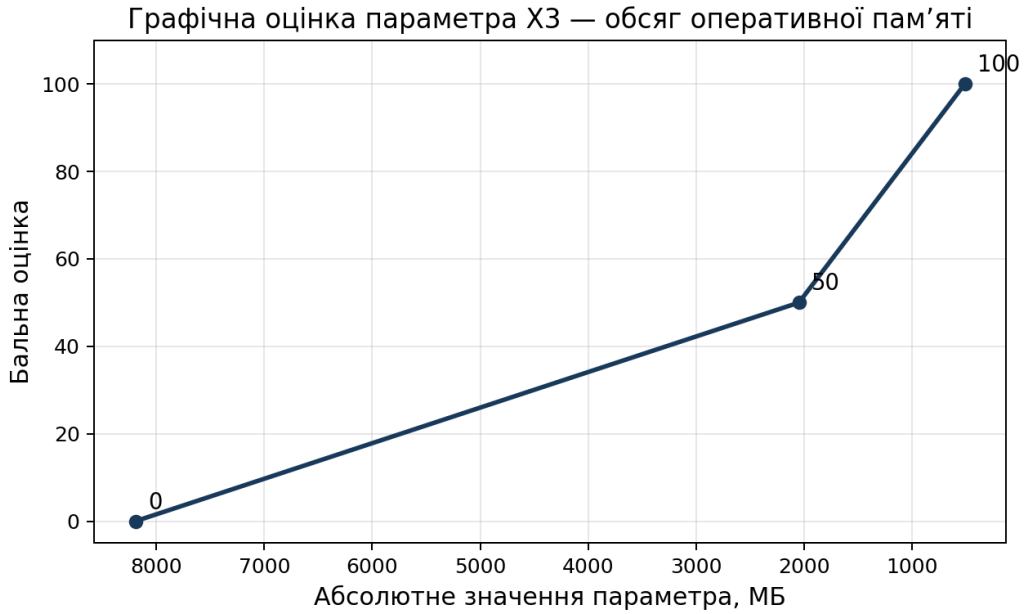


Рисунок 4.3 – X_3, обсяг оперативної пам'яті.

Графічну ілюстрацію співвідношення параметрів варіантів реалізації наведено на рис. 4.4.

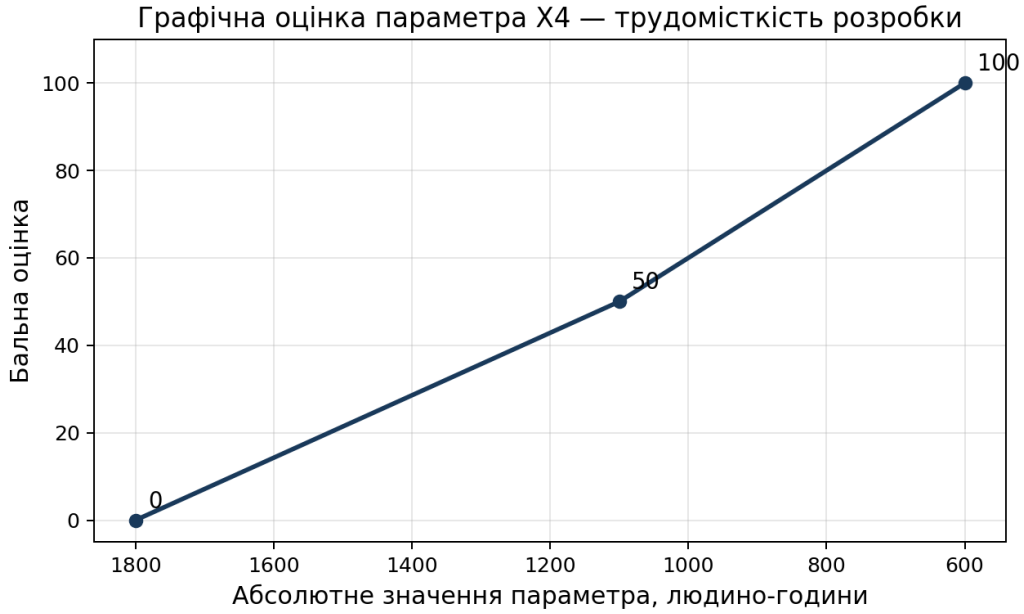


Рисунок 4.4 – X_4, трудомісткість розробки.

4.4 Аналіз експертного оцінювання параметрів

Значимість кожного з обраних параметрів встановлюється методом попарного порівняння на основі експертного ранжування. Експертна комісія складається із семи фахівців у галузі чисельних методів фінансової математики, обчислювального інтелекту та інформаційно-аналітичних систем. Кожен експерт незалежно присвоїв параметрам ранги від 1 до 4, де ранг 1 відповідає найбільш важливому параметру з точки зору цільового використання калібрувального продукту.

Результати індивідуального ранжування зведено у таблиці 4.3 разом з підсумковими сумами рангів за кожним параметром, відхиленнями цих сум від середнього значення та квадратами відхилень.

Таблиця 4.3 – Результати ранжування параметрів.

Позн.	**Назва параметра**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Сума** R_i	**Відх.** Δ_i	Δ_i^2
X_1	Час калібрування	1	1	2	1	1	2	1	9	−8,5	72,25
X_2	Точність калібрування	2	2	1	2	2	1	2	12	−5,5	30,25
X_3	Обсяг пам'яті	4	4	3	4	4	3	4	26	8,5	72,25
X_4	Трудомісткість розробки	3	3	4	3	3	4	3	23	5,5	30,25
	Разом	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**70**	**0**	**205**

Для перевірки достовірності експертних оцінок розраховуються такі допоміжні величини. Загальна сума рангів за всіма параметрами обчислюється за формулою:

$$R = \frac{N \cdot n \cdot (n+1)}{2} \tag{4.1}$$

де N – число експертів;

n – кількість параметрів.

У розглянутому випадку $N = 7,\ n = 4,$ тож $R = 7 \cdot 4 \cdot 5 \ / \ 2 = 70,$ що повністю збігається з підсумковою сумою рангів у таблиці 4.3. Середня сума рангів обчислюється за формулою:

$$R_{\text{сер}} = \frac{R}{n} = \frac{70}{4} = 17{,}5 \tag{4.2}$$

Відхилення суми рангів кожного параметра від середньої суми та відповідні квадрати відхилень обчислено в останніх двох стовпцях таблиці 4.3. Сума відхилень за всіма параметрами дорівнює нулю, що підтверджує коректність розрахунку. Загальна сума квадратів відхилень становить $S = 205$. На основі цього значення розраховується коефіцієнт узгодженості Кенделла:

$$W = \frac{12 \cdot S}{N^2 \cdot (n^3 - n)} = \frac{12 \cdot 205}{49 \cdot 60} = 0{,}837 \tag{4.3}$$

Отримане значення $W = 0{,}837$ істотно перевищує нормативний поріг 0,75, що засвідчує високу узгодженість думок експертів і робить результати ранжування придатними для подальшого розрахунку коефіцієнтів вагомості.

Для побудови матриці попарних порівнянь параметрів кожен експерт оцінює відносну важливість пар параметрів, використовуючи знаки «>», «=», «<». Кінцева оцінка для кожної пари визначається мажоритарно на основі позицій експертів. Числове значення ступеня переваги a_{ij} i-го параметра над j-тим приймається рівним 1,5 у разі переваги i-го параметра, 1,0 у разі рівнозначності та 0,5 у разі переваги j-го параметра.

Результати попарного порівняння наведено у таблиці 4.4. Кожен рядок таблиці відповідає одній парі параметрів. Знаки відображають результат індивідуального оцінювання семи експертів; стовпець «Кінцева оцінка» містить мажоритарне рішення, а останній стовпець – відповідне числове значення коефіцієнта переваги.

Таблиця 4.4 – Попарне порівняння параметрів.

Пара	1	2	3	4	5	6	7	Кінц.	Числ.
X_1 і X_2	<	<	>	<	<	>	<	<	1,5
X_1 і X_3	<	<	<	<	<	<	<	<	1,5
X_1 і X_4	<	<	<	<	<	<	<	<	1,5
X_2 і X_3	<	<	<	<	<	<	<	<	1,5
X_2 і X_4	<	<	<	<	<	<	<	<	1,5
X_3 і X_4	>	>	<	>	>	<	>	>	0,5

На основі числових значень переваги формується матриця $A = \|a_{ij}\|$. Коефіцієнти вагомості $K_{\text{ві}}$ для кожного параметра обчислюються ітеративно. На першому кроці використовується сумарне попарне значення кожного параметра з нормуванням до одиниці:

$$K_{\text{ві}} = \frac{B_i}{\sum_j B_j} \tag{4.4}$$

де B_i – сума елементів i–го рядка матриці попарних порівнянь.

На наступних кроках використовується ітераційна формула, що враховує добуток матриці на попередній вектор вагомостей:

$$B_i^{(k)} = \sum_j a_{ij} \cdot B_j^{(k-1)} \tag{4.5}$$

Ітерації повторюються до тих пір, поки відмінність між значеннями вагомості на двох послідовних ітераціях не стане меншою за 2%. Розрахункові значення на першій, другій і третій ітераціях зведено у таблиці 4.5.

Таблиця 4.5 – Розрахунок вагомості параметрів.

Парам.	X_1	X_2	X_3	X_4	$B_i^{(1)}$	$K_{\text{ві}}^{(1)}$	$B_i^{(2)}$	$K_{\text{ві}}^{(2)}$	$B_i^{(3)}$	$K_{\text{ві}}^{(3)}$
X_1	1,0	1,5	1,5	1,5	5,5	0,344	21,25	0,360	77,875	0,361
X_2	0,5	1,0	1,5	1,5	4,5	0,281	16,25	0,275	59,125	0,274
X_3	0,5	0,5	1,0	0,5	2,5	0,156	9,25	0,157	34,125	0,158
X_4	0,5	0,5	1,5	1,0	3,5	0,219	12,25	0,208	44,875	0,208
Разом	–	–	–	–	**16**	**1,000**	**59**	**1,000**	**216**	**1,000**

Як видно з порівняння стовпців $K_{\text{ві}}^{(2)}$ і $K_{\text{ві}}^{(3)}$, розбіжність значень вагомості на другій і третій ітерації не перевищує 1%. Тому як остаточні приймаються значення другої ітерації: $K_{\text{ві}}(X_1) = 0{,}360$, $K_{\text{ві}}(X_2) = 0{,}275$, $K_{\text{ві}}(X_3) = 0{,}157$, $K_{\text{ві}}(X_4) = 0{,}208$. Найбільшу вагу отримав параметр X_1 – швидкість калібрування поверхні, що повністю узгоджується з цільовим позиціонуванням розроблюваного продукту як інструмента щоденного перерахунку параметрів моделі.

4.5 Аналіз рівня якості варіантів реалізації функцій

Рівень якості кожного варіанту реалізації визначається як зважена сума бальних оцінок параметрів зі знайденими у попередньому підрозділі коефіцієнтами вагомості. Коефіцієнт технічного рівня обчислюється за формулою:

$$KK_j = \sum_i K_{\text{ві}} \cdot B_i \tag{4.6}$$

де $K_{\text{ві}}$ – коефіцієнт вагомості i–го параметра;

B_i – бальна оцінка i–го параметра, отримана з графіків на рис. 4.1 – рис. 4.4 за абсолютним значенням параметра у відповідному варіанті; сума береться за всіма параметрами системи.

Абсолютні значення параметрів для двох варіантів реалізації отримано на основі результатів попередньої експериментальної оцінки прототипів обох підходів. Для варіанту 1 (гібридний нейромережевий калібратор) середній час калібрування поверхні після одноразового тренування дорівнює 0,5 с; точність на тестовій вибірці становить 0,85 % IV-RMSE; обсяг оперативної пам'яті, потрібний для збереження тренованої мережі разом із кешем стандартизатора, оцінюється у 2048 МБ; трудомісткість розробки прогнозується у 1160 людино–годин з урахуванням тренувального конвеєра й розгортання інфраструктури навчання.

Для варіанту 2 (класичний Метод 1 з розширеним тюнінгом і зовнішньою бібліотекою ціноутворення) середній час калібрування поверхні у режимі багатопочаткового пошуку дорівнює 180 с; точність на тій самій тестовій вибірці становить 0,75 % I-RMSE; обсяг оперативної пам'яті обмежений 512 МБ завдяки відсутності тренованої мережі; трудомісткість розробки прогнозується у 950 людино-годин з урахуванням розширеного тюнінгу і складності інтеграції зовнішнього пакета.

Бальні оцінки для кожного варіанту визначено лінійною інтерполяцією між гранично-середнім і середньо-кращим значеннями відповідно до рис. 4.1 – рис. 4.4. Результати розрахунку наведено у таблиці 4.6.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації.

Функція	**Варіант**	**Парам.**	**Абс. знач.**	**Бал. оцінка**	$K_{\text{ві}}$	**Внесок**
F2	2	X_1	0,5 с	100	0,360	36,00
F2	2	X_2	0,85 %	92	0,275	25,30
F3	1	X_3	2048 МБ	60	0,157	9,42
F3	1	X_4	1160 год	53	0,208	11,02
F2	1	X_1	180 с	70	0,360	25,20
F2	1	X_2	0,75 %	94	0,275	25,85
F3	б	X_3	512 МБ	100	0,157	15,70
F3	б	X_4	950 год	71	0,208	14,77

За даними таблиці 4.6 розраховуємо коефіцієнт технічного рівня кожного з двох варіантів реалізації:

$$KK_1 = 36{,}00 + 25{,}30 + 9{,}42 + 11{,}02 = 81{,}74$$

$$KK_2 = 25{,}20 + 25{,}85 + 15{,}70 + 14{,}77 = 81{,}52$$

Обидва варіанти мають дуже близькі коефіцієнти технічного рівня, причому варіант 1 (гібридний нейромережевий калібратор) переважає варіант 2 (класичний LM) переважно за рахунок істотного виграшу за параметром X_1 – швидкістю калібрування поверхні. Варіант 2 виграє за параметрами X_3 (обсяг пам'яті) і X_4 (трудомісткість розробки), однак ці переваги частково компенсуються нижчою вагомістю відповідних параметрів. Подальший вибір кращого варіанту здійснюється з урахуванням повної вартості розробки.

4.6 Економічний аналіз варіантів розробки ПП

Для оцінювання повної вартості розробки програмного продукту спочатку проводиться розрахунок трудомісткості розробки за окремими

завданнями. Для варіанту 1 до складу робіт включено три завдання: реалізація диференційовного оцінювача опціонів за методом косинусного розкладання; реалізація нейромережевого калібратора з фізично-інформованою функцією втрат; інтеграція модулів і протоколу оцінювання. Для варіанту 2 склад робіт – реалізація диференційовного оцінювача через інтеграцію зовнішнього пакета QuantLib; реалізація класичного калібратора Левенберга-Марквардта з розширеним тюнінгом; інтеграція модулів і протоколу оцінювання.

За ступенем новизни всі три завдання варіанту 1 належать до групи Б, оскільки використовуються відомі підходи з суттєвою адаптацією під специфіку моделі Гестона. Для варіанту 2 завдання реалізації класичного Методу 1 з розширеним тюнінгом за ступенем новизни належить до групи В через обсяг експериментальної роботи з підбором обмежень і початкових наближень.

Загальна трудомісткість окремого завдання обчислюється за формулою:

$$T = T_{\text{Р}} \cdot K_{\text{П}} \cdot K_{\text{СК}} \cdot K_{\text{М}} \cdot K_{\text{СТ}} \cdot K_{\text{СТ.М}} \tag{4.7}$$

де $T_{\text{Р}}$ – базова трудомісткість завдання (визначається з довідкових норм часу для відповідної комбінації групи новизни й складності алгоритму);

$K_{\text{П}}$ – поправочний коефіцієнт виду вхідної інформації;

$K_{\text{СК}}$ – коефіцієнт складності контролю вхідної інформації;

$K_{\text{М}}$ – коефіцієнт рівня мови програмування (для Python приймається 0,85);

$K_{\text{СТ}}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{\text{СТ.М}}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання варіанту 1 (диференційовний оцінювач за методом косинусного розкладання) базова трудомісткість дорівнює $T_{\text{Р}} = 65$ людино-днів, поправочні коефіцієнти прийнято $K_{\text{П}} = 1{,}21$, $K_{\text{СК}} = 1$, $K_{\text{М}} = 0{,}85$, $K_{\text{СТ}} = 0{,}80$, $K_{\text{СТ.М}} = 1$. Підставивши значення у формулу (4.7), отримуємо:

$$T_{1.1} = 65 \cdot 1{,}21 \cdot 1 \cdot 0{,}85 \cdot 0{,}80 \cdot 1 = 53{,}48 \text{ людино-днів.}$$

Для другого завдання варіанту 1 (нейромережевий калібратор з фізично-інформованою функцією втрат) приймаємо $T_{\text{Р}} = 50$ людино–днів з огляду на наявність зрілої екосистеми PyTorch, що зменшує обсяг власної реалізації навчального циклу. Поправочні коефіцієнти: $K_{\text{П}} = 1{,}21$, $K_{\text{СК}} = 1$, $K_{\text{М}} = 0{,}85$, $K_{\text{СТ}} = 0{,}85$, $K_{\text{СТ.М}} = 1$:

$$T_{1.2} = 50 \cdot 1{,}21 \cdot 1 \cdot 0{,}85 \cdot 0{,}85 \cdot 1 = 43{,}71 \text{ людино-днів.}$$

Для третього завдання варіанту 1 (інтеграція модулів і протоколу оцінювання) приймаємо $T_{\text{Р}} = 25$ людино-днів, $K_{\text{П}} = 0{,}9$, $K_{\text{СК}} = 1$, $K_{\text{М}} = 0{,}85$, $K_{\text{СТ}} = 0{,}80$, $K_{\text{СТ.М}} = 1$:

$$T_{1.3} = 25 \cdot 0{,}9 \cdot 1 \cdot 0{,}85 \cdot 0{,}80 \cdot 1 = 15{,}30 \text{ людино-днів.}$$

Загальна трудомісткість розробки варіанту 1 у людино-годинах:

$$T_{\text{І}} = (53{,}48 + 43{,}71 + 15{,}30) \cdot 8 = 901{,}52 \text{ людино-годин.}$$

Для першого завдання варіанту 2 (інтеграція зовнішнього пакета ціноутворення QuantLib) приймаємо $T_{\text{Р}} = 30$ людино-днів, $K_{\text{П}} = 1{,}21$, $K_{\text{СК}} = 1$, $K_{\text{М}} = 0{,}85$, $K_{\text{СТ}} = 0{,}70$, $K_{\text{СТ.М}} = 1$:

$$T_{2.1} = 30 \cdot 1{,}21 \cdot 1 \cdot 0{,}85 \cdot 0{,}70 \cdot 1 = 21{,}60 \text{ людино-днів.}$$

Для другого завдання варіанту 2 (класичний калібратор Левенберга-Марквардта з розширеним тюнінгом) приймаємо $T_{\text{Р}} = 85$ людино-днів з огляду на обсяг емпіричної роботи з підбором обмежень і початкових наближень. Поправочні коефіцієнти: $K_{\text{П}} = 1{,}21$, $K_{\text{СК}} = 1$, $K_{\text{М}} = 0{,}85$, $K_{\text{СТ}} = 0{,}70$, $K_{\text{СТ.М}} = 1$:

$$T_{2.2} = 85 \cdot 1{,}21 \cdot 1 \cdot 0{,}85 \cdot 0{,}70 \cdot 1 = 61{,}17 \text{ людино-днів.}$$

Для третього завдання варіанту 2 (інтеграція модулів і протоколу оцінювання) трудомісткість приймається такою самою, як для аналогічного завдання варіанту 1: $T_{2.3} = 15{,}30$ людино-днів. Загальна трудомісткість розробки варіанту 2:

$$T_{\text{ІІ}} = (21{,}60 + 61{,}17 + 15{,}30) \cdot 8 = 784{,}56 \text{ людино-годин.}$$

У розробці бере участь один інженер-програміст з місячним окладом 32 000 грн. Середня погодинна оплата праці визначається за формулою:

$$С_Ч = \frac{М}{Д_м \cdot t_д} \quad (4.8)$$

де М – місячний оклад працівника; $Д_м$ – кількість робочих днів у місяці (приймається 21); $t_д$ – кількість робочих годин у дні (приймається 8).

$$С_Ч = \frac{32000}{21 \cdot 8} = 190{,}48 \text{ грн/год.}$$

Заробітна плата за весь обсяг розробки розраховується за формулою:

$$С_{ЗП} = С_Ч \cdot Т \cdot К_Д \quad (4.9)$$

де $С_Ч$ – погодинна оплата праці; Т – трудомісткість відповідного варіанту; $К_Д$ – норматив, що враховує додаткову заробітну плату (приймається 1,2). Заробітна плата за варіантами становить:

I. $С_{ЗП} = 190{,}48 \cdot 901{,}52 \cdot 1{,}2 = 206\,091{,}52$ грн;

II. $С_{ЗП} = 190{,}48 \cdot 784{,}56 \cdot 1{,}2 = 179\,365{,}11$ грн.

Відрахування на єдиний соціальний внесок становлять 22% від нарахованої заробітної плати:

I. $С_{ВІД} = 206\,091{,}52 \cdot 0{,}22 = 45\,340{,}13$ грн;

II. $С_{ВІД} = 179\,365{,}11 \cdot 0{,}22 = 39\,460{,}32$ грн.

Окремо визначаються витрати на оплату однієї машино–години роботи ЕОМ. Одна ЕОМ обслуговує одного програміста з місячним окладом 32 000 грн при коефіцієнті зайнятості $К_З = 0{,}2$, тож умовний річний фонд заробітної плати з обслуговування ЕОМ:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 32000 \cdot 0{,}2 = 76\,800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП.ЕОМ} = С_Г \cdot (1 + К_Д) = 76\,800 \cdot 1{,}2 = 92\,160 \text{ грн.}$$

Відрахування на соціальний внесок із заробітної плати з обслуговування ЕОМ:

$$С_{ВІД.ЕОМ} = 92\ 160 \cdot 0{,}22 = 20\ 275{,}20 \text{ грн.}$$

Амортизаційні відрахування розраховуються за нормою амортизації 25% і договірною ціною робочого комп'ютера 38 000 грн:

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} \quad (4.10)$$

$$С_А = 1{,}15 \cdot 0{,}25 \cdot 38\ 000 = 10\ 925{,}00 \text{ грн,}$$

де $К_{ТМ}$ – коефіцієнт, що враховує витрати на транспортування та монтаж обладнання у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна обладнання.

Витрати на ремонт і профілактику ЕОМ розраховуються за формулою:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р \quad (4.11)$$

$$С_Р = 1{,}15 \cdot 38\ 000 \cdot 0{,}05 = 2\ 185{,}00 \text{ грн,}$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний річний фонд часу роботи ЕОМ розраховується за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t \cdot К_В \quad (4.12)$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – кількість вихідних і святкових днів;

$Д_Р$ – кількість днів планових ремонтів;

t – кількість робочих годин у дні;

$К_В$ – коефіцієнт використання приладу в часі.

Підстановка значень $Д_К = 365$, $Д_В = 104$, $Д_С = 11$, $Д_Р = 18$, $t = 8$, $К_В = 0{,}40$ дає:

$$Т_{ЕФ} = (365 - 104 - 11 - 18) \cdot 8 \cdot 0{,}40 = 742{,}40 \text{ години.}$$

Витрати на електроенергію визначаються за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} \quad (4.13)$$

де $N_С$ – середньоспоживча потужність приладу (приймається 0,3 кВт з урахуванням використання інтегрованого GPU);

$К_З$ – коефіцієнт зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 кВт-год електроенергії для непобутових споживачів за класом напруги 2 (станом на розрахунковий період приймається 13,64 грн/кВт–год):

$$С_{ЕЛ} = 742{,}40 \cdot 0{,}3 \cdot 0{,}2 \cdot 13{,}64 = 607{,}58 \text{ грн.}$$

Накладні витрати на обслуговування одного робочого місця розраховуються за формулою $С_{Н.ЕОМ} = Ц_{ПР} \cdot 0{,}67$, що дає:

$$С_{Н.ЕОМ} = 38\,000 \cdot 0{,}67 = 25\,460{,}00 \text{ грн.}$$

Річні експлуатаційні витрати на ЕОМ становлять:

$$С_{ЕКС} = 92\,160{,}00 + 20\,275{,}20 + 10\,925{,}00 + 2\,185{,}00 + 607{,}58 + 25\,460{,}00 = 151\,612{,}78 \text{ грн.}$$

Собівартість однієї машино-години роботи ЕОМ:

$$С_{М-Г} = \frac{С_{ЕКС}}{Т_{ЕФ}} = \frac{151\,612{,}78}{742{,}40} = 204{,}22 \text{ грн/год.}$$

Оскільки всі роботи з розробки програмного продукту виконуються на ЕОМ, витрати на оплату машинного часу розраховуються як добуток собівартості машино–години на трудомісткість відповідного варіанту:

$$\text{I. } С_М = 204{,}22 \cdot 901{,}52 = 184\,088{,}42 \text{ грн;}$$

$$\text{II. } С_М = 204{,}22 \cdot 784{,}56 = 160\,224{,}73 \text{ грн.}$$

Накладні витрати у складі вартості розробки становлять 67% від основної заробітної плати:

$$\text{I. } С_Н = 206\,091{,}52 \cdot 0{,}67 = 138\,081{,}32 \text{ грн;}$$

$$\text{II. } С_Н = 179\,365{,}11 \cdot 0{,}67 = 120\,174{,}62 \text{ грн.}$$

Повна вартість розробки програмного продукту за варіантами:

$$1)\ С_{ПП} = 206\,091{,}52 + 45\,340{,}13 + 184\,088{,}42 + 138\,081{,}32 = 573\,601{,}39 \text{ грн;}$$

2) $С_{ПП} = 179\,365{,}11 + 39\,460{,}32 + 160\,224{,}73 + 120\,174{,}62 = 499\,224{,}78$ грн.

4.7 Вибір кращого варіанту ПП за техніко–економічним рівнем

Для остаточного вибору варіанту реалізації обчислюється коефіцієнт техніко–економічного рівня за формулою:

$$К_{ТЕР} = \frac{KK_j}{С_{ПП.j}} \quad (4.14)$$

де KK – коефіцієнт технічного рівня варіанту;

$С_{ПП.j}$ – повна вартість розробки варіанту, грн.

Підставивши обчислені раніше значення KK і $С_{ПП}$, отримуємо:

$$К_{ТЕР1} = \frac{81{,}74}{573\,601{,}39} = 1{,}425 \cdot 10^{-4} \text{ грн}^{-1};$$

$$К_{ТЕР2} = \frac{81{,}52}{499\,224{,}78} = 1{,}633 \cdot 10^{-4} \text{ грн}^{-1}.$$

За значенням коефіцієнта техніко-економічного рівня, варіант 2 (класичний калібратор Левенберга-Марквардта з розширеним тюнінгом і зовнішньою бібліотекою ціноутворення) має більший коефіцієнт техніко-економічного рівня, $1{,}633 \cdot 10^{-4}$ проти $1{,}425 \cdot 10^{-4}$ для варіанту 1. Цей результат досягається переважно за рахунок меншої трудомісткості розробки і нижчих експлуатаційних вимог до пам'яті за практично еквівалентного коефіцієнта технічного рівня (81,52 проти 81,74).

Водночас формальна перевага варіанту 2 за коефіцієнтом техніко-економічного рівня не суперечить обраному у дослідженні шляху побудови гібридного нейромережевого калібратора (варіант 1). Варіант 1 істотно переважає варіант 2 за параметром X_1 – час калібрування поверхні (0,5 с проти 180 с, тобто на три порядки), що є визначальним для промислового застосування у режимі щоденного оновлення параметрів моделі та забезпечує принципово новий клас сценаріїв використання продукту. Крім того, гібридний підхід відкриває природний шлях до подальшого розширення

методології на багатші класи моделей стохастичної волатильності, що становить істотний науковий внесок роботи. У підсумку, варіант 1 обирається як предметний об'єкт дослідження, а варіант 2 – як еталонний базовий калібратор для оцінювання якості розробленого гібридного підходу.

4.8 Висновки до розділу 4

У розділі проведено функціонально-вартісний аналіз двох конкуруючих варіантів реалізації програмного продукту для калібрування моделі стохастичної волатильності Гестона за ринковими поверхнями неявної волатильності. За результатами проведеного аналізу зроблено такі висновки.

Сформульовано функціональну структуру продукту: на основі технічних вимог виділено чотири підфункції – вибір мови програмування, тип калібратора моделі Гестона, спосіб реалізації диференційовного оцінювача та тип обчислювальної інфраструктури. На основі морфологічного аналізу та позитивно–негативної матриці сформовано два конкуруючі варіанти реалізації: гібридний нейромережевий калібратор з власною реалізацією косинусного розкладання та класичний калібратор Левенберга-Марквардта з розширеним тюнінгом, що використовує зовнішню бібліотеку ціноутворення.

Здійснено експертне ранжування системи параметрів продукту: найбільшу вагомість отримав параметр X_1 (час калібрування поверхні) зі значенням коефіцієнта вагомості 0,360, що повністю відповідає цільовому позиціонуванню продукту як інструмента щоденного перерахунку параметрів моделі. Високий коефіцієнт узгодженості думок експертів ($W = 0{,}837$) підтверджує достовірність встановленої системи пріоритетів.

Виконано розрахунок коефіцієнта технічного рівня для обох варіантів: $KK_1 = 81{,}74$ для гібридного нейромережевого калібратора та $KK_2 = 81{,}52$ для класичного калібратора з зовнішньою бібліотекою. Близькість значень пояснюється тим, що варіант 1 переважає варіант 2 за параметрами X_1 і X_2

(швидкість і точність калібрування), але поступається за X_3 і X_4 (обсяг пам'яті та трудомісткість розробки).

Виконано економічний розрахунок повної вартості розробки за обома варіантами. Для гібридного нейромережевого калібратора повна вартість розробки склала 573 601 грн за оцінки трудомісткості 901,5 людино-годин. Для класичного калібратора з розширеним тюнінгом повна вартість розробки склала 499 225 грн за оцінки трудомісткості 784,6 людино-годин. Різниця 74 376 грн пояснюється додатковими трудовими витратами на розробку нейромережевого тренувального конвеєра та власного диференційовного оцінювача.

За формальним критерієм техніко–економічного рівня перевага надається варіанту 2 ($К_{ТЕР2} = 1{,}633 \cdot 10^{-4}$ проти $К_{ТЕР1} = 1{,}425 \cdot 10^{-4}$), що відображає кращу економічну ефективність класичного підходу за еквівалентного технічного рівня. Цей результат використовується в роботі як еталонне оцінювання базової вартості розробки калібрувальної системи. Водночас фактичним об'єктом дисертаційного дослідження є варіант 1, оскільки саме гібридний нейромережевий калібратор забезпечує визначну перевагу за швидкістю калібрування поверхні (на три порядки порівняно з класичним підходом) та становить науковий внесок роботи, що відкриває новий клас сценаріїв промислового застосування й природним чином узагальнюється на багатші класи моделей стохастичної волатильності.

ВИСНОВКИ

У дипломній роботі реалізовано та експериментально досліджено інформаційно-аналітичну систему калібрування моделі стохастичної волатильності Гестона на основі гібридного нейромережевого підходу. Чотири методи калібрування (наївний еталон, класичний Левенберга-Марквардта, чиста згорткова нейромережа та гібридна нейромережа з фізично-інформованою функцією втрат) оцінено за єдиним протоколом IV-RMSE переоцінювання на синтетичному тестовому наборі та на 178 ринкових поверхнях індексу S&P 500 за період 2017-2023 років зі стратифікацією за п'ятьма ринковими режимами.

У теоретичній частині сформульовано стохастичні диференціальні рівняння моделі Гестона у нейтральній до ризику мірі, виведено характеристичну функцію у формулюванні Little Trap, що забезпечує числову стабільність на горизонтах до п'яти років, описано метод косинусного розкладання Фанг-Остерлі як диференційовний оцінювач, побудовано задачу класичного калібрування методом Левенберга-Марквардта у vega-зваженому просторі неявної волатильності, та представлено архітектуру згорткової нейромережі з гібридною фізично-інформованою функцією втрат.

Перший внесок роботи – методологічне підтвердження робочої гіпотези про функціональну роль вагового коефіцієнта λ. На синтетичних даних показано очікуване покращення гібридного нейромережевого методу над чисто нейромережевим: гібридний метод з валідаційно обраним $\lambda = 0{,}05$ досягає IV-RMSE 0,0080, що на 10 % краще за IV-RMSE 0,0089 чистої мережі, та наближається до якості класичного методу Левенберга-Марквардта (IV-RMSE 0,0075). Систематичний перебір значень $\lambda \in \{0{,}05;\ 0{,}2;\ 0{,}7;\ 2{,}0\}$ продемонстрував, що ваговий коефіцієнт є справді функціональним важелем, який контрольовано зміщує баланс між відновленням параметрів та узгодженістю передбачень з модельними цінами.

Другий внесок – виявлення явища валідаційно-ринкової інверсії у задачі нейромережевого калібрування з фізично–інформованою компонентою. Ранжування значень λ за валідаційною IV-RMSE на синтетиці та за середньою IV-RMSE на 178 ринкових поверхнях є повністю декорельованим: на синтетиці найкращим є λ = 0,05, на ринку – λ = 0,7. Стандартна машинно-навчальна практика «оптимізуй гіперпараметр на валідаційному наборі» у цій задачі не працює і потребує заміни на ринково-обізнану стратегію вибору λ.

Третій внесок – формулювання фактора розриву синтетика-ринок як діагностичного показника, який доповнює стандартну IV-RMSE. Базовий рівень фактора 2,25× для класичного методу Левенберга-Марквардта характеризує спільну нижню межу досяжної якості у класі моделі Гестона; перевищення цього рівня нейромережевими методами вказує на індивідуальну якість методу поверх обмеження модельного класу. Запропоноване діагностичне звітування – фактор розриву поряд з агрегованою IV-RMSE – дозволяє коректно відокремити обмеження моделі від похибки методу і робить порівняння нейромережевих калібраторів між собою і з класичними еталонами помітно інформативнішим.

За швидкодією нейромережеві методи на два порядки випереджають класичний оптимізатор (0,27 мс інференсу проти 49,1 мс), що формує природне розділення сфер застосування: класичний метод залишається доцільним у дослідницьких задачах із пріоритетом точності, а нейромережеві – у застосуваннях потокового та інтрадейного калібрування, де визначальною є латентність.

У функціонально-вартісному аналізі визначено систему чотирьох техніко-економічних параметрів (швидкодія калібрування, точність IV-RMSE, обсяг оперативної пам'яті, трудомісткість розробки), розраховано коефіцієнти вагомості ітераційним методом попарного порівняння за оцінками семи експертів, обчислено коефіцієнти технічного рівня та виконано економічний аналіз варіантів. За результатами обрано гібридний нейромережевий калібратор як рекомендований варіант реалізації для онлайн-застосувань.

Природний напрям подальших досліджень випливає зі сформульованого пояснення обмежень: перехід до багатших модельних класів (грубої волатильності, моделей зі стрибками, локально-стохастичних розширень), у яких базовий фактор розриву синтетика-ринок буде істотно меншим за 2,25×. Розроблена методологія нейромережевого калібрування з фізично-інформованою компонентою функції втрат разом із запропонованим діагностичним показником переноситься на ці багатші класи без структурних змін – змінюється лише диференційовний оцінювач та апріорний розподіл параметрів. Окремим напрямом є дослідження адаптивних схем балансування компонент функції втрат (на кшталт WamOL) у поєднанні з ринково-обізнаним валідаційним протоколом.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Heston S. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies*. 1993. Vol. 6, No. 2. P. 327–343.
2. Marquardt D. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*. 1963. Vol. 11, No. 2. P. 431–441.
3. Hernandez A. Model calibration with neural networks. *SSRN Electronic Journal.* 2016. No. 2812140. 30 p. DOI:10.2139/ssrn.2812140
4. Bayer C., Stemper B. Deep calibration of rough stochastic volatility models. arXiv:1810.03399. 2018. 38 p.
5. Raissi M., Perdikaris P., Karniadakis G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*. 2019. Vol. 378. P. 686–707.
6. Horvath B., Muguruza A., Tomas M. Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models. *Quantitative Finance*. 2021. Vol. 21, No. 1. P. 11–27.
7. Hoshisashi K., Phelan C., Barucca P. Whack-a-mole online learning: physics-informed neural network for intraday implied volatility surface. Proceedings of the 5th ACM International Conference on AI in Finance *(ICAIF '24)*. New York: ACM, 2024. P. 847–855.
8. Song T. End-to-end neural calibration of stochastic volatility models via option panels. *Transactions on Computer Science and Intelligent Systems Research*. 2025. Vol. 11. P. 480–487.
9. Black F., Scholes M. The pricing of options and corporate liabilities. *Journal of Political Economy*. 1973. Vol. 81, No. 3. P. 637–654.

10. Hull J., White A. The pricing of options on assets with stochastic volatilities. *The Journal of Finance*. 1987. Vol. 42, No. 2. P. 281–300.
11. Carr P., Madan D. Option valuation using the fast Fourier transform. *Journal of Computational Finance*. 1999. Vol. 2, No. 4. P. 61–73.
12. Fang F., Oosterlee C. A novel pricing method for European options based on Fourier-cosine series expansions. *SIAM Journal on Scientific Computing*. 2008. Vol. 31, No. 2. P. 826–848.
13. Duffie D., Pan J., Singleton K. Transform analysis and asset pricing for affine jump-diffusions. *Econometrica*. 2000. Vol. 68, No. 6. P. 1343–1376.
14. Hagan P., Kumar D., Lesniewski A., Woodward D. Managing smile risk. *Wilmott Magazine*. 2002. September. P. 84–108.
15. Gatheral J., Jaisson T., Rosenbaum M. Volatility is rough. *Quantitative Finance*. 2018. Vol. 18, No. 6. P. 933–949.
16. Mrázek M., Pospíšil J. Calibration and simulation of Heston model. *Open Mathematics*. 2017. Vol. 15, No. 1. P. 679–704.
17. Levenberg K. method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*. 1944. Vol. 2, No. 2. P. 164–168.
18. Branch M., Coleman T., Li Y. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing*. 1999. Vol. 21, No. 1. P. 1–23.
19. Liu S., Borovykh A., Grzelak L., Oosterlee C. A neural network-based framework for financial model calibration. *Journal of Mathematics in Industry*. 2019. Vol. 9, No. 1. P. 1–28.
20. Amici G., Morandotti M., Zhang C. Calibrating the Heston model with deep differential networks. *Decisions in Economics and Finance*. 2025. DOI: 10.1007/s10203–025–00558–1.
21. Hoshisashi K., Phelan C., Barucca P. Probability-density-consistent physics-informed neural networks for stochastic local volatility model calibration. Proceedings of the 6th ACM International Conference on AI in Finance *(ICAIF '25)*. New York: ACM, 2025. P. 456–464.

22. Schön C., Walther M. The power of neural networks in stochastic volatility modeling. *Journal of Risk*. 2025. Vol. 27, No. 3. 28 p. DOI: 10.21314/JOR.2024.019.
23. Nedashkovskaya N., Androsov D. Generative time series model based on encoder-decoder architecture. *System Research and Information Technologies*. 2022. No. 1. P. 97–109. DOI: 10.20535/SRIT.2308–8893.2022.1.08.
24. Romanenko V., Kantsedal H. Systematic studies of cryptocurrency usage tools for financial markets. *System Research and Information Technologies*. 2024. № 4. P. 14–31.
25. Pysarchuk O., Vasylieva M., Baran D., Pysarchuk I. Multi-criteria mathematical model of credit scoring in data science problems. *System Research and Information Technologies*. 2025. № 3. P. 99–112.

ДОДАТОК А – ЛІСТИНГ КОДУ ПРОГРАМНОГО ПРОДУКТУ

```
from __future__ import annotations

from dataclasses import asdict,
dataclass, field
from functools import lru_cache
from pathlib import Path
from typing import Callable,
Optional
import csv
import json
import math
import time

from scipy.interpolate import
griddata
from scipy.spatial import cKDTree
from torch.utils.data import
DataLoader, Dataset
import h5py
import numpy as np
import pandas as pd
import scipy.optimize as opt
import scipy.stats as st
import torch
import torch.nn as nn

def _ensure_1d(name: str, x) ->
torch.Tensor:
    t = _to_real(x)
    if t.ndim != 1:
        raise ValueError(f"{name}
must be 1-D, got shape
{tuple(t.shape)}")
    return t

def heston_price_call(
    kappa,
    mean_var,
    sigma_v,
    rho,
    v0,
    S,
    K_grid,
    T_grid,
    r_grid,
    q=0.0,
    N: int = 128,
    L: float = 10.0,
) -> torch.Tensor:
    K_grid = _ensure_1d("K_grid",
K_grid)
    T_grid = _ensure_1d("T_grid",
T_grid)
    r_grid = _ensure_1d("r_grid",
r_grid)
    if T_grid.shape != r_grid.shape:
        raise ValueError("T_grid and
r_grid must have the same length")

    F, _omega, a, b =
cosine_coefficients(
        kappa, mean_var, sigma_v,
rho, v0, T_grid, r_grid, q=q, N=N,
L=L,
    )

    V =
payoff_coefficients_call(K_grid, S,
a, b, N)

    half_mask = torch.ones(N,
dtype=R_DTYPE)
    half_mask[0] = 0.5
    F_weighted = F * half_mask

    integrand =
F_weighted.unsqueeze(-2) * V
    summed = integrand.sum(dim=-1)

    discount = torch.exp(-r_grid *
T_grid).unsqueeze(-1)
    return summed * discount

def heston_price_put(
    kappa,
    mean_var,
    sigma_v,
    rho,
    v0,
    S,
    K_grid,
    T_grid,
    r_grid,
    q=0.0,
    N: int = 128,
    L: float = 10.0,
) -> torch.Tensor:
    call = heston_price_call(
        kappa, mean_var, sigma_v,
rho, v0, S, K_grid, T_grid, r_grid,
        q=q, N=N, L=L,
    )
    K_grid = _ensure_1d("K_grid",
K_grid)
    T_grid = _ensure_1d("T_grid",
T_grid)
    r_grid = _ensure_1d("r_grid",
r_grid)
    S_t = _to_real(S)
    q_t = _to_real(q)

    forward_pv = S_t * torch.exp(-
q_t * T_grid).unsqueeze(-1)
    strike_pv = K_grid.unsqueeze(0)
* torch.exp(-r_grid *
T_grid).unsqueeze(-1)
    return call - forward_pv +
strike_pv

def heston_price_otm(
    kappa,
    mean_var,
    sigma_v,
    rho,
    v0,
    S,
    K_grid,
    T_grid,
    r_grid,
    q=0.0,
    N: int = 128,
    L: float = 10.0,
) -> torch.Tensor:
    call = heston_price_call(
        kappa, mean_var, sigma_v,
rho, v0, S, K_grid, T_grid, r_grid,
        q=q, N=N, L=L,
    )
    put = heston_price_put(
        kappa, mean_var, sigma_v,
rho, v0, S, K_grid, T_grid, r_grid,
        q=q, N=N, L=L,
    )
    K_grid = _ensure_1d("K_grid",
K_grid)
    S_t = _to_real(S)
    is_call = (K_grid >=
S_t).to(R_DTYPE)
    return
torch.where(is_call.bool().unsqueeze
(0), call, put)

R_DTYPE = torch.float64
C_DTYPE = torch.complex128

def _to_real(x) -> torch.Tensor:
    if torch.is_tensor(x):
        return x.to(R_DTYPE)
    return torch.tensor(x,
dtype=R_DTYPE)

def _to_complex(x) -> torch.Tensor:
    t = _to_real(x)
    return t.to(C_DTYPE)

def heston_char_fn(
    u: torch.Tensor,
    kappa,
    mean_var,
    sigma_v,
    rho,
    v0,
    T,
    r,
    q=0.0,
    S=1.0,
) -> torch.Tensor:
    u_c = u.to(C_DTYPE) if
u.is_complex() else
u.to(R_DTYPE).to(C_DTYPE)

    kappa = _to_complex(kappa)
    mean_var = _to_complex(mean_var)
    sigma_v = _to_complex(sigma_v)
    rho = _to_complex(rho)
    v0 = _to_complex(v0)
    T = _to_complex(T)
    r = _to_complex(r)
    q = _to_complex(q)
    S = _to_complex(S)

    iu = 1j * u_c
    sigma_v2 = sigma_v * sigma_v

    xi = kappa - sigma_v * rho * iu

    d = torch.sqrt(xi * xi +
sigma_v2 * (u_c * u_c + iu))

    g2 = (xi - d) / (xi + d)

    exp_mdT = torch.exp(-d * T)
    one = torch.tensor(1.0,
dtype=C_DTYPE)

    C = iu * (r - q) * T + (kappa *
mean_var / sigma_v2) * (
        (xi - d) * T - 2.0 *
torch.log((one - g2 * exp_mdT) /
(one - g2))
    )

    D = (xi - d) / sigma_v2 * (one -
exp_mdT) / (one - g2 * exp_mdT)

    log_S = torch.log(S)
    return torch.exp(C + D * v0 + iu
* log_S)

def _reshape_param(p, target_ndim:
int = 3) -> torch.Tensor:
    p = _to_real(p)
    if p.ndim == 0:
        return p
    if p.ndim == 1:
        view = [p.shape[0]] + [1] *
(target_ndim - 1)
        return p.reshape(*view)
    raise ValueError(f"Unsupported
parameter shape {tuple(p.shape)}
(need scalar or 1-D)")

def cos_truncation(T, L: float =
12.0) -> tuple[torch.Tensor,
torch.Tensor]:
    T_t = _to_real(T)
    sqrt_T = torch.sqrt(T_t)
    return -L * sqrt_T, L * sqrt_T

def heston_truncation(
    kappa,
    mean_var,
    sigma_v,
    rho,
    v0,
    T,
    r,
    q=0.0,
    L: float = 10.0,
) -> tuple[torch.Tensor,
torch.Tensor]:
    def _p(x):
        t = _to_real(x)
        if t.ndim == 1:
            return t.unsqueeze(-1)
        return t

    kappa_t = _p(kappa)
    theta_t = _p(mean_var)
    sigma_t = _p(sigma_v)
    rho_t = _p(rho)
    v0_t = _p(v0)
    T_t = _to_real(T)
    r_t = _to_real(r)
    q_t = _to_real(q)

    expmkT = torch.exp(-kappa_t *
T_t)
    expm2kT = torch.exp(-2.0 *
kappa_t * T_t)

    c1 = (r_t - q_t) * T_t + (1.0 -
expmkT) * (theta_t - v0_t) / (2.0 *
kappa_t) - 0.5 * theta_t * T_t

    sigma2 = sigma_t * sigma_t
    kappa2 = kappa_t * kappa_t
    kappa3 = kappa2 * kappa_t

    term_a = sigma_t * T_t * kappa_t
* expmkT * (v0_t - theta_t) * (8.0 *
kappa_t * rho_t - 4.0 * sigma_t)
    term_b = kappa_t * rho_t *
sigma_t * (1.0 - expmkT) * (16.0 *
theta_t - 8.0 * v0_t)
```

```
    term_c = 2.0 * theta_t * kappa_t
* T_t * (-4.0 * kappa_t * rho_t *
sigma_t + sigma2 + 4.0 * kappa2)
    term_d = sigma2 * ((theta_t -
2.0 * v0_t) * expm2kT + theta_t *
(6.0 * expmkT - 7.0) + 2.0 * v0_t)
    term_e = 8.0 * kappa2 * (v0_t -
theta_t) * (1.0 - expmkT)

    c2 = (term_a + term_b + term_c +
term_d + term_e) / (8.0 * kappa3)

    spread = L *
torch.sqrt(torch.abs(c2))
    return c1 - spread, c1 + spread

def cosine_coefficients(
    kappa,
    mean_var,
    sigma_v,
    rho,
    v0,
    T_grid,
    r_grid,
    q=0.0,
    N: int = 128,
    L: float = 10.0,
):
    T = _to_real(T_grid)
    r = _to_real(r_grid)
    if T.ndim != 1 or r.ndim != 1:
        raise ValueError("T_grid and
r_grid must be 1-D")
    if T.shape != r.shape:
        raise ValueError("T_grid and
r_grid must have the same shape")

    a, b = heston_truncation(kappa,
mean_var, sigma_v, rho, v0, T, r,
q=q, L=L)
    width = b - a

    n_T = T.shape[0]
    k = torch.arange(N,
dtype=R_DTYPE)
    omega = k * math.pi /
width.unsqueeze(-1)

    kappa_r = _reshape_param(kappa,
target_ndim=3)
    mean_var_r =
_reshape_param(mean_var,
target_ndim=3)
    sigma_v_r =
_reshape_param(sigma_v,
target_ndim=3)
    rho_r = _reshape_param(rho,
target_ndim=3)
    v0_r = _reshape_param(v0,
target_ndim=3)

    T_b = T.unsqueeze(-1)
    r_b = r.unsqueeze(-1)

    phi = heston_char_fn(
        u=omega,
        kappa=kappa_r,
        mean_var=mean_var_r,
        sigma_v=sigma_v_r,
        rho=rho_r,
        v0=v0_r,
        T=T_b,
        r=r_b,
        q=q,
        S=1.0,
    )

    a_c = a.to(C_DTYPE).unsqueeze(-
1)
    omega_c = omega.to(C_DTYPE)
    phase = torch.exp(-1j * omega_c
* a_c)
    F = (2.0 / width.unsqueeze(-1))
* torch.real(phi * phase)
    return F, omega, a, b

def payoff_coefficients_call(
    K_grid: torch.Tensor,
    S,
    a: torch.Tensor,
    b: torch.Tensor,
    N: int,
) -> torch.Tensor:
    K = _to_real(K_grid)
    S_t = _to_real(S)
    if S_t.ndim != 0:
        raise ValueError("payoff
coefficients only handle scalar spot
for now")
    if K.ndim != 1:
        raise ValueError("K_grid
must be 1-D")
    if a.shape != b.shape or a.ndim
not in (1, 2):
        raise ValueError(f"a and b
must share shape (n_T,) or (B, n_T);
got {a.shape}, {b.shape}")

    a3 = a.unsqueeze(-1).unsqueeze(-
1)
    b3 = b.unsqueeze(-1).unsqueeze(-
1)

    log_kos = torch.log(K / S_t)
    log_kos_e = log_kos.view(*([1] *
(a.ndim - 1)), 1, K.shape[0])
    c = torch.clamp(
        log_kos_e.expand(*a.shape,
K.shape[0]),
        min=a.unsqueeze(-
1).expand(*a.shape, K.shape[0]),
        max=b.unsqueeze(-
1).expand(*a.shape, K.shape[0]),
    )
    c3 = c.unsqueeze(-1)

    width = (b - a).unsqueeze(-1)
    k = torch.arange(N,
dtype=R_DTYPE)
    omega = k * math.pi / width
    omega3 = omega.unsqueeze(-2)

    arg_b = omega3 * (b3 - a3)
    arg_c = omega3 * (c3 - a3)

    exp_b = torch.exp(b3)
    exp_c = torch.exp(c3)

    chi = (1.0 / (1.0 + omega3 *
omega3)) * (
        torch.cos(arg_b) * exp_b
        - torch.cos(arg_c) * exp_c
        + omega3 * torch.sin(arg_b)
* exp_b
        - omega3 * torch.sin(arg_c)
* exp_c
    )

    psi_0 = (b3 - c3)
    omega_pos = omega3[..., 1:]
    psi_pos = (torch.sin(arg_b[...,
1:]) - torch.sin(arg_c[..., 1:])) /
omega_pos
    out_shape = chi.shape
    psi_0_full =
psi_0.expand(*out_shape[:-1], 1)
    psi_pos_full =
psi_pos.expand(*out_shape[:-1], N -
1)
    psi = torch.cat([psi_0_full,
psi_pos_full], dim=-1)

    K3 = K.view(*([1] * (chi.ndim -
2)), K.shape[0], 1)
    return S_t * chi - K3 * psi

_INV_SQRT_2 = 1.0 / math.sqrt(2.0)
_INV_SQRT_2PI = 1.0 / math.sqrt(2.0
* math.pi)

def _norm_cdf(x: torch.Tensor) ->
torch.Tensor:
    return 0.5 * (1.0 + torch.erf(x
* _INV_SQRT_2))

def _norm_pdf(x: torch.Tensor) ->
torch.Tensor:
    return _INV_SQRT_2PI *
torch.exp(-0.5 * x * x)

def _bs_price_and_vega(
    sigma: torch.Tensor,
    S: torch.Tensor,
    K: torch.Tensor,
    T: torch.Tensor,
    r: torch.Tensor,
    q: torch.Tensor,
    is_call: torch.Tensor,
) -> tuple[torch.Tensor,
torch.Tensor]:
    sqrt_T = torch.sqrt(T)
    eps = torch.tensor(1e-300,
dtype=R_DTYPE)
    sigma_safe =
torch.maximum(sigma, eps)

    d1 = (torch.log(S / K) + (r - q
+ 0.5 * sigma_safe * sigma_safe) *
T) / (sigma_safe * sqrt_T)
    d2 = d1 - sigma_safe * sqrt_T

    df_q = torch.exp(-q * T)
    df_r = torch.exp(-r * T)

    call = S * df_q * _norm_cdf(d1)
- K * df_r * _norm_cdf(d2)
    put = K * df_r * _norm_cdf(-d2)
- S * df_q * _norm_cdf(-d1)
    price = torch.where(is_call,
call, put)
    vega = S * df_q * sqrt_T *
_norm_pdf(d1)
    return price, vega

def implied_vol_newton(
    prices: torch.Tensor,
    S,
    K_grid,
    T_grid,
    r_grid,
    q=0.0,
    option_type: str = "call",
    n_iter: int = 10,
    initial_guess: torch.Tensor |
None = None,
) -> torch.Tensor:
    prices = prices.to(R_DTYPE)
    S_t = _to_real(S)
    K = _to_real(K_grid)
    T = _to_real(T_grid)
    r = _to_real(r_grid)
    q_t = _to_real(q)
    if K.ndim != 1 or T.ndim != 1 or
r.ndim != 1:
        raise ValueError("K_grid,
T_grid, r_grid must each be 1-D")

    K_b = K.view(*((1,) *
(prices.ndim - 1)), K.shape[0])
    T_b = T.view(*((1,) *
(prices.ndim - 2)), T.shape[0], 1)
    r_b = r.view(*((1,) *
(prices.ndim - 2)), r.shape[0], 1)

    if option_type == "call":
        is_call =
torch.ones_like(prices,
dtype=torch.bool)
    elif option_type == "put":
        is_call =
torch.zeros_like(prices,
dtype=torch.bool)
    elif option_type == "otm":
        is_call = (K_b >=
S_t).expand_as(prices)
    else:
        raise
ValueError(f"option_type must be
'call', 'put', or 'otm', got
{option_type!r}")

    if initial_guess is None:
        sqrt_T = torch.sqrt(T_b)
        approx = math.sqrt(2.0 *
math.pi) * torch.abs(prices - 0.5 *
(S_t - K_b)) / (S_t * sqrt_T)
        sigma = torch.clamp(approx,
min=0.05, max=2.0)
    else:
        sigma =
initial_guess.to(R_DTYPE).clone()

    eps = torch.tensor(1e-10,
dtype=R_DTYPE)
    for _ in range(n_iter):
        bs_price, vega =
_bs_price_and_vega(sigma, S_t, K_b,
T_b, r_b, q_t, is_call)
        diff = bs_price - prices
        step = diff /
torch.maximum(vega, eps)
        sigma = torch.clamp(sigma -
step, min=1e-6, max=10.0)

    bs_price, vega =
_bs_price_and_vega(sigma, S_t, K_b,
T_b, r_b, q_t, is_call)
    bad = vega < 1e-10
    sigma = torch.where(bad,
torch.full_like(sigma,
float("nan")), sigma)
    return sigma

def implied_vol_newton_robust(
    prices: torch.Tensor,
    S,
    K_grid,
    T_grid,
    r_grid,
    q=0.0,
    option_type: str = "otm",
    n_iter: int = 10,
) -> torch.Tensor:
    iv = implied_vol_newton(
        prices, S, K_grid, T_grid,
r_grid,
        q=q,
option_type=option_type,
n_iter=n_iter,
    )

    nan_mask = torch.isnan(iv)
    if bool(nan_mask.any().item()):
        guess_low =
torch.full_like(prices, 0.20)
```

```
        iv_alt = implied_vol_newton(
            prices, S, K_grid,
T_grid, r_grid,
            q=q,
option_type=option_type,
n_iter=n_iter,
            initial_guess=guess_low,
        )
        iv = torch.where(nan_mask,
iv_alt, iv)

    nan_mask = torch.isnan(iv)
    if bool(nan_mask.any().item()):
        guess_high =
torch.full_like(prices, 0.50)
        iv_alt = implied_vol_newton(
            prices, S, K_grid,
T_grid, r_grid,
            q=q,
option_type=option_type,
n_iter=n_iter,

initial_guess=guess_high,
        )
        iv = torch.where(nan_mask,
iv_alt, iv)

    return iv

@dataclass
class CalibrationResult:
    theta: np.ndarray
    loss: float
    n_iter: int
    converged: bool
    elapsed_seconds: float
    init_theta: np.ndarray
    diagnostics: dict

class Calibrator:
    def calibrate(self, surface:
dict, theta_init: np.ndarray = None)
-> CalibrationResult:
        raise NotImplementedError

@dataclass
class NaiveState:
    theta: np.ndarray | None = None

class NaiveCalibrator(Calibrator):

    def __init__(self) -> None:
        self.state = NaiveState()

    def calibrate(self, surface:
dict, theta_init: np.ndarray | None
= None) -> CalibrationResult:
        t0 = time.perf_counter()

        if self.state.theta is None:
            if theta_init is None:
                raise ValueError(
                    "NaiveCalibrator
has no prior theta; pass theta_init
on the first call."
                )
            theta =
np.asarray(theta_init,
dtype=float).copy()
        else:
            theta =
self.state.theta.copy()

        elapsed =
time.perf_counter() - t0
        self.state.theta =
theta.copy()

        return CalibrationResult(
            theta=theta,
            loss=float("nan"),
            n_iter=0,
            converged=True,
            elapsed_seconds=elapsed,
            init_theta=theta.copy(),
            diagnostics={"method":
"naive (yesterday)"},
        )

@dataclass
class LMConfig:
    max_iter: int
    ftol: float
    xtol: float
    gtol: float
    multistart: bool
    n_random_starts: int = 0
    seed: int = 42

RESEARCH_CONFIG = LMConfig(
    max_iter=200,
    ftol=1e-10, xtol=1e-10, gtol=1e-
10,
    multistart=True,
    n_random_starts=3,
    seed=42,
)

PRODUCTION_CONFIG = LMConfig(
    max_iter=50,
    ftol=1e-6, xtol=1e-6, gtol=1e-6,
    multistart=False,
    seed=42,
)

PARAM_BOUNDS = (
    np.array([0.01, 1e-4, 0.01, -
0.999, 1e-4]),
    np.array([20.0, 1.0,  5.0,
0.999,  1.0]),
)

_SAMPLE_BOUNDS = (
    np.array([0.5,  0.005, 0.1, -
0.95, 0.005]),
    np.array([10.0, 0.25,  2.0, -
0.2,   0.25]),
)

def _sample_random_start(rng:
np.random.RandomState) ->
np.ndarray:
    lo, hi = _SAMPLE_BOUNDS
    kappa =
float(np.exp(rng.uniform(np.log(lo[0
]), np.log(hi[0]))))
    mean_var =
float(np.exp(rng.uniform(np.log(lo[1
]), np.log(hi[1]))))
    sigma_v =
float(np.exp(rng.uniform(np.log(lo[2
]), np.log(hi[2]))))
    rho = float(rng.uniform(lo[3],
hi[3]))
    v0 =
float(np.exp(rng.uniform(np.log(lo[4
]), np.log(hi[4]))))
    return np.array([kappa,
mean_var, sigma_v, rho, v0])

class LMCalibrator(Calibrator):
    def __init__(self, config:
LMConfig):
        self.config = config

    def calibrate(self, surface:
dict, theta_init: np.ndarray | None
= None) -> CalibrationResult:
        start_time = time.time()

        iv_market_raw =
surface["iv_surface"]
        sigma_market =
iv_market_raw.T

        moneyness_grid =
np.asarray(surface["moneyness_grid"]
)
        ttm_grid =
np.asarray(surface["ttm_grid"])
        S = float(surface["spot"])
        rate_curve =
surface["rate_curve"]

        if isinstance(rate_curve,
dict):
            ttm_days = ttm_grid *
365.0
            r_grid =
np.array([rate_curve[float(d)] for d
in ttm_days])
        else:
            r_grid =
np.asarray(rate_curve)

        K_grid = moneyness_grid * S

        valid_mask =
~np.isnan(sigma_market)
        sigma_safe =
np.where(valid_mask, sigma_market,
1.0)

        K_b = K_grid.reshape(1, -1)
        T_b = ttm_grid.reshape(-1,
1)
        r_b = r_grid.reshape(-1, 1)

        d1 = (np.log(S / K_b) + (r_b
+ 0.5 * sigma_safe ** 2) * T_b) / (
            sigma_safe *
np.sqrt(T_b)
        )
        d2 = d1 - sigma_safe *
np.sqrt(T_b)
        vega = S * np.sqrt(T_b) *
st.norm.pdf(d1)
        vega[~valid_mask] = 0.0

        is_call = (K_b >=
S).astype(float)
        df_r = np.exp(-r_b * T_b)
        Phi = lambda x: 0.5 * (1.0 +
np.vectorize(__import__("math").erf)
(x / np.sqrt(2.0)))
        call_px = S * Phi(d1) - K_b
* df_r * Phi(d2)
        put_px = K_b * df_r * Phi(-
d2) - S * Phi(-d1)
        market_price = is_call *
call_px + (1.0 - is_call) * put_px
        market_price =
np.broadcast_to(market_price,
sigma_market.shape).copy()

        vega_sum = float(vega.sum())
        if vega_sum > 0:
            weights = vega /
vega_sum
        else:
            weights =
np.zeros_like(vega)
        sqrt_weights =
np.sqrt(weights)
        vega_floor = max(1e-6, 1e-3
* float(vega.max()) if vega.max() >
0 else 1e-6)

        S_t = torch.tensor(S,
dtype=torch.float64)
        K_t = torch.tensor(K_grid,
dtype=torch.float64)
        T_t = torch.tensor(ttm_grid,
dtype=torch.float64)
        r_t = torch.tensor(r_grid,
dtype=torch.float64)

        def residuals_np(theta_np:
np.ndarray) -> np.ndarray:
            theta_t =
torch.tensor(theta_np,
dtype=torch.float64)
            with torch.no_grad():
                model_prices =
heston_price_otm(*theta_t, S_t, K_t,
T_t, r_t)
                ivs =
implied_vol_newton(model_prices,
S_t, K_t, T_t, r_t,
option_type="otm")
                ivs_np =
ivs.cpu().numpy()
                model_prices_np =
model_prices.cpu().numpy()

            r =
np.zeros_like(sigma_market)
            model_nan =
np.isnan(ivs_np)
            bad = valid_mask &
model_nan
            good = valid_mask &
~model_nan

            r[good] = (ivs_np[good]
- sigma_market[good]) *
sqrt_weights[good]

            if bad.any():
                vega_clamped =
np.maximum(vega[bad], vega_floor)
                iv_diff =
(model_prices_np[bad] -
market_price[bad]) / vega_clamped
                iv_diff =
np.clip(iv_diff, -0.5, 0.5)
                r[bad] = iv_diff *
sqrt_weights[bad]
            return r.flatten()

        mean_iv =
float(np.nanmean(sigma_market)) if
np.any(valid_mask) else 0.2
        default_start = np.clip(
            np.array([2.0, mean_iv
** 2, 0.3, -0.7, mean_iv ** 2]),
            PARAM_BOUNDS[0],
PARAM_BOUNDS[1],
        )

        starts: list[np.ndarray] =
[]
        if theta_init is not None:

starts.append(np.clip(np.asarray(the
ta_init, dtype=float),

PARAM_BOUNDS[0], PARAM_BOUNDS[1]))

        if self.config.multistart:

starts.append(default_start)
            rng =
np.random.RandomState(self.config.se
ed)
            for _ in
range(self.config.n_random_starts):
```

```
starts.append(_sample_random_start(r
ng))
        elif not starts:

starts.append(default_start)

        best_res = None
        best_cost = np.inf
        best_init = None
        winning_idx = -1

        for i, x0 in
enumerate(starts):
            try:
                res =
opt.least_squares(
                    residuals_np,
                    x0=x0,

bounds=PARAM_BOUNDS,
                    method="trf",

max_nfev=self.config.max_iter,

ftol=self.config.ftol,

xtol=self.config.xtol,

gtol=self.config.gtol,
                )
            except Exception:
                continue
            x0_cost = 0.5 *
float(np.sum(residuals_np(x0) ** 2))
            if res.cost > x0_cost:
                res.x =
np.asarray(x0, dtype=float)
                res.cost = x0_cost
                res.fun =
residuals_np(x0)
                res.status = 4
                res.message = "TRF
result worse than start; pinned to
start point."
            if res.cost < best_cost:
                best_cost = res.cost
                best_res = res
                best_init = x0
                winning_idx = i

        elapsed = time.time() -
start_time

        if best_res is None:
            fallback = starts[0] if
starts else default_start
            return
CalibrationResult(
                theta=fallback,
                loss=float("inf"),
                n_iter=0,
                converged=False,

elapsed_seconds=elapsed,
                init_theta=fallback,

diagnostics={"error": "all solver
runs failed"},
            )

        final_loss =
float(np.sqrt(2.0 * best_res.cost))

        return CalibrationResult(
            theta=best_res.x,
            loss=final_loss,

n_iter=int(best_res.nfev),

converged=bool(best_res.status > 0),
            elapsed_seconds=elapsed,
            init_theta=best_init,
            diagnostics={

"winning_start_index": winning_idx,
                "n_starts":
len(starts),
                "status":
int(best_res.status),
                "message":
str(best_res.message),
            },
        )

PARAM_ORDER = ["kappa", "mean_var",
"sigma_v", "rho", "v0"]
_LOG_IDX = (0, 1, 2, 4)
_RHO_IDX = 3

_RHO_CLAMP = 1.0 - 1e-6
_POS_FLOOR = 1e-8
_Z_CLAMP = 10.0

@dataclass
class StandardizerConfig:
    mean: list[float]
    std: list[float]

    def to_dict(self) -> dict:
        return {"mean":
list(self.mean), "std":
list(self.std)}

    @classmethod
    def from_dict(cls, d: dict) ->
"StandardizerConfig":
        return
cls(mean=list(d["mean"]),
std=list(d["std"]))

def _to_transformed_torch(theta:
torch.Tensor) -> torch.Tensor:
    kappa = torch.clamp(theta[...,
0], min=_POS_FLOOR)
    mean_var =
torch.clamp(theta[..., 1],
min=_POS_FLOOR)
    sigma_v = torch.clamp(theta[...,
2], min=_POS_FLOOR)
    rho = torch.clamp(theta[..., 3],
min=-_RHO_CLAMP, max=_RHO_CLAMP)
    v0 = torch.clamp(theta[..., 4],
min=_POS_FLOOR)
    return torch.stack(
        [torch.log(kappa),
torch.log(mean_var),
torch.log(sigma_v),
         torch.atanh(rho),
torch.log(v0)],
        dim=-1,
    )

def _from_transformed_torch(z:
torch.Tensor) -> torch.Tensor:
    z_clamped = torch.clamp(z, min=-
_Z_CLAMP, max=_Z_CLAMP)
    kappa = torch.exp(z_clamped[...,
0])
    mean_var =
torch.exp(z_clamped[..., 1])
    sigma_v =
torch.exp(z_clamped[..., 2])
    rho = torch.tanh(z_clamped[...,
3])
    v0 = torch.exp(z_clamped[...,
4])
    return torch.stack([kappa,
mean_var, sigma_v, rho, v0], dim=-1)

class ParameterStandardizer:

    PARAM_ORDER = PARAM_ORDER

    def __init__(self, config:
StandardizerConfig):
        self.config = config
        self.mean =
torch.tensor(config.mean,
dtype=torch.float32)
        self.std =
torch.tensor(config.std,
dtype=torch.float32)

    @classmethod
    def fit(cls, theta_natural:
np.ndarray) ->
"ParameterStandardizer":
        if theta_natural.ndim != 2
or theta_natural.shape[1] != 5:
            raise ValueError(
                f"theta_natural must
be (N, 5); got
{theta_natural.shape}"
            )
        z =
np.empty_like(theta_natural,
dtype=np.float64)
        z[:, list(_LOG_IDX)] =
np.log(
            np.clip(theta_natural[:,
list(_LOG_IDX)], _POS_FLOOR, None)
        )
        rho =
np.clip(theta_natural[:, _RHO_IDX],
-_RHO_CLAMP, _RHO_CLAMP)
        z[:, _RHO_IDX] =
np.arctanh(rho)
        mean =
z.mean(axis=0).astype(float).tolist(
)
        std = z.std(axis=0,
ddof=0).astype(float).tolist()
        std = [s if s > 1e-12 else
1.0 for s in std]
        return
cls(StandardizerConfig(mean=mean,
std=std))

    def to_standardized(self,
theta_natural: torch.Tensor) ->
torch.Tensor:
        z_trans =
_to_transformed_torch(theta_natural)
        return (z_trans -
self.mean.to(z_trans.device,
z_trans.dtype)) / \

self.std.to(z_trans.device,
z_trans.dtype)

    def from_standardized(self, z:
torch.Tensor) -> torch.Tensor:
        z_trans = z *
self.std.to(z.device, z.dtype) + \
            self.mean.to(z.device,
z.dtype)
        return
_from_transformed_torch(z_trans)

    def to(self, device) ->
"ParameterStandardizer":
        self.mean =
self.mean.to(device)
        self.std =
self.std.to(device)
        return self

    def state_dict(self) -> dict:
        return {
            "mean":
self.mean.detach().cpu().tolist(),
            "std":
self.std.detach().cpu().tolist(),
        }

    @classmethod
    def from_state_dict(cls, sd:
dict) -> "ParameterStandardizer":
        return
cls(StandardizerConfig(mean=list(sd[
"mean"]), std=list(sd["std"])))

class HestonCNN(nn.Module):

    def __init__(
        self,
        channels: int = 32,
        mlp_hidden: int = 64,
        dropout: float = 0.1,
    ):
        super().__init__()
        c1 = channels
        c2 = 2 * channels
        self.conv = nn.Sequential(
            nn.Conv2d(1, c1,
kernel_size=3, padding=1),
            nn.BatchNorm2d(c1),
            nn.GELU(),
            nn.Conv2d(c1, c2,
kernel_size=3, padding=1),
            nn.BatchNorm2d(c2),
            nn.GELU(),
            nn.Conv2d(c2, c2,
kernel_size=3, padding=1),
            nn.BatchNorm2d(c2),
            nn.GELU(),
        )
        flat_features = c2 * 11 * 8
        self.head = nn.Sequential(
            nn.Flatten(),
            nn.Linear(flat_features,
mlp_hidden),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(mlp_hidden,
5),
        )

    def forward(self, x:
torch.Tensor) -> torch.Tensor:
        if x.ndim != 4 or x.shape[1]
!= 1 or x.shape[2:] != (11, 8):
            raise ValueError(
                f"HestonCNN expects
input shape (B, 1, 11, 8); got
{tuple(x.shape)}"
            )
        return
self.head(self.conv(x))

    def num_parameters(self) -> int:
        return sum(p.numel() for p
in self.parameters())

PARAM_NAMES = ["kappa", "mean_var",
"sigma_v", "rho", "v0"]
_LOG_IDX = [0, 1, 2, 4]
_RHO_IDX = 3

def _to_transformed(theta:
np.ndarray) -> np.ndarray:
    z = np.empty_like(theta,
dtype=float)
    z[:, _LOG_IDX] = np.log(theta[:,
_LOG_IDX])
    z[:, _RHO_IDX] =
np.arctanh(theta[:, _RHO_IDX])
```

```
    return z

def _from_transformed(z: np.ndarray)
-> np.ndarray:
    theta = np.empty_like(z,
dtype=float)
    theta[:, _LOG_IDX] = np.exp(z[:,
_LOG_IDX])
    theta[:, _RHO_IDX] =
np.tanh(z[:, _RHO_IDX])
    return theta

@dataclass
class PriorConfig:
    loss_threshold: float = 0.01
    bound_margin: float = 0.01
    min_samples: int = 100
    ridge: float = 1e-6
    clip_inset: float = 1e-3

    def to_dict(self) -> dict:
        return {
            "loss_threshold":
self.loss_threshold,
            "bound_margin":
self.bound_margin,
            "min_samples":
self.min_samples,
            "ridge": self.ridge,
            "clip_inset":
self.clip_inset,
        }

    @classmethod
    def from_dict(cls, d: dict) ->
"PriorConfig":
        return cls(**d)

def _filter_lm_history(df:
pd.DataFrame, config: PriorConfig) -
> pd.DataFrame:
    if "converged" in df.columns:
        df =
df[df["converged"].astype(bool)]
    df = df[df["loss"] <
config.loss_threshold]

    lo, hi = PARAM_BOUNDS
    margin = config.bound_margin
    keep = np.ones(len(df),
dtype=bool)
    for i, name in
enumerate(PARAM_NAMES):
        vals =
df[name].to_numpy(dtype=float)
        span = hi[i] - lo[i]
        slack = margin * span
        keep &= (vals > lo[i] +
slack) & (vals < hi[i] - slack)
    return
df.iloc[keep].reset_index(drop=True)

@dataclass
class EmpiricalPrior:

    mu: np.ndarray
    sigma: np.ndarray
    config: PriorConfig =
field(default_factory=PriorConfig)
    n_used: int = 0

    def __post_init__(self):
        self.mu =
np.asarray(self.mu,
dtype=float).reshape(5)
        self.sigma =
np.asarray(self.sigma,
dtype=float).reshape(5, 5)
        self.sigma = 0.5 *
(self.sigma + self.sigma.T)
        self._chol =
np.linalg.cholesky(self.sigma)

    @classmethod
    def fit(
        cls,
        lm_history: pd.DataFrame,
        config: PriorConfig | None =
None,
    ) -> "EmpiricalPrior":
        config = config or
PriorConfig()
        filtered =
_filter_lm_history(lm_history,
config)
        if len(filtered) <
config.min_samples:
            raise ValueError(
                f"Only
{len(filtered)} rows survived
filtering "
                f"(need >=
{config.min_samples}). Loosen
PriorConfig or "
                f"check lm_history
quality."
            )
        theta =
filtered[PARAM_NAMES].to_numpy(dtype
=float)
        z = _to_transformed(theta)
        mu = z.mean(axis=0)
        sigma = np.cov(z,
rowvar=False, ddof=1) + config.ridge
* np.eye(5)
        return cls(mu=mu,
sigma=sigma, config=config,
n_used=len(filtered))

    def sample(
        self, n: int, rng:
np.random.Generator
    ) -> np.ndarray:
        z_std =
rng.standard_normal(size=(n, 5))
        z = self.mu[None, :] + z_std
@ self._chol.T
        theta = _from_transformed(z)

        lo, hi = PARAM_BOUNDS
        inset =
self.config.clip_inset
        lo_inset = lo + inset * (hi
- lo)
        hi_inset = hi - inset * (hi
- lo)
        clipped = np.clip(theta,
lo_inset[None, :], hi_inset[None,
:])
        return clipped

    def sample_with_clip_rate(
        self, n: int, rng:
np.random.Generator
    ) -> tuple[np.ndarray, float]:
        z_std =
rng.standard_normal(size=(n, 5))
        z = self.mu[None, :] + z_std
@ self._chol.T
        theta = _from_transformed(z)

        lo, hi = PARAM_BOUNDS
        inset =
self.config.clip_inset
        lo_inset = lo + inset * (hi
- lo)
        hi_inset = hi - inset * (hi
- lo)
        out_mask = (theta <
lo_inset[None, :]) | (theta >
hi_inset[None, :])
        clip_rate =
float(out_mask.any(axis=1).mean())
        clipped = np.clip(theta,
lo_inset[None, :], hi_inset[None,
:])
        return clipped, clip_rate

    def to_dict(self) -> dict:
        return {
            "mu": self.mu.tolist(),
            "sigma":
self.sigma.tolist(),
            "param_names":
PARAM_NAMES,
            "n_used":
int(self.n_used),
            "config":
self.config.to_dict(),
        }

    @classmethod
    def from_dict(cls, d: dict) ->
"EmpiricalPrior":
        return cls(
            mu=np.asarray(d["mu"],
dtype=float),

sigma=np.asarray(d["sigma"],
dtype=float),

config=PriorConfig.from_dict(d.get("
config", {})),

n_used=int(d.get("n_used", 0)),
        )

    def save(self, path: str | Path)
-> None:
        path = Path(path)

path.parent.mkdir(parents=True,
exist_ok=True)
        with open(path, "w") as f:

json.dump(self.to_dict(), f,
indent=2)

    @classmethod
    def load(cls, path: str | Path)
-> "EmpiricalPrior":
        with open(path) as f:
            return
cls.from_dict(json.load(f))

@dataclass
class GeneratorConfig:
    n_samples: int = 500_000
    batch_size: int = 1024
    nan_threshold: float = 0.05
    seed: int = 42
    cos_N: int = 128
    cos_L: float = 10.0

def _nearest_fill_2d(surface:
np.ndarray) -> np.ndarray:
    if not np.isnan(surface).any():
        return surface
    out = surface.copy()
    n_k, n_t = surface.shape
    finite = np.isfinite(surface)
    if not finite.any():
        return out
    fi = np.argwhere(finite)
    nan_idx = np.argwhere(~finite)
    for (i, j) in nan_idx:
        d = np.maximum(np.abs(fi[:,
0] - i), np.abs(fi[:, 1] - j))
        k = int(np.argmin(d))
        out[i, j] = surface[fi[k,
0], fi[k, 1]]
    return out

@dataclass
class SyntheticGenerator:
    prior: EmpiricalPrior
    cleaned_metadata: pd.DataFrame
    rate_curve: object
    grid: dict
    config: GeneratorConfig =
field(default_factory=GeneratorConfi
g)

    def __post_init__(self):
        self.moneyness_grid =
np.asarray(self.grid["moneyness_grid
"], dtype=float)
        self.ttm_grid_days =
np.asarray(self.grid["ttm_grid_days"
], dtype=float)
        self.ttm_grid_years =
self.ttm_grid_days / 365.0
        self._meta_dates =
pd.to_datetime(self.cleaned_metadata
["date"]).dt.normalize().to_numpy()
        self._meta_spots =
self.cleaned_metadata["spot"].to_num
py(dtype=float)
        self._cached_rate_curves =
np.stack(

[self.rate_curve.get_rates(d,
self.ttm_grid_days) for d in
self._meta_dates],
            axis=0,
        )

    def sample_market_context(
        self, n: int, rng:
np.random.Generator
    ) -> tuple[np.ndarray,
np.ndarray]:
        idx = rng.integers(0,
len(self._meta_spots), size=n)
        spots =
self._meta_spots[idx]
        r_curves =
self._cached_rate_curves[idx]
        return spots, r_curves

    def generate_batch(
        self, batch_size: int, rng:
np.random.Generator
    ) -> tuple[dict, dict]:
        theta_np, clip_rate =
self.prior.sample_with_clip_rate(bat
ch_size, rng)

        spots_full, r_curves_full =
self.sample_market_context(1, rng)
        S = float(spots_full[0])
        r_curve = r_curves_full[0]

        K_grid = self.moneyness_grid
* S

        theta_t =
torch.tensor(theta_np,
dtype=torch.float64)
        S_t = torch.tensor(S,
dtype=torch.float64)
        K_t = torch.tensor(K_grid,
dtype=torch.float64)
        T_t =
torch.tensor(self.ttm_grid_years,
dtype=torch.float64)
```

```
        r_t = torch.tensor(r_curve,
dtype=torch.float64)

        with torch.no_grad():
            prices =
heston_price_otm(
                theta_t[:, 0],
theta_t[:, 1], theta_t[:, 2],
                theta_t[:, 3],
theta_t[:, 4],
                S_t, K_t, T_t, r_t,
                N=self.config.cos_N,
L=self.config.cos_L,
            )
            ivs =
implied_vol_newton(
                prices, S_t, K_t,
T_t, r_t, option_type="otm",
n_iter=30,
            )
            init_high =
torch.full_like(prices, 0.5)
            ivs_h =
implied_vol_newton(
                prices, S_t, K_t,
T_t, r_t, option_type="otm",
                n_iter=30,
initial_guess=init_high,
            )
            ivs =
torch.where(torch.isnan(ivs), ivs_h,
ivs)
            init_low =
torch.full_like(prices, 0.15)
            ivs_l =
implied_vol_newton(
                prices, S_t, K_t,
T_t, r_t, option_type="otm",
                n_iter=30,
initial_guess=init_low,
            )
            ivs =
torch.where(torch.isnan(ivs), ivs_l,
ivs)
        ivs_np = ivs.cpu().numpy()
        ivs_np = np.where(
            np.isfinite(ivs_np) &
(ivs_np > 1e-4) & (ivs_np < 5.0),
            ivs_np, np.nan,
        )

        n_cells = ivs_np.shape[1] *
ivs_np.shape[2]
        nan_frac =
np.isnan(ivs_np).reshape(batch_size,
-1).mean(axis=1)
        keep = nan_frac <=
self.config.nan_threshold
        n_dropped =
int((~keep).sum())

        ivs_kept =
np.transpose(ivs_np[keep], axes=(0,
2, 1))
        ivs_filled = np.stack(
            [_nearest_fill_2d(s) for
s in ivs_kept], axis=0,
        )

        theta_kept = theta_np[keep]
        n_kept = int(keep.sum())

        batch = {
            "theta":
theta_kept,
            "iv_surface":
ivs_filled,
            "spot":
np.full(n_kept, S, dtype=float),
            "r_curve":
np.tile(r_curve, (n_kept, 1)),
        }
        stats = {
            "n_drawn": batch_size,
            "n_kept": n_kept,
            "n_dropped_nan":
n_dropped,
            "clip_rate": clip_rate,
        }
        return batch, stats

    def generate_full_dataset(
        self, output_path: str |
Path, prior_path: str | Path | None
= None,
    ) -> dict:
        output_path =
Path(output_path)

output_path.parent.mkdir(parents=Tru
e, exist_ok=True)

        rng =
np.random.default_rng(self.config.se
ed)

        n_target =
self.config.n_samples
        B = self.config.batch_size
        n_batches = (n_target + B -
1) // B
        n_K =
len(self.moneyness_grid)
        n_T =
len(self.ttm_grid_years)

        total_drawn = 0
        total_dropped = 0
        clip_rate_sum = 0.0
        n_written = 0
        t0 = time.time()

        with h5py.File(output_path,
"w") as f:
            chunk = min(4096,
n_target)
            theta_ds =
f.create_dataset(
                "theta", shape=(0,
5), maxshape=(None, 5),
                dtype="float64",
chunks=(chunk, 5),
                compression="gzip",
compression_opts=4,
            )
            iv_ds =
f.create_dataset(
                "iv_surface",
shape=(0, n_K, n_T), maxshape=(None,
n_K, n_T),
                dtype="float64",
chunks=(chunk, n_K, n_T),
                compression="gzip",
compression_opts=4,
            )
            spot_ds =
f.create_dataset(
                "spot", shape=(0,),
maxshape=(None,),
                dtype="float64",
chunks=(chunk,),
                compression="gzip",
compression_opts=4,
            )
            r_ds = f.create_dataset(
                "r_curve", shape=(0,
n_T), maxshape=(None, n_T),
                dtype="float64",
chunks=(chunk, n_T),
                compression="gzip",
compression_opts=4,
            )

            for i in
range(n_batches):
                remaining = n_target
- n_written
                this_b = min(B,
max(remaining, 1))
                batch, stats =
self.generate_batch(this_b, rng)
                total_drawn +=
stats["n_drawn"]
                total_dropped +=
stats["n_dropped_nan"]
                clip_rate_sum +=
stats["clip_rate"]

                n_kept =
batch["theta"].shape[0]
                if n_written +
n_kept > n_target:
                    n_kept =
n_target - n_written
                    for k in batch:
                        batch[k] =
batch[k][:n_kept]

                if n_kept == 0:
                    continue

                new_size = n_written
+ n_kept

theta_ds.resize((new_size, 5))

iv_ds.resize((new_size, n_K, n_T))

spot_ds.resize((new_size,))

r_ds.resize((new_size, n_T))


theta_ds[n_written:new_size] =
batch["theta"]

iv_ds[n_written:new_size] =
batch["iv_surface"]

spot_ds[n_written:new_size] =
batch["spot"]

r_ds[n_written:new_size] =
batch["r_curve"]
                n_written = new_size

                if (i + 1) % 50 == 0
or (i + 1) == n_batches:
                    elapsed =
time.time() - t0
                    drop_rate =
total_dropped / max(total_drawn, 1)
                    print(

f"[{i+1}/{n_batches}]
written={n_written}/{n_target} "

f"elapsed={elapsed:.1f}s
drop_rate={drop_rate*100:.2f}%"
                    )

                if n_written >=
n_target:
                    break


f.attrs["moneyness_grid"] =
self.moneyness_grid
            f.attrs["ttm_grid_days"]
= self.ttm_grid_days

f.attrs["ttm_grid_years"] =
self.ttm_grid_years
            f.attrs["seed"] =
self.config.seed
            f.attrs["prior_path"] =
str(prior_path) if prior_path else
""

f.attrs["generation_date"] =
pd.Timestamp.utcnow().isoformat()
            f.attrs["param_names"] =
np.array(PARAM_NAMES, dtype="S")

        elapsed = time.time() - t0
        return {
            "n_written": n_written,
            "n_drawn": total_drawn,
            "n_dropped_nan":
total_dropped,
            "drop_rate":
total_dropped / max(total_drawn, 1),
            "mean_clip_rate":
clip_rate_sum / max(n_batches, 1),
            "elapsed_seconds":
elapsed,
            "output_path":
str(output_path),
        }

class SyntheticDataset(Dataset):

    def __init__(
        self,
        h5_path: str,
        indices: np.ndarray,
        standardizer:
ParameterStandardizer,
        include_context: bool =
False,
    ):
        self.h5_path = str(h5_path)
        idx_sorted =
np.sort(np.asarray(indices,
dtype=np.int64))
        self.indices = idx_sorted
        self.standardizer =
standardizer
        self.include_context =
bool(include_context)

        with h5py.File(self.h5_path,
"r") as f:
            iv =
f["iv_surface"][idx_sorted, :, :]
            theta =
f["theta"][idx_sorted, :]
            if self.include_context:
                spot =
f["spot"][idx_sorted]
                r_curve =
f["r_curve"][idx_sorted, :]

        self.iv_tensor =
torch.from_numpy(
            iv.astype(np.float32,
copy=False)
        ).unsqueeze(1)
        self.theta_tensor =
torch.from_numpy(
            theta.astype(np.float32,
copy=False)
        )
        with torch.no_grad():
            self.z_tensor = (
```

```
standardizer.to_standardized(self.th
eta_tensor.cpu()).contiguous()
            )

        if self.include_context:
            self.spot_tensor =
torch.from_numpy(spot.astype(np.floa
t32, copy=False))
            self.r_curve_tensor =
torch.from_numpy(r_curve.astype(np.f
loat32, copy=False))
        else:
            self.spot_tensor = None
            self.r_curve_tensor =
None

    def __len__(self) -> int:
        return
self.iv_tensor.shape[0]

    def __getitem__(self, i: int) ->
dict:
        batch = {
            "iv_surface":
self.iv_tensor[i],
            "z": self.z_tensor[i],
        }
        if self.include_context:
            batch["theta"] =
self.theta_tensor[i]
            batch["spot"] =
self.spot_tensor[i]
            batch["r_curve"] =
self.r_curve_tensor[i]
        return batch

def SyntheticDatasetWithContext(
    h5_path: str, indices:
np.ndarray, standardizer:
ParameterStandardizer,
) -> SyntheticDataset:
    return SyntheticDataset(h5_path,
indices, standardizer,
include_context=True)

@dataclass
class GridInfo:
    moneyness_grid: np.ndarray
    ttm_grid_years: np.ndarray

def read_grid_info(h5_path: str) ->
GridInfo:
    with h5py.File(h5_path, "r") as
f:
        return GridInfo(

moneyness_grid=np.asarray(f.attrs["m
oneyness_grid"], dtype=np.float64),

ttm_grid_years=np.asarray(f.attrs["t
tm_grid_years"], dtype=np.float64),
        )

def make_split_indices(
    n_total: int,
    seed: int,
    val_frac: float = 0.05,
    test_frac: float = 0.05,
) -> tuple[np.ndarray, np.ndarray,
np.ndarray]:
    rng =
np.random.default_rng(seed)
    perm = rng.permutation(n_total)
    n_val = int(round(val_frac *
n_total))
    n_test = int(round(test_frac *
n_total))
    val = perm[:n_val]
    test = perm[n_val:n_val +
n_test]
    train = perm[n_val + n_test:]
    return np.sort(train),
np.sort(val), np.sort(test)

def make_subset_split_indices(
    h5_path: str,
    seed: int = 42,
    n_samples: int | None = None,
) -> tuple[np.ndarray, np.ndarray,
np.ndarray]:
    with h5py.File(h5_path, "r") as
f:
        total =
int(f["theta"].shape[0])
    rng =
np.random.default_rng(seed)
    all_indices = np.arange(total)
    rng.shuffle(all_indices)
    if n_samples is not None:
        if n_samples > total:
            raise ValueError(
                f"Requested
{n_samples} samples but only {total}
available "
                f"in {h5_path}."
            )
        all_indices =
all_indices[:n_samples]
    n = len(all_indices)
    n_train = int(0.90 * n)
    n_val = int(0.05 * n)
    train_idx =
all_indices[:n_train]
    val_idx =
all_indices[n_train:n_train + n_val]
    test_idx = all_indices[n_train +
n_val:]
    return train_idx, val_idx,
test_idx

def fit_standardizer_on_train(
    h5_path: str, train_indices:
np.ndarray
) -> ParameterStandardizer:
    with h5py.File(h5_path, "r") as
f:
        theta_all = f["theta"][...]
    theta_train =
theta_all[train_indices]
    return
ParameterStandardizer.fit(theta_trai
n)

def
_macos_dataloader_kwargs(num_workers
: int) -> dict:
    if num_workers > 0:
        return {"num_workers":
num_workers,
"multiprocessing_context": "fork",

"persistent_workers": True}
    return {"num_workers": 0}

def make_dataloaders(
    h5_path: str,
    batch_size: int,
    num_workers: int = 0,
    seed: int = 42,
    standardizer:
ParameterStandardizer | None = None,
    include_context: bool = False,
    n_samples: int | None = None,
) -> tuple[DataLoader, DataLoader,
DataLoader, ParameterStandardizer]:
    train_idx, val_idx, test_idx =
make_subset_split_indices(
        h5_path, seed=seed,
n_samples=n_samples,
    )
    if standardizer is None:
        standardizer =
fit_standardizer_on_train(h5_path,
train_idx)

    train_ds =
SyntheticDataset(h5_path, train_idx,
standardizer, include_context)
    val_ds =
SyntheticDataset(h5_path, val_idx,
standardizer, include_context)
    test_ds =
SyntheticDataset(h5_path, test_idx,
standardizer, include_context)

    dl_kwargs =
_macos_dataloader_kwargs(num_workers
)
    train_loader = DataLoader(
        train_ds,
batch_size=batch_size, shuffle=True,
drop_last=True,
        pin_memory=False,
**dl_kwargs,
    )
    val_loader = DataLoader(
        val_ds,
batch_size=batch_size,
shuffle=False, drop_last=False,
        pin_memory=False,
**dl_kwargs,
    )
    test_loader = DataLoader(
        test_ds,
batch_size=batch_size,
shuffle=False, drop_last=False,
        pin_memory=False,
**dl_kwargs,
    )
    return train_loader, val_loader,
test_loader, standardizer

def standardized_mse(z_pred:
torch.Tensor, z_true: torch.Tensor)
-> torch.Tensor:
    return torch.mean((z_pred -
z_true) ** 2)

def per_parameter_metrics(
    theta_pred: torch.Tensor,
    theta_true: torch.Tensor,
) -> dict[str, float]:
    pred =
theta_pred.detach().cpu().numpy()
    true =
theta_true.detach().cpu().numpy()
    out: dict[str, float] = {}
    for i, name in
enumerate(PARAM_ORDER):
        diff = pred[:, i] - true[:,
i]
        out[f"{name}_mae"] =
float(np.mean(np.abs(diff)))
        if name == "rho":
            continue
        denom =
np.where(np.abs(true[:, i]) > 1e-12,
np.abs(true[:, i]), np.nan)
        mape =
np.nanmean(np.abs(diff) / denom)
        out[f"{name}_mape"] =
float(mape)
    return out

def _hash_context(spot: float,
r_curve: np.ndarray) -> bytes:
    return spot.tobytes() +
r_curve.tobytes()

def repricing_rmse_iv(
    theta_pred: torch.Tensor,
    iv_surface_true: torch.Tensor,
    spot: torch.Tensor,
    r_curve: torch.Tensor,
    moneyness_grid: torch.Tensor,
    ttm_grid: torch.Tensor,
    cos_N: int = 128,
    cos_L: float = 10.0,
) -> torch.Tensor:
    if theta_pred.shape[0] !=
iv_surface_true.shape[0]:
        raise ValueError("theta_pred
and iv_surface_true batch sizes
differ")

    out_device = theta_pred.device
    out_dtype = theta_pred.dtype

    K_dtype = torch.float64
    cpu = torch.device("cpu")

    def _to_cpu64(t: torch.Tensor) -
> torch.Tensor:
        return
t.to(device=cpu).to(dtype=K_dtype)

    theta_cpu =
_to_cpu64(theta_pred)
    iv_true_cpu =
_to_cpu64(iv_surface_true)
    moneyness =
_to_cpu64(moneyness_grid)
    T_grid = _to_cpu64(ttm_grid)

    spot_cpu =
_to_cpu64(spot.detach()).numpy()
    r_cpu =
_to_cpu64(r_curve.detach()).numpy()
    groups: dict[bytes, list[int]] =
{}
    for i in
range(spot_cpu.shape[0]):
        key =
_hash_context(spot_cpu[i], r_cpu[i])
        groups.setdefault(key,
[]).append(i)

    sq_terms: list[torch.Tensor] =
[]
    n_finite_terms:
list[torch.Tensor] = []

    for idxs in groups.values():
        idx_t = torch.tensor(idxs,
device=cpu, dtype=torch.long)
        theta_g =
theta_cpu.index_select(0, idx_t)
        iv_true_g =
iv_true_cpu.index_select(0, idx_t)
        S_g =
float(spot_cpu[idxs[0]])
        r_g =
torch.from_numpy(r_cpu[idxs[0]]).to(
dtype=K_dtype)

        S_t = torch.tensor(S_g,
dtype=K_dtype)
        K_grid = moneyness * S_t

        prices = heston_price_otm(
            theta_g[:, 0],
theta_g[:, 1], theta_g[:, 2],
            theta_g[:, 3],
theta_g[:, 4],
            S_t, K_grid, T_grid,
r_g, N=cos_N, L=cos_L,
        )
```

```
        ivs =
implied_vol_newton_robust(
            prices, S_t, K_grid,
T_grid, r_g, option_type="otm",
n_iter=10,
        )

        iv_true_g_pricer =
iv_true_g.transpose(-2, -1)

        valid = torch.isfinite(ivs)
& torch.isfinite(iv_true_g_pricer)
        diff = torch.where(valid,
ivs - iv_true_g_pricer,

torch.zeros_like(ivs))
        sq_terms.append((diff *
diff).sum())

n_finite_terms.append(valid.sum().to
(K_dtype))

    total_sq =
torch.stack(sq_terms).sum()
    total_n =
torch.stack(n_finite_terms).sum()
    rmse = torch.sqrt(total_sq /
torch.clamp(total_n, min=1.0))
    return
rmse.to(dtype=out_dtype).to(device=o
ut_device)

@dataclass
class TrainingConfig:
    epochs: int = 30
    batch_size: int = 256
    lr: float = 1e-3
    weight_decay: float = 1e-4
    grad_clip: float = 1.0
    num_workers: int = 2
    early_stop_patience: int = 5
    log_dir: str = "models/"
    checkpoint_every: int = 5
    seed: int = 42
    method_tag: str = "method2"

    def to_dict(self) -> dict:
        return asdict(self)

LossFn = Callable[
    [torch.Tensor, torch.Tensor,
dict],
    torch.Tensor,
]
MonitoringFn = Callable[
    [nn.Module,
torch.utils.data.DataLoader,
ParameterStandardizer,
torch.device],
    dict[str, float],
]

def _set_seeds(seed: int) -> None:
    torch.manual_seed(seed)
    np.random.seed(seed)

def _save_checkpoint(
    path: Path,
    model: nn.Module,
    optimizer:
torch.optim.Optimizer,
    standardizer:
ParameterStandardizer,
    epoch: int,
    best_val_loss: float,
    config: TrainingConfig,
) -> None:
    torch.save(
        {
            "model_state_dict":
model.state_dict(),
            "optimizer_state_dict":
optimizer.state_dict(),

"standardizer_state_dict":
standardizer.state_dict(),
            "epoch": epoch,
            "best_val_loss":
best_val_loss,
            "config":
config.to_dict(),
        },
        path,
    )

def _move_batch(batch: dict, device:
torch.device) -> dict:
    return {
        k: (v.to(device) if
isinstance(v, torch.Tensor) else v)
        for k, v in batch.items()
    }

def _eval_loss(
    model: nn.Module,
    loader:
torch.utils.data.DataLoader,
    loss_fn: LossFn,
    device: torch.device,
) -> float:
    model.eval()
    total = 0.0
    n = 0
    with torch.no_grad():
        for batch in loader:
            batch =
_move_batch(batch, device)
            z_pred =
model(batch["iv_surface"])
            loss = loss_fn(z_pred,
batch["z"], batch)
            bs = batch["z"].shape[0]
            total +=
float(loss.item()) * bs
            n += bs
    return total / max(n, 1)

_EARLY_STOP_METRIC_KEY = {
    "val_loss": "val_loss",
    "val_repricing_rmse":
"repricing_rmse",
}

def train(
    model: nn.Module,
    loss_fn: LossFn,
    train_loader:
torch.utils.data.DataLoader,
    val_loader:
torch.utils.data.DataLoader,
    standardizer:
ParameterStandardizer,
    config: TrainingConfig,
    device: torch.device,
    monitoring_fn:
Optional[MonitoringFn] = None,
    early_stop_metric: str =
"val_loss",
) -> dict:
    if early_stop_metric not in
_EARLY_STOP_METRIC_KEY:
        raise ValueError(
            f"Unknown
early_stop_metric={early_stop_metric
!r}; "
            f"must be one of
{list(_EARLY_STOP_METRIC_KEY)}."
        )
    metric_row_key =
_EARLY_STOP_METRIC_KEY[early_stop_me
tric]
    if early_stop_metric ==
"val_repricing_rmse" and
monitoring_fn is None:
        raise ValueError(

"early_stop_metric='val_repricing_rm
se' requires a monitoring_fn "
            "that returns
'repricing_rmse'."
        )

    _set_seeds(config.seed)
    log_dir = Path(config.log_dir)
    log_dir.mkdir(parents=True,
exist_ok=True)
    tag = config.method_tag
    best_path = log_dir /
f"{tag}_best.pt"
    final_path = log_dir /
f"{tag}_final.pt"
    csv_path = log_dir /
f"{tag}_training_log.csv"

    model = model.to(device)
    standardizer.to(device)

    optimizer = torch.optim.AdamW(
        model.parameters(),
lr=config.lr,
weight_decay=config.weight_decay,
    )
    scheduler =
torch.optim.lr_scheduler.CosineAnnea
lingLR(
        optimizer,
T_max=max(config.epochs, 1),
    )

    best_val_loss = float("inf")
    best_metric_value = float("inf")
    best_epoch = -1
    epochs_since_best = 0
    t_start = time.time()

    csv_columns: list[str] | None =
None
    csv_rows: list[dict] = []

    for epoch in range(1,
config.epochs + 1):
        model.train()
        epoch_t0 = time.time()
        train_total = 0.0
        train_n = 0
        for batch in train_loader:
            batch =
_move_batch(batch, device)
            z_pred =
model(batch["iv_surface"])
            loss = loss_fn(z_pred,
batch["z"], batch)

optimizer.zero_grad(set_to_none=True
)
            loss.backward()

torch.nn.utils.clip_grad_norm_(model
.parameters(), config.grad_clip)
            optimizer.step()
            bs = batch["z"].shape[0]
            train_total +=
float(loss.item()) * bs
            train_n += bs

        train_loss = train_total /
max(train_n, 1)
        val_loss = _eval_loss(model,
val_loader, loss_fn, device)
        scheduler.step()

        wall = time.time() -
epoch_t0
        lr =
optimizer.param_groups[0]["lr"]
        row: dict = {
            "epoch": epoch,
            "train_loss":
train_loss,
            "val_loss": val_loss,
            "lr": lr,
            "wall_time": wall,
        }
        if monitoring_fn is not
None:
            extra =
monitoring_fn(model, val_loader,
standardizer, device)
            row.update(extra)

        if metric_row_key not in
row:
            raise KeyError(

f"early_stop_metric={early_stop_metr
ic!r} expects "

f"row[{metric_row_key!r}] but only
got: {list(row.keys())}"
            )
        current_metric =
float(row[metric_row_key])
        improved = current_metric <
best_metric_value - 1e-9

        row["best_metric_name"] =
early_stop_metric
        row["best_metric_value"] = (
            current_metric if
improved else best_metric_value
        )

        if csv_columns is None:
            csv_columns =
list(row.keys())
        csv_rows.append(row)

        with open(csv_path, "w",
newline="") as f:
            writer =
csv.DictWriter(f,
fieldnames=csv_columns)
            writer.writeheader()
            for r in csv_rows:
                writer.writerow(r)

        if improved:
            best_metric_value =
current_metric
            best_val_loss = val_loss
            best_epoch = epoch
            epochs_since_best = 0
            _save_checkpoint(
                best_path, model,
optimizer, standardizer,
                epoch,
best_metric_value, config,
            )
        else:
            epochs_since_best += 1

        msg = (
            f"[{tag} epoch
{epoch:>3}/{config.epochs}] "
            f"train={train_loss:.4e}
val={val_loss:.4e} "
```

```
            f"lr={lr:.2e}
wall={wall:.1f}s"
        )
        if monitoring_fn is not None
and "repricing_rmse" in row:
            msg += f"
reprice_iv_rmse={row['repricing_rmse
']:.4e}"
        if improved:
            msg += " *"
        print(msg, flush=True)

        if epochs_since_best >=
config.early_stop_patience:
            print(
                f"[{tag}] early stop
after {epoch} epochs "
                f"(no improvement in
{early_stop_metric} for "

f"{config.early_stop_patience}).",
                flush=True,
            )
            break

    _save_checkpoint(
        final_path, model,
optimizer, standardizer,
        epoch, best_metric_value,
config,
    )

    total_wall = time.time() -
t_start
    return {
        "best_val_loss":
best_val_loss,
        "best_metric_name":
early_stop_metric,
        "best_metric_value":
best_metric_value,
        "best_epoch": best_epoch,
        "final_epoch": epoch,
        "total_wall_seconds":
total_wall,
        "best_checkpoint":
str(best_path),
        "final_checkpoint":
str(final_path),
        "log_csv": str(csv_path),
    }

def make_method2_loss():
    def loss_fn(z_pred, z_true,
batch):
        return
standardized_mse(z_pred, z_true)
    return loss_fn

def _select_monitoring_indices(
    val_indices_path: str | Path,
    n_monitor: int,
    seed: int,
) -> np.ndarray:
    val_indices =
np.load(val_indices_path)
    rng =
np.random.default_rng(seed)
    if len(val_indices) <=
n_monitor:
        return val_indices.copy()
    return rng.choice(val_indices,
size=n_monitor, replace=False)

class MonitoringBatch:

    def __init__(
        self,
        h5_path: str,
        indices: np.ndarray,
        grid_info: GridInfo,
        device: torch.device,
    ):
        idx =
np.sort(np.asarray(indices))
        with h5py.File(h5_path, "r")
as f:
            iv =
f["iv_surface"][idx, :, :]
            theta = f["theta"][idx,
:]
            spot = f["spot"][idx]
            r_curve =
f["r_curve"][idx, :]

        self.iv_surface =
torch.from_numpy(iv).float().unsquee
ze(1).to(device)
        self.theta_true =
torch.from_numpy(theta).float().to(d
evice)
        self.spot =
torch.from_numpy(spot).double()
        self.r_curve =
torch.from_numpy(r_curve).double()
        self.iv_true_for_rmse =
torch.from_numpy(iv).double()
        self.moneyness_grid =
torch.from_numpy(grid_info.moneyness
_grid).double()
        self.ttm_grid =
torch.from_numpy(grid_info.ttm_grid_
years).double()

def make_monitoring_fn(
    monitoring_batch:
MonitoringBatch,
    standardizer:
ParameterStandardizer,
):
    def monitoring_fn(model,
val_loader, standardizer_unused,
device):
        del val_loader,
standardizer_unused
        model.eval()
        with torch.no_grad():
            z_pred =
model(monitoring_batch.iv_surface)
            theta_pred =
standardizer.from_standardized(z_pre
d)
            metrics =
per_parameter_metrics(theta_pred,
monitoring_batch.theta_true)
            try:
                rmse =
repricing_rmse_iv(
                    theta_pred,

monitoring_batch.iv_true_for_rmse,

monitoring_batch.spot,

monitoring_batch.r_curve,

monitoring_batch.moneyness_grid,

monitoring_batch.ttm_grid,
                )

metrics["repricing_rmse"] =
float(rmse.item())
            except Exception as e:

metrics["repricing_rmse"] =
float("nan")

metrics["repricing_error"] =
str(e)[:80]
        return metrics
    return monitoring_fn

def make_method3_loss(
    standardizer:
ParameterStandardizer,
    moneyness_grid: torch.Tensor,
    ttm_grid: torch.Tensor,
    lambda_repricing: float,
    pricing_device: torch.device |
str = "cpu",
) -> Callable:
    pricing_device =
torch.device(pricing_device)

    moneyness_grid =
moneyness_grid.detach().to(device=pr
icing_device).double()
    ttm_grid =
ttm_grid.detach().to(device=pricing_
device).double()

    def loss_fn(
        z_pred: torch.Tensor,
z_true: torch.Tensor, batch: dict,
    ) -> torch.Tensor:
        param_loss =
standardized_mse(z_pred, z_true)
        if lambda_repricing == 0.0:
            return param_loss

        if "spot" not in batch or
"r_curve" not in batch:
            raise KeyError(
                "Method 3 loss
requires 'spot' and 'r_curve' in the
batch dict; "
                "build the
dataloader with
include_context=True."
            )

        theta_pred_natural =
standardizer.from_standardized(z_pre
d)

        theta_for_pricing =
theta_pred_natural.to(device=pricing
_device).double()
        iv_true = (
batch["iv_surface"].squeeze(1).to(de
vice=pricing_device).double()
        )
        spot =
batch["spot"].to(device=pricing_devi
ce).double()
        r_curve =
batch["r_curve"].to(device=pricing_d
evice).double()

        rmse = repricing_rmse_iv(
            theta_for_pricing,
iv_true, spot, r_curve,
            moneyness_grid,
ttm_grid,
        )

        if not torch.isfinite(rmse):
            rmse = torch.zeros((),
dtype=rmse.dtype,
device=rmse.device)

        rmse =
rmse.to(dtype=z_pred.dtype).to(devic
e=z_pred.device)

        return param_loss +
lambda_repricing * rmse

    return loss_fn

CANONICAL_COLS = {
    "QUOTE_UNIXTIME",
    "QUOTE_DATE",
    "UNDERLYING_LAST",
    "EXPIRE_DATE",
    "DTE",
    "STRIKE",
    "C_BID",
    "C_ASK",
    "C_IV",
    "C_VOLUME",
    "C_DELTA",
    "C_VEGA",
    "P_BID",
    "P_ASK",
    "P_IV",
    "P_VOLUME",
    "P_DELTA",
    "P_VEGA",
}

class RawDataLoader:
    def __init__(self,
manifest_path="data/raw/manifest.csv
", raw_root="data/raw"):
        self.manifest = pd.read_csv(
            manifest_path,
parse_dates=["start_date",
"end_date"]
        )
        self.raw_root =
Path(raw_root)

    @lru_cache(maxsize=4)
    def _load_file(self, rel_path:
str) -> pd.DataFrame:
        full = self.raw_root /
rel_path
        df = pd.read_csv(full)
        df.columns =
[c.strip().strip("[]").upper() for c
in df.columns]
        keep = [c for c in
df.columns if c in CANONICAL_COLS]
        df = df[keep]
        for col in ("QUOTE_DATE",
"EXPIRE_DATE"):
            if col in df.columns:
                df[col] =
pd.to_datetime(df[col].astype(str).s
tr.strip(" []"))
        return df

    def get_date(self, date) ->
pd.DataFrame:
        date =
pd.Timestamp(date).normalize()
        match = self.manifest[

(self.manifest.start_date <= date) &
(self.manifest.end_date >= date)
        ]
        if match.empty:
            raise KeyError(f"No raw
file covers {date.date()}")
        df =
self._load_file(match.iloc[0].file_p
ath)
        return df[df.QUOTE_DATE ==
date].copy()

MONEYNESS_GRID = np.linspace(0.8,
1.2, 11)
```

```
TTM_DAYS_GRID = np.array([14, 30,
60, 90, 120, 180, 270, 365],
dtype=float)
TTM_GRID = TTM_DAYS_GRID / 365.0

MONEYNESS_MIN, MONEYNESS_MAX = 0.8,
1.2
DTE_MIN, DTE_MAX = 7, 365
IV_MIN, IV_MAX = 0.01, 5.0
MAX_REL_SPREAD = 0.5

MAX_FILL_DISTANCE = 0.05

def _to_numeric(s: pd.Series) ->
pd.Series:
    return
pd.to_numeric(s.astype(str).str.stri
p(" []"), errors="coerce")

def _filter_side(df: pd.DataFrame,
side: str) -> pd.DataFrame:
    bid_col = f"{side}_BID"
    ask_col = f"{side}_ASK"
    iv_col = f"{side}_IV"

    bid = _to_numeric(df[bid_col])
    ask = _to_numeric(df[ask_col])
    iv = _to_numeric(df[iv_col])
    mid = (bid + ask) / 2.0

    with
np.errstate(divide="ignore",
invalid="ignore"):
        rel_spread = (ask - bid) /
mid

    keep = (
        bid.notna()
        & ask.notna()
        & iv.notna()
        & (bid > 0)
        & (ask > 0)
        & (mid > 0)
        & (rel_spread <
MAX_REL_SPREAD)
        & (iv > IV_MIN)
        & (iv < IV_MAX)
    )

    out = df.loc[keep, ["STRIKE",
"TTM", "TTM_DAYS",
"MONEYNESS"]].copy()
    out["bid"] = bid[keep].values
    out["ask"] = ask[keep].values
    out["iv"] = iv[keep].values
    out["mid_price"] =
mid[keep].values
    out["side"] = "C" if side == "C"
else "P"
    return out

def clean_surface(raw_df:
pd.DataFrame, S: float, rate_curve)
-> dict:
    n_raw = len(raw_df)

    df = raw_df.copy()
    df["STRIKE"] =
_to_numeric(df["STRIKE"])
    ttm_days = (df["EXPIRE_DATE"] -
df["QUOTE_DATE"]).dt.days.astype(flo
at)
    df["TTM_DAYS"] = ttm_days
    df["TTM"] = ttm_days / 365.0
    df["MONEYNESS"] = df["STRIKE"] /
float(S)

    in_window = (
        (df["MONEYNESS"] >=
MONEYNESS_MIN)
        & (df["MONEYNESS"] <=
MONEYNESS_MAX)
        & (df["TTM_DAYS"] >=
DTE_MIN)
        & (df["TTM_DAYS"] <=
DTE_MAX)
        & df["STRIKE"].notna()
    )
    df = df.loc[in_window].copy()

    is_put = df["MONEYNESS"] < 1.0
    put_rows =
_filter_side(df.loc[is_put], "P")
    call_rows =
_filter_side(df.loc[~is_put], "C")

    long = pd.concat([put_rows,
call_rows], ignore_index=True)
    long = long.rename(
        columns={"STRIKE": "strike",
"TTM": "ttm", "MONEYNESS":
"moneyness"}
    )
    long = long[["strike", "ttm",
"TTM_DAYS", "moneyness", "iv",
"mid_price", "side"]]

    n_after = len(long)

    iv_surface =
np.full((len(MONEYNESS_GRID),
len(TTM_GRID)), np.nan)
    max_gap = float("inf")
    if n_after >= 4:
        points = long[["moneyness",
"ttm"]].to_numpy()
        values =
long["iv"].to_numpy()
        Mg, Tg =
np.meshgrid(MONEYNESS_GRID,
TTM_GRID, indexing="ij")
        target =
np.column_stack([Mg.ravel(),
Tg.ravel()])

        try:
            linear =
griddata(points, values, target,
method="linear")
        except Exception:
            linear =
np.full(target.shape[0], np.nan)
        nearest = griddata(points,
values, target, method="nearest")

        tree = cKDTree(points)
        dists, _ =
tree.query(target, k=1)
        max_gap = float(dists.max())

        filled =
np.where(np.isnan(linear), nearest,
linear)
        filled = np.where(
            np.isnan(linear) &
(dists > MAX_FILL_DISTANCE), np.nan,
filled
        )
        iv_surface =
filled.reshape(len(MONEYNESS_GRID),
len(TTM_GRID))

    coverage_fraction =
float(np.isfinite(iv_surface).mean()
)

    rate_lookup = {}
    quote_date =
pd.Timestamp(raw_df["QUOTE_DATE"].il
oc[0]).normalize()
    for ttm_d in TTM_DAYS_GRID:
        rate_lookup[float(ttm_d)] =
float(rate_curve.get_rate(quote_date
, float(ttm_d)))

    return {
        "date": quote_date,
        "spot": float(S),
        "rate_curve": rate_lookup,
        "moneyness_grid":
MONEYNESS_GRID.copy(),
        "ttm_grid": TTM_GRID.copy(),
        "iv_surface": iv_surface,
        "n_raw_quotes": int(n_raw),
        "n_after_filter":
int(n_after),
        "coverage_fraction":
coverage_fraction,
        "max_gap": max_gap,
    }

class RateCurve:
    def __init__(self, rates_path:
str | Path = "data/rates.parquet"):
        df =
pd.read_parquet(rates_path)
        df["date"] =
pd.to_datetime(df["date"]).dt.normal
ize()
        df = df.sort_values(["date",
"tenor_days"]).reset_index(drop=True
)
        self._df = df
        self._dates =
np.array(sorted(df["date"].unique())
)
        self._by_date:
dict[pd.Timestamp, tuple[np.ndarray,
np.ndarray]] = {}
        for d, sub in
df.groupby("date"):

self._by_date[pd.Timestamp(d)] = (

sub["tenor_days"].to_numpy(dtype=flo
at),

sub["rate"].to_numpy(dtype=float),
            )

    def _resolve_date(self, date) ->
pd.Timestamp:
        d =
pd.Timestamp(date).normalize()
        if d in self._by_date:
            return d
        idx =
np.searchsorted(self._dates,
np.datetime64(d), side="right") - 1
        if idx < 0:
            return
pd.Timestamp(self._dates[0])
        return
pd.Timestamp(self._dates[idx])

    def get_rate(self, date,
ttm_days: float) -> float:
        d = self._resolve_date(date)
        tenors, rates =
self._by_date[d]
        ttm = float(ttm_days)
        if ttm <= tenors[0]:
            return float(rates[0])
        if ttm >= tenors[-1]:
            return float(rates[-1])
        return float(np.interp(ttm,
tenors, rates))

    def get_rates(self, date,
ttm_days_array) -> np.ndarray:
        d = self._resolve_date(date)
        tenors, rates =
self._by_date[d]
        arr =
np.asarray(ttm_days_array,
dtype=float)
        out = np.interp(arr, tenors,
rates, left=rates[0], right=rates[-
1])
        return out
```

ДОДАТОК Б – ІЛЮСТРАТИВНІ МАТЕРІАЛИ ДОПОВІДІ

Система калібрування моделі стохастичної волатильності Гестона на основі гібридного нейромережевого підходу

Шевцов Олександр Олександрович

Керівник: асистент кафедри ММСА Древаль Максим Михайлович

Інститут прикладного системного аналізу Національного технічного університету України
«Київський політехнічний інститут імені Ігоря Сікорського»

Актуальність

Модель Гестона залишається галузевим стандартом оцінювання індексних опціонів. Її класичне калібрування потребує десятків мілісекунд на одну поверхню, що обмежує застосовність у режимі реального часу.

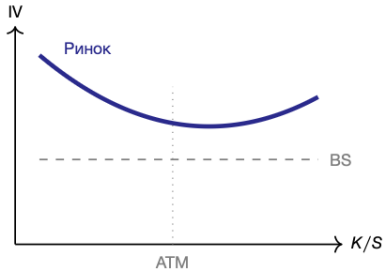
1. Швидкодія. Системи ризик-менеджменту потребують онлайн-перерахунку параметрів моделі для портфелів з тисячами опціонних позицій.
2. ML у фінансах. Нейромережеві калібратори дають інференс на два порядки швидший, але страждають від розподілового зсуву синтетика-ринок.
3. Фізично-інформоване навчання. Гібридні функції втрат з прямомодельним переоцінюванням поєднують переваги класичних та нейромережевих підходів.
4. Бенчмарк-методологія. Стандартні метрики не розрізняють обмежень модельного класу від похибки конкретного методу.

Предметна область

Калібрування афінних моделей стохастичної волатильності на спостережуваних ринкових поверхнях неявної волатильності опціонів.

Поверхня неявної волатильності — двовимірна сітка значень $\sigma(K/S, T)$ за грошовістю та строком до експірації, отриманих інверсією формули Блека — Шоулза з ринкових цін опціонів.

Спостережувана форма поверхні (усмішка та косий перекіс) систематично порушує припущення сталої волатильності класичної моделі Блека — Шоулза, що мотивує використання моделей стохастичної волатильності — насамперед моделі Гестона (1993).



Усмішка волатильності — порушення припущення сталої σ

Об'єкт та предмет дослідження

Об'єкт дослідження

Процес калібрування афінних моделей стохастичної волатильності на спостережуваних ринкових поверхнях неявної волатильності опціонів.

Предмет дослідження

Гібридні нейромережеві методи калібрування моделі Гестона з фізично-інформованою компонентою функції втрат на основі диференційовного оцінювача опціонів за методом косинусного розкладання.

Мета та завдання дослідження

Мета роботи

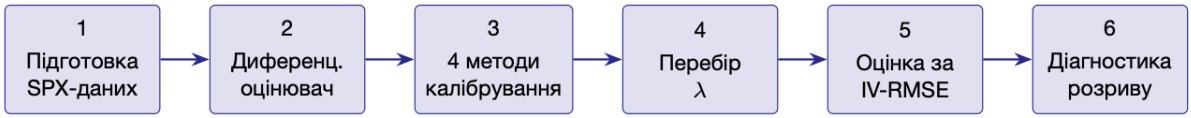
Розробити та порівняти чотири методи калібрування моделі стохастичної волатильності Гестона з систематичним дослідженням ролі фізично-інформованої компоненти функції втрат та запропонувати діагностику якості калібрування на ринкових даних.

Завдання дослідження:

1. Реалізувати диференційовний оцінювач опціонів за методом косинусного розкладання Фанг — Остерлі.
2. Імплементувати чотири методи калібрування: наївний еталон, Левенберг — Марквардт, нейромережевий MSE-калібратор, гібридний фізично-інформований калібратор.
3. Провести експерименти на 500 тис. синтетичних та 178 ринкових поверхнях S&P 500 за 2017 — 2022.
4. Виконати порівняльний аналіз чотирьох методів та запропонувати діагностичну методологію.

Алгоритм розв'язку

Завдання — розробити інформаційно-аналітичну систему калібрування моделі Гестона на ринкових даних S&P 500 з порівняльним аналізом чотирьох методів за єдиним протоколом IV-RMSE.



Кожен крок інтегровано у спільний експериментальний протокол; усі чотири методи оцінюються на тестовому наборі з 178 ринкових поверхонь індексу S&P 500 за період 2017-2022.

Модель Гестона

Стохастичні диференціальні рівняння у нейтральній до ризику мірі:

$$dS_t = (r - q)\, S_t\, dt + \sqrt{v_t}\, S_t\, dW_t^S,$$
$$dv_t = \kappa(\theta - v_t)\, dt + \sigma_v \sqrt{v_t}\, dW_t^v, \quad \langle dW^S, dW^v \rangle = \rho\, dt.$$

П'ять параметрів калібрування: $\Theta = (\kappa, \theta, \sigma_v, \rho, v_0) \in \mathbb{R}^5$.

Замкнутий вираз ціноутворення

Характеристична функція логарифма ціни активу має експоненційно-афінну форму $\varphi(u; T, \Theta) = \exp\{C(u,T) + D(u,T)\, v_0 + iu(\ln S_0 + (r-q)T)\}$. Ціна опціона обчислюється через косинусне розкладання Фанг — Остерлі за мілісекунди, без симуляції Монте-Карло на кожному кроці оптимізатора.

Огляд чотирьох методів

Класичні підходи

1. Метод 0 — наївна екстраполяція з попереднього дня; визначає нижню межу якості.
2. Метод 1 — оптимізатор Левенберга — Марквардта; галузевий еталон точності.

Нейромережеві підходи

3. Метод 2 — згорткова мережа з MSE у просторі параметрів моделі.
4. Метод 3 — гібрид з фізично-інформованою компонентою (вагою λ).

Гібридна функція втрат

$L_{\text{гібр}}(w; \lambda) = L_{\text{парам}}(w) + \lambda \cdot L_{\text{переоц}}(w)$ — поєднує MSE-помилку параметрів та помилку IV-RMSE реконструйованої поверхні через диференційовний оцінювач. Перебір $\lambda \in \{0{,}05;\ 0{,}20;\ 0{,}70;\ 2{,}00\}$ для аналізу ролі фізичної регуляризації.

Експериментальні дані

Синтетичний набір (для тренування)

1. 500 тис. поверхонь IV
2. Емпіричний апріорний розподіл параметрів Θ
3. Сітка 11×8 (грошовість $\times$ строк)
4. Поділ 90/5/5 (тренування/валідація/тест)

Ринковий набір (для тестування)

1. 178 поверхонь S&P 500
2. Період квітень 2017 — грудень 2022
3. Котирування OptionsDX
4. Ставки FRED Federal Reserve
5. 5 ринкових режимів (`covid_2020`, `stress_2022`)

Метрика оцінювання

IV-RMSE переоцінювання — корінь з середньоквадратичного відхилення між вхідною поверхнею та реконструйованою через диференційовний оцінювач від каліброваних параметрів. Єдиний протокол для всіх чотирьох методів на синтетичних і ринкових даних.

Технології

Програмний продукт розроблено мовою програмування Python з використанням сучасного стека наукових обчислень та машинного навчання.

Джерела даних: OptionsDX (котирування S&P 500), FRED (крива безризикових ставок). Інфраструктура: локальне обчислювальне середовище, Jupyter Notebook для прототипування.

Результати на синтетичних даних

Метод	IV-RMSE	MAE κ	MAE θ	Час, мс
Метод 0 (наївний)	-	4,720	0,098	0,001
Метод 1 (LM)	**0,0075**	0,065	0,001	49,07
Метод 2 (CNN, MSE)	0,0089	0,283	0,054	0,27
Метод 3 ($\lambda = 0{,}05$)	**0,0080**	0,112	0,052	0,27

Метод 1 досягає найвищої точності IV-RMSE = 0,0075. Гібрид з $\lambda = 0{,}05$ покращує чисто-нейромережевий Метод 2 на 10 % і наближається до Методу 1 при швидкості інференсу в 180 разів вищій. Параметричні похибки κ, θ також помітно зменшуються, що підтверджує функціональну роль вагового коефіцієнта λ.

Результати на ринкових даних та фактор розриву

Метод	Синт. IV-RMSE	Ринок IV-RMSE	Фактор розриву
Метод 1 (LM)	0,0075	0,0169	**2,25**×
Метод 2 (CNN)	0,0089	0,0203	2,28×
Метод 3 ($\lambda = 0{,}05$)	0,0080	0,0260	3,25×
Метод 3 ($\lambda = 0{,}70$)	0,0087	**0,0203**	**2,33**×

Фактор розриву = ринкова IV-RMSE / синтетична IV-RMSE. Класичний Метод 1 не залежить від тренувального розподілу, тож рівень $\approx 2{,}25\times$ характеризує нижню межу похибки в класі моделі Гестона. На стресовому періоді `covid_2020` усі методи показують спільне погіршення — це підтверджує природу обмеження як характеристики модельного класу, а не методу.

Валідаційно-ринкова інверсія: $\rho_{\text{Спірмен}} = 0$

Ранжування значень λ за синтетичною валідаційною IV-RMSE та за середньою ринковою IV-RMSE повністю декорельовані.

Синтетична валідація обирає $\lambda = 0{,}05$;
ринкова метрика обирає $\lambda = 0{,}70$.

Коефіцієнт рангової кореляції Спірмена $\rho = 0$ — валідаційний рейтинг не несе інформації про ринковий.

Стандартна ML-практика «обирати гіперпараметр на валідаційному наборі» у задачі калібрування Гестона не переноситься з синтетики на реальні ринкові дані.

λ	Синт.	Ринок	Р.С.	Р.Р.
0,05	0,00909	0,0260	**1**	3
0,20	0,00968	0,0267	3	4
0,70	0,00958	0,0203	2	**1**
2,00	0,00984	0,0215	4	2

Швидкодія інференсу

Метод	Інференс	Прискорення
Метод 1 (LM)	49,0731 мс	1×
Метод 2 (CNN)	0,2741 мс	179,03×
Метод 3 (гібрид)	0,2726 мс	**180,01**×

180×

швидше за класичний LM

на два порядки вище інференсу

Природне розділення сфер застосування: дослідницькі задачі з пріоритетом точності — Метод 1; онлайн-системи потокового та інтрадейного калібрування — Методи 2 і 3.

Висновки

1. Реалізовано та порівняно чотири методи калібрування моделі Гестона на 178 ринкових поверхнях S&P 500 за єдиним протоколом IV-RMSE.
2. Гібридний нейромережевий калібратор з фізично-інформованою компонентою досягає 180× прискорення відносно класичного LM при співмірній точності, що відкриває можливості онлайн-калібрування портфелів.
3. Виявлено валідаційно-ринкову інверсію ($\rho_{\text{Спірмен}} \approx 0$): підбір λ на синтетичній валідації не переноситься на ринкову метрику.
4. Запропоновано фактор розриву синтетика-ринок як діагностику, що відокремлює обмеження модельного класу ($\approx 2{,}25\times$ для Гестона) від похибки конкретного методу калібрування.

Подальші дослідження

1

Багатші моделі

Перенесення методології на моделі грубої волатильності (rBergomi), стрибкові (Bates) та локально-стохастичні розширення.

2

Адаптивне λ

Дослідження адаптивних схем балансування λ (WamOL, PDC-PINN) з ринково-обізнаним валідаційним протоколом.

3

Інтеграція

Розгортання гібридного калібратора у виробничих системах переоцінювання портфелів з підтримкою потокових даних.

Дякую за увагу

Дякую за увагу!

Шевцов Олександр Олександрович

КА-21 · кафедра ММСА · НН ІПСА

КПІ ім. Ігоря Сікорського