КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

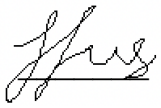
Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

Кваліфікаційна робота

на здобуття ступеня бакалавра
за спеціальністю 113 Прикладна математика
на тему:

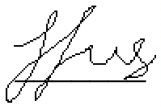
Прогнозування ціни біткоїна засобами математичного аналізу

Виконав студент 4-го курсу
Сидоренко Нікіта Сергійович



Науковий керівник:
Доктор філософії
Асистент кафедри обчислювальної математики
Тимошенко Андрій Анатолійович

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент 

Роботу розглянуто й допущено до захисту на засіданні кафедри обчислювальної математики
«29» травня 2023р.,
протокол № 8
Завідувач кафедри
Ляшко Сергій Іванович 

Київ – 2023

ЗМІСТ

АНОТАЦІЇ

Дипломна робота складається зі вступу, 2 основних розділів, допоміжних застосунків, висновків, списку використаних джерел (8) та додатків з кодом програми (3). Загальний обсяг роботи становить 58 сторінок, основний текст роботи викладено на 26 сторінках.

Ключові слова: ціна біткоїна, похідні, Монте Карло, фази ринку, сплайн інтерполяція, тренд, акумулювання.

Реферат. В роботі запропоновані підходи до роботи з похідними функції ціни біткоїна. Реалізована Симуляція Монте-Карло для отримання імовірнісного прогнозу. Реалізована сплайн інтерполяція 3го степеня, за допомогою якої створений індекс сили тренду, який враховує швидкість зміни ціни та об’єм торгів. Також було реалізовано пошук акумуляцій, що доповнює описані методи до практичного використання.

Keywords: Bitcoin price, derivatives, Monte Carlo, market phases, spline interpolation, trend, accumulation.

Abstract. The paper proposes approaches to working with derivatives of the bitcoin price function. Implemented Monte Carlo simulation to obtain a probabilistic forecast. Implemented spline interpolation of the 3rd degree, with the help of which an index of trend strength is created, which takes into account the speed of price change and trading volume. The search for accumulations was also implemented, which complements the described methods for practical use.

ВСТУП

Загальні відомості

Криптовалюта – цифрова або віртуальна форма валюти, що використовує криптографію для проведення транзакцій. У криптовалют немає якогось центрального регулятора, замість цього для здійснення транзакцій використовується децентралізована система. Наведемо деякі приклади криптовалют:

Bitcoin(BTC) – перша криптовалюта. Був заснований ще в 2009 році, проте досі має найбільший об’єм торгів. Засновником є Сатоші Накамото – псевдонім, під яким анонімно ховається людина або група людей.

Ціна криптовалют неперервно змінюється протягом часу. За динамікою ціни зручно спостерігати на графіках. Вони бувають таких двох основних типів:

Лінійний – з’єднує ціни валюти кожну задану одиницю часу.

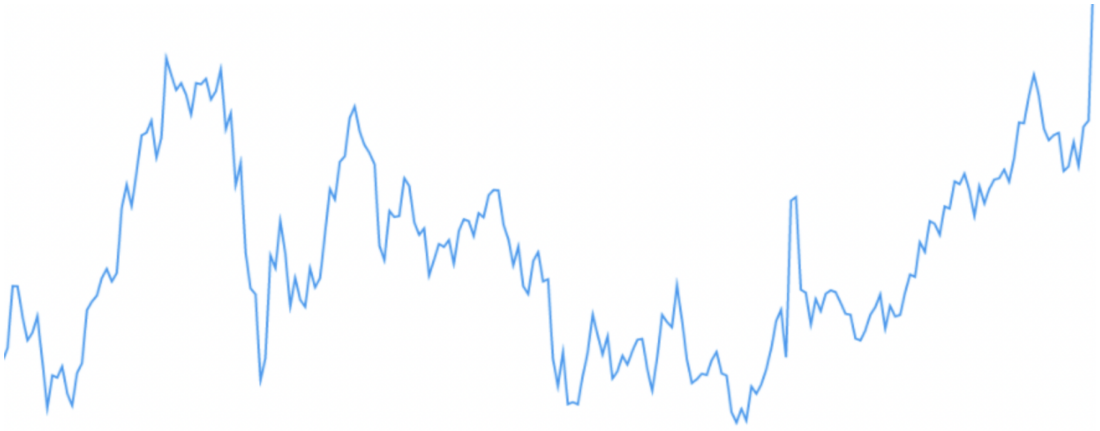


Рис. 1 Лінійний графік Біткоїна

Свічковий – для кожного з однакових періодів показує екстремуми ціни, ціну на початку та на кінці періоду.

Рис. 2 Свічковий графік Біткоїна

Основною характеристикою графіку є timeframe (TF) – одиниця часу, на які поділяється графік. Кожна свічка відповідає одному періоду TF. Великі TF використовуються для більш довгострокових прогнозів, а маленькі – для локальних.

Висока волатильність відкриває можливості робити прибуткові інвестиції та спекуляції. Для цього інвестори аналізують ринок, розробляють певні сценарії. Є два основні види аналізу: технічний та фундаментальний.

Технічний аналіз – аналіз, який робить висновки виключно опираючись на історію ціни певної криптовалюти. Такий підхід допомагає добре локально розуміти ситуацію та знаходити вдало точки входу та виходу з угоди. Даний аналіз потребує гарного розуміння ринку.

Фундаментальний аналіз – враховує показники проекту, новини, перспективи. Є важливою частиною в прогнозуванні ціни. Допомагає зрозуміти глобальну ситуацію. Це не тільки пошук новин – це комплексний аналіз проекту і ситуації.

У ході роботи ми розглянемо певні рішення для технічного аналізу.

Ціна як функція від часу

Розгляд ціни як функції від часу відкриває певні можливості для аналізу графіків. Але в чистому вигляді до функції ціни не можна застосувати певні методи хоча б через не диференційованість. Навіть лінійний графік, що виглядає досить подібно до гладкої функції, насправді являє собою з’єднані точки в певні моменти часу, тобто сплайн 1 степеня.
Справді, розглянемо лінійний графік:

Рис. 3 Лінійний графік Біткоїна за 30 днів

А тепер наблизимо:

Рис. 4 Лінійний графік Біткоїна за 14 днів

А тепер ще:

Рис. 5 Лінійний графік Біткоїна за 24 години

І скільки б ми не приближали, ми не отримаємо гладку криву.

Отже, похідні в чистому вигляді не мають сенсу для даної функції. Проте вони можуть бути нам корисними, адже похідні як дають певну інформацію про функцію, описуючи швидкість зміни, так і задають саму функцію за допомогою інтегрування.

В ході роботи ми розглянемо підходи, які замінять звичайні похідні для виконання цих двох задач.

РОЗДІЛ 1
ПІДХОДИ ДО ПОХІДНИХ ДЛЯ ФУНКЦІЇ ЦІНИ

1.1 Аналіз руху ціни

Часто для певних висновків аналізують рух ціни останнім часом - наскільки він сильний. Рух ціни в одному напрямку називатимемо трендом. Зробти висновки про силу тренду дають можливість багато параметрів, основні з яких об’єм та швидкість зміни ціни. Тут би в нагоді нам стали похідні, які не можемо знайти на графіках які маємо. Для знаходження неперервних похідних від функції ціни зробимо сплайн-інтерполяцію 3 степеня[7], яка може гарантувати неперервність та диференційовність функції. Сплайн інтерполяція - це представлення функції у вигляді набору поліномів, кожен з яких задає значення на відповідному відрізку. Степінь інтерполяції - це степінь цих поліномів.

Розглянемо як це працює на прикладах. Точки показують відомі з минулого екстремуми цін на кожній одиниці TF. Лінія - об’єднання інтерполяційних поліномів.

Рис. 6 Сплайн-інтерполяція 3 степеня графіку Біткоїна за 120 днів

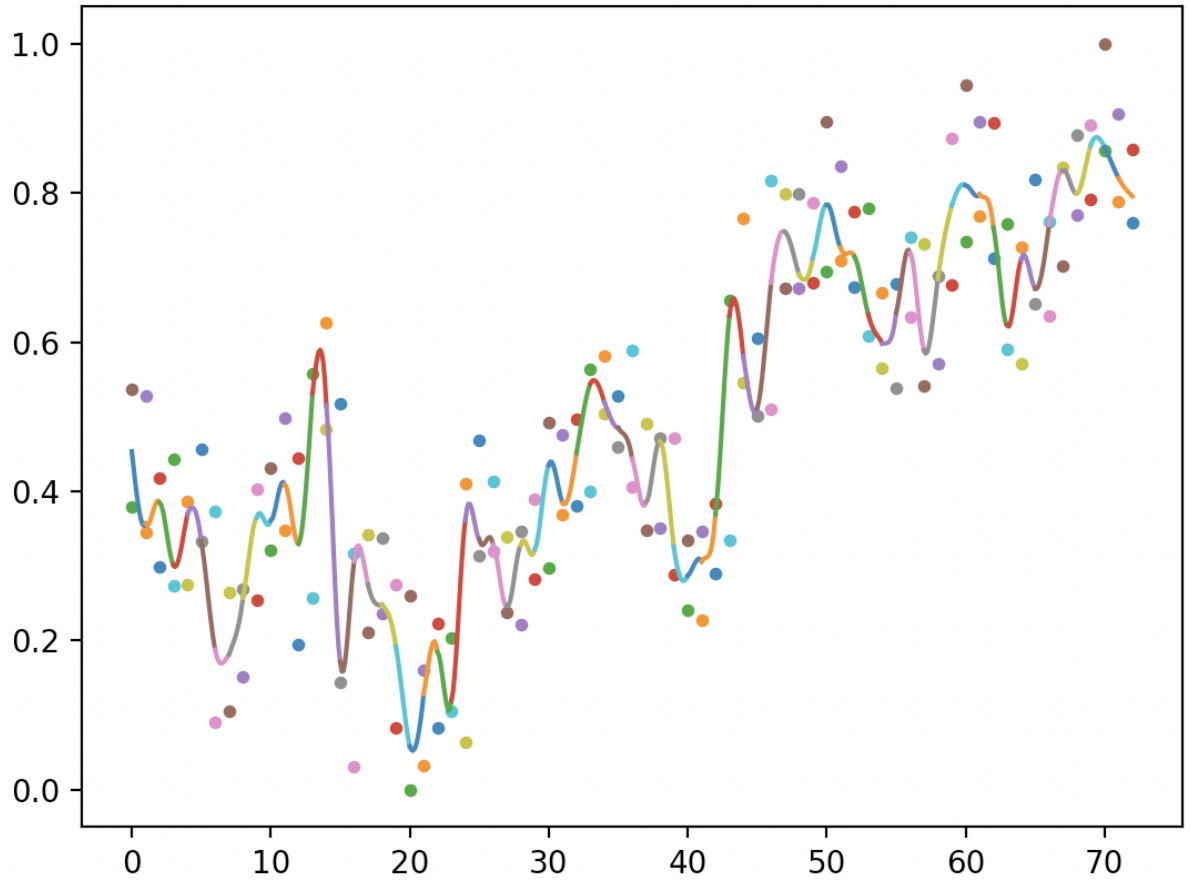


Рис. 7 Сплайн-інтерполяція 3 степеня графіку Біткоїна за 70 днів

Отримуємо гладкі функції, які повторюють рухи ціни.

Але в такому вигляді функція не дуже корисна для дослідження, адже через високу волатильність ринку похідні постійно змінюють знак, причому мають великі абсолютні значення. Для вирішення цієї проблеми зробимо згладжування функції - в кожній точці обчислимо нове значення як середнє арифметичне її самої та кількох точок з обох сторін.

Візьмемо 1 точку з кожної сторони, згладимо останній графік:

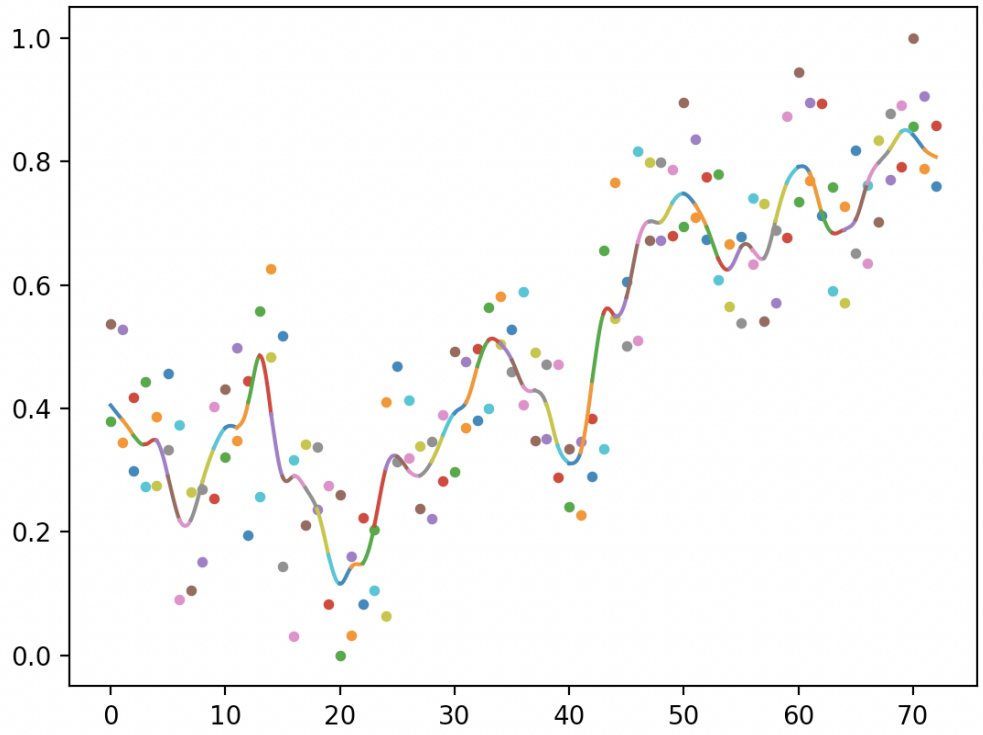


Рис. 8 Сплайн-інтерполяція згладженого графіку, беручи значення як середнє між значенням точки та значеннями 1 сусіда з кожної сторони

Все ще маємо забагато шумів. Візьмемо 2:

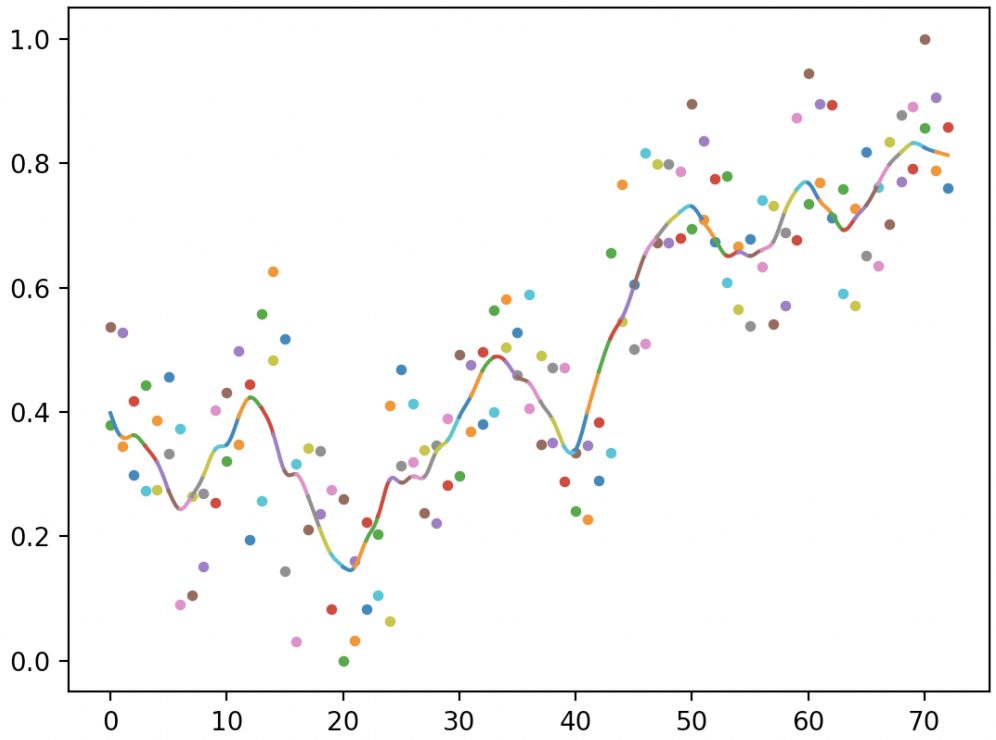


Рис. 9 Сплайн-інтерполяція згладженого графіку, беручи значення як середнє між значенням точки та значеннями 2 сусіда з кожної сторони

Чудовий результат. Тепер розглянемо похідну отриманої функції, на графіку зверху сплайн, а знизу його похідна:

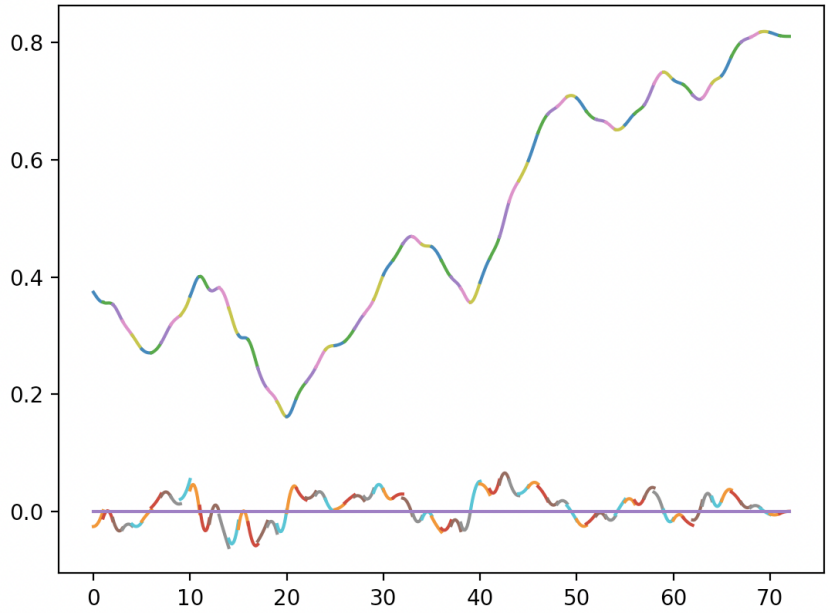


Рис. 10 Верхня крива - сплайн-інтерполяція функції, нижня крива - її похідна

Для більш універсального індикатору сили тренду домножимо похідну на нормований об’єм та отримаємо:

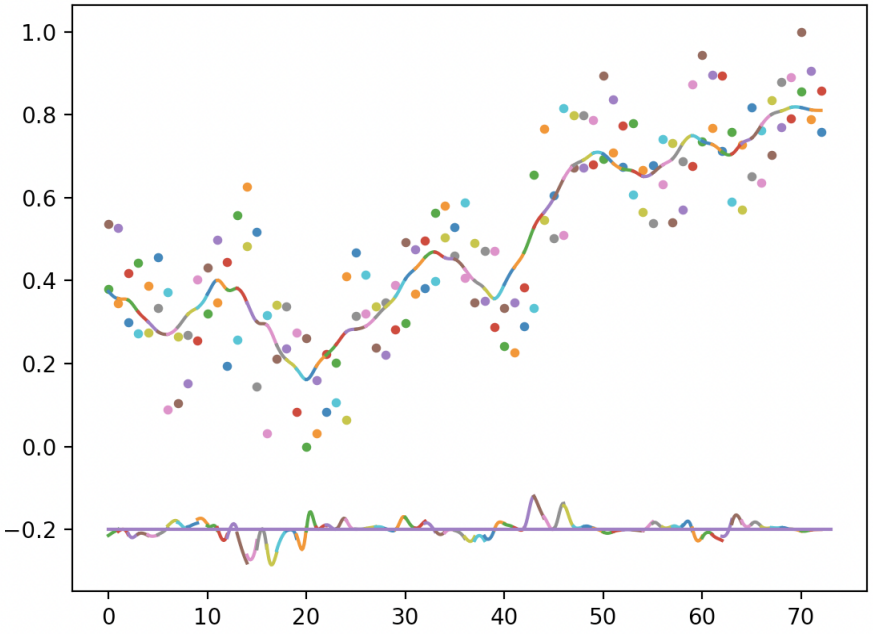


Рис. 11 Верхня крива - сплайн-інтерполяція функції, нижня крива - добуток похідної на нормований об’єм

Цей індекс враховує обидва основні показники сили тренду - швидкість зміни ціни та об’єм торгів. Цей підхід дозволяє розглядати неперервну похідну від функції ціни.

1.2 Стохастичний аналіз

1.2.1 Перша модель

В процесі нашої задачі присутня випадкова складова. В такому випадку закон зміни ціни може мати вигляд геометричного броунівського руху[5]:

$$dS = S(\mu dt + \sigma \epsilon dt^{0.5}), \text{ де:}$$

S – поточна ціна,

μ – очікувана зміна в ціні за проміжок TF

$$\mu = \sum \frac{\text{Зміна ціни за 1 } TF}{\text{Кількість } TF \text{ в даних}},$$

dt – одиниця часу, в нашому випадку 1(TF),

ϵ – нормально розподілена випадкова змінна

$$\epsilon \sim N(0, 1),$$

σ – вибіркове стандартне відхилення

$$\sigma = \sqrt{\sum \frac{(\text{Зміна ціни за 1 } TF - \mu)^2}{\text{Кількість } TF \text{ в даних} - 1}}$$

Розглянемо симуляцію ціни для отримання імовірнісного прогнозу. Основним методом, яким користуються багато інвесторів, є Монте Карло. Він допомагає робити прогнози з певною надійністю.

Процес блукання ціни розглядається як Марківський процес, за крок беремо 1 одиницю часу(TF). Метод полягає в багатьох симуляціях. Після симуляцій результати підраховуються і отримується розподіл імовірностей щодо результуючої ціни.

Перейдемо до реалізації. Нижче зображені результати роботи симуляції та оригінальні частини графіку з вхідних даних. Фіолетові лінії показують очікуване значення і також інтервал довіри з надійністю 50%. Точки показують відомі з минулого екстремуми цін на кожній одиниці TF.

Рис. 12 Три результати моделювання руху ціни Біткоїна першим методом

Проте під час тестування я помітив проблему: в такій моделі симуляція з однаковою імовірністю іде вниз та вверх на одну відстань. А ці кроки не рівноцінні.

Наприклад ми стоїмо на ціні S та ідемо на 50% вниз. Тоді щоб повернутися на S, нам потрібно піти вверх на 100%. Якщо ж ми підемо на 50% вверх, щоб відновитися нам треба піти вниз лише на 33%. Таким чином наша симуляція буде понижувати ціну.

Нижче наведено приклад такої поведінки:

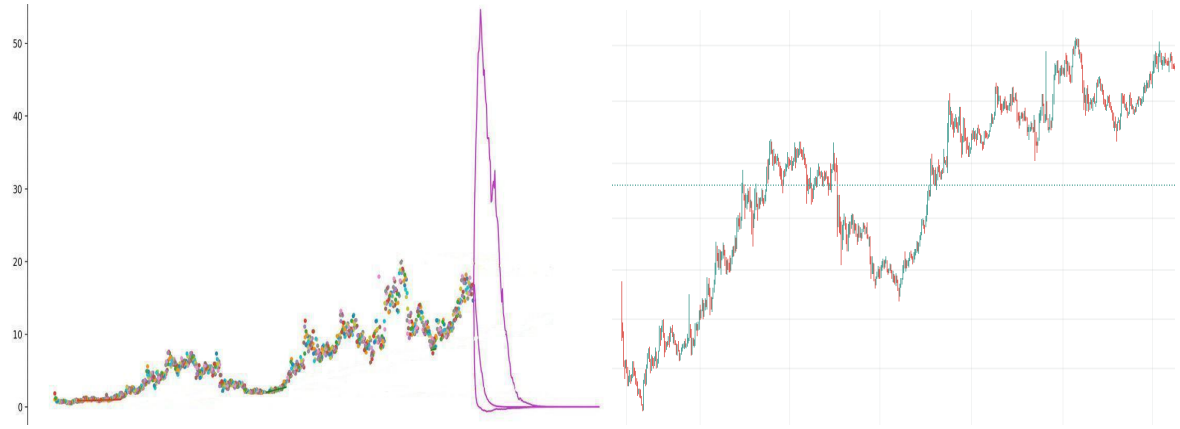


Рис. 13 Поганий результат моделювання руху ціни біткоїна за великий період часу першим методом

Це відбувається тому, що на великих TF при великому часовому проміжку ми отримуємо велике середнє відхилення, що значно збільшує кроки. І з кожним великим кроком вниз, ціна не може відновитися. Проблема не сильно виражається на менших TF, як видно на перших тестах.

1.2.2 Друга модель

Для виправлення цієї проблеми будемо використовувати іншу модель[1]:

$$S_1 = S_0 e^{drift+\epsilon\sigma}$$, де:

S_1 – ціна на наступному кроці,

S_0 – поточна ціна,

ϵ – нормально розподілена випадкова змінна

$$\epsilon \sim N(0, 1),$$

σ – вибіркове стандартне відхилення

$$\sigma = \sqrt{\sum \frac{\left(ln\left(\frac{\text{Нова ціна}}{\text{Стара ціна}}\right) - \mu\right)^2}{\text{Кількість } TF \text{ в даних} - 1}},$$

$$\text{drift} = \mu - \frac{\sigma^2}{2},$$

μ – очікуваний натуральний логарифм відношення ціни на кінці TF до ціни на початку цього ж TF.

$$\mu = \sum \frac{ln\left(\frac{\text{Нова ціна}}{\text{Стара ціна}}\right)}{\text{Кількість } TF \text{ в даних}}$$

Протестуємо цей підхід

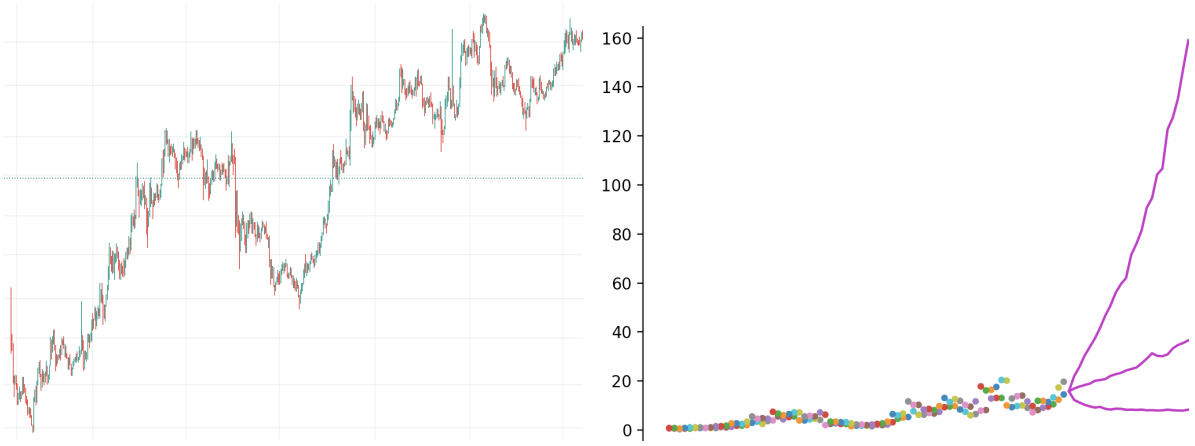


Рис. 14 Моделювання руху ціни біткоїна другим методом на останньому прикладі

На тому ж наборі даних нова модель працює добре. Зробимо ще кілька тестів:

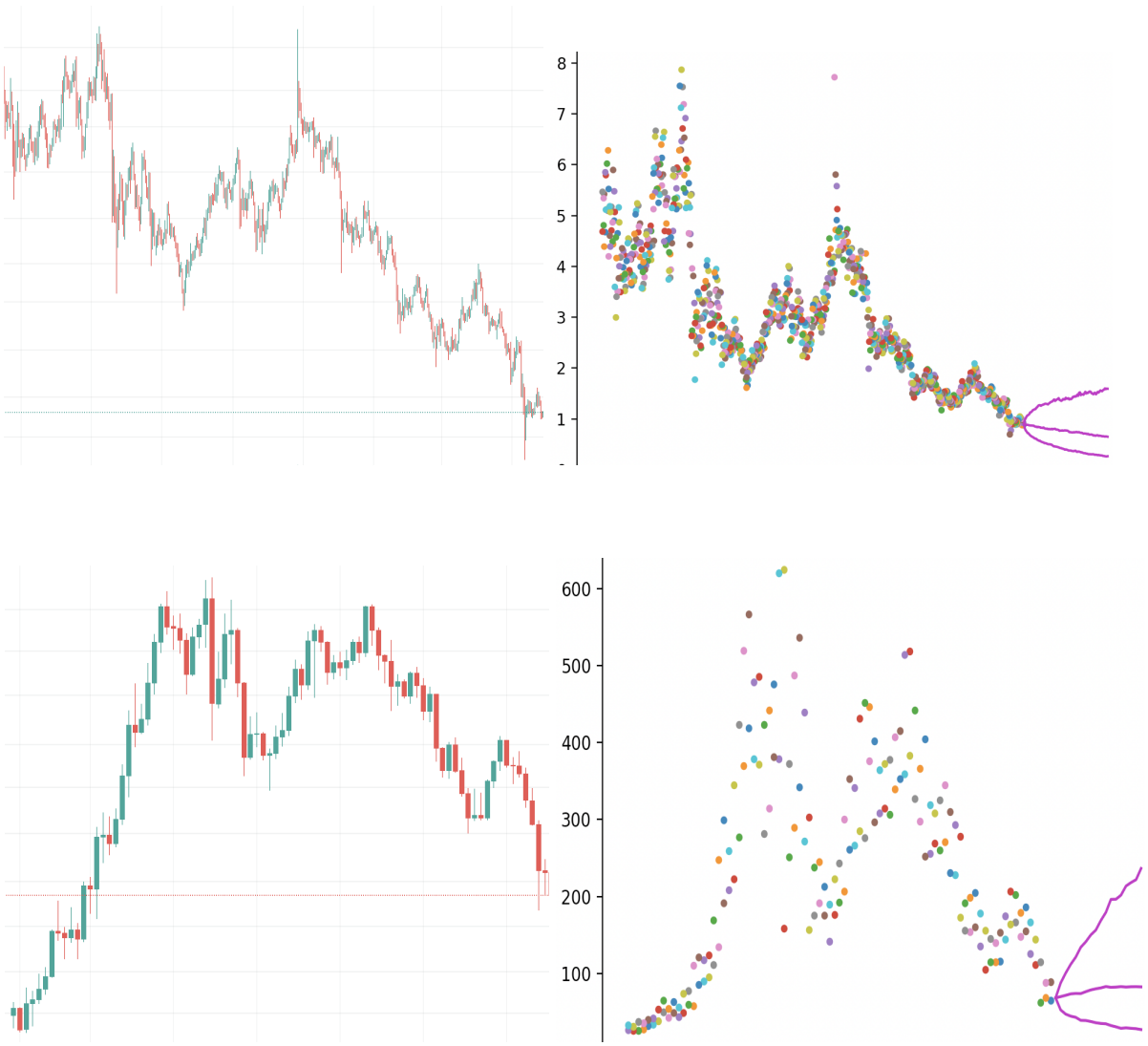


Рис. 15 Два результати моделювання руху ціни Біткоїна другим методом

Переглянувши результати роботи симуляції, бачимо, що вона досить добре відчуває загальний тренд ціни. Такий підхід є аналогом звичайного інтегрування похідної для знаходження значення функції. Але загального тренду не завжди вистачає для прогнозування ціни, адже ринок циклічний. Опишемо це в наступному розділі.

РОЗДІЛ 2
ФАЗИ РИНКУ

2.1 Введення

У ринку є три основні стани: висхідний тренд, низхідний тренд та боковий рух. Більшість часу ринок знаходиться у стадії бокового руху і тільки інколи проходять якісь імпульси зі сторони покупця або продавця. Основні принципи аналізу циклів ринку описав Річард Вайкофф.
Перший закон Вайкоффа - про попит і пропозицію. Тобто перевага попиту над пропозицією призводить до росту цін. Перевага пропозиції над попитом – до падіння цін. Рівність призводить до стабільної, низько-волатильної ситуації. Оцінити попит і пропозицію можна по об’ємам торгів на свічках.
Другий закон Вайкоффа наголошує, що різниця між попитом і пропозицією не випадкові, а показують підготовчі дії до певного руху.
Третій закон наголошує, що зміна в ціні є результатом спільних зусиль, які відображені на об’ємі торгів. Якщо рух супроводжувався великим об’ємом, можна очікувати його продовження. Якщо ж малим – то він може обірватися в будь-який момент.

Розглянемо фази ринку за Вайкоффом[3]:

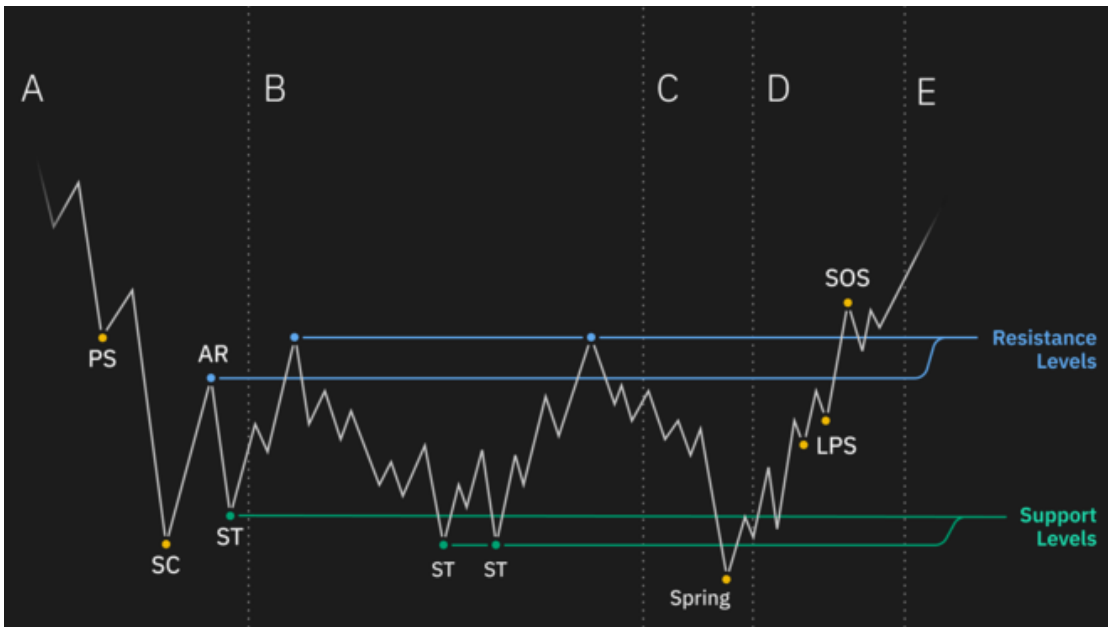


Рис. 16 Фази ринку за Вайкоффом

В фазі А сила продаж зменшується і низхідний тренд затухає. Цей момент часто супроводжується збільшенням об’ємів торгів.

Фазу В можна розглядати як боковий рух, де великі гравці накопичують позицію.

Фаза С – ще один період накопичення активів а також вивід слабких гравців з ринку.

Фаза D – перехід від причини до наслідку, зазвичай супроводжується збільшенням об’ємів.

Фаза Е – остання фаза, яка є пробоєм діапазону, що є знаком початку нового тренду.

2.2 Тренди

Тренд – це рух ціни в певному напрямку протягом певного часу. Для глибшого розуміння графіку будемо шукати тренди. Знайдемо точки екстремуму описаної раніше сплайн-інтерполяції, це і будуть розвороти –

точки початки і кінці трендів. Сплайн у нас 3 степеня, отже пошук екстремума зводиться до розв’язку квадратного рівняння на відрізку. Розв’язувати будемо його наступним чином - крокуємо по відрізку малим кроком. Вважаємо що в кожному кроці може міститися лише один корінь, адже крок у нас малий. Якщо на поточному кроці значення похідної на кінцхях відрізка різних знаків, ми шукаємо 0 функції бінарним пошуком.

Зобразимо знайдені точки екстремуму на графіку, що розглядали раніше:

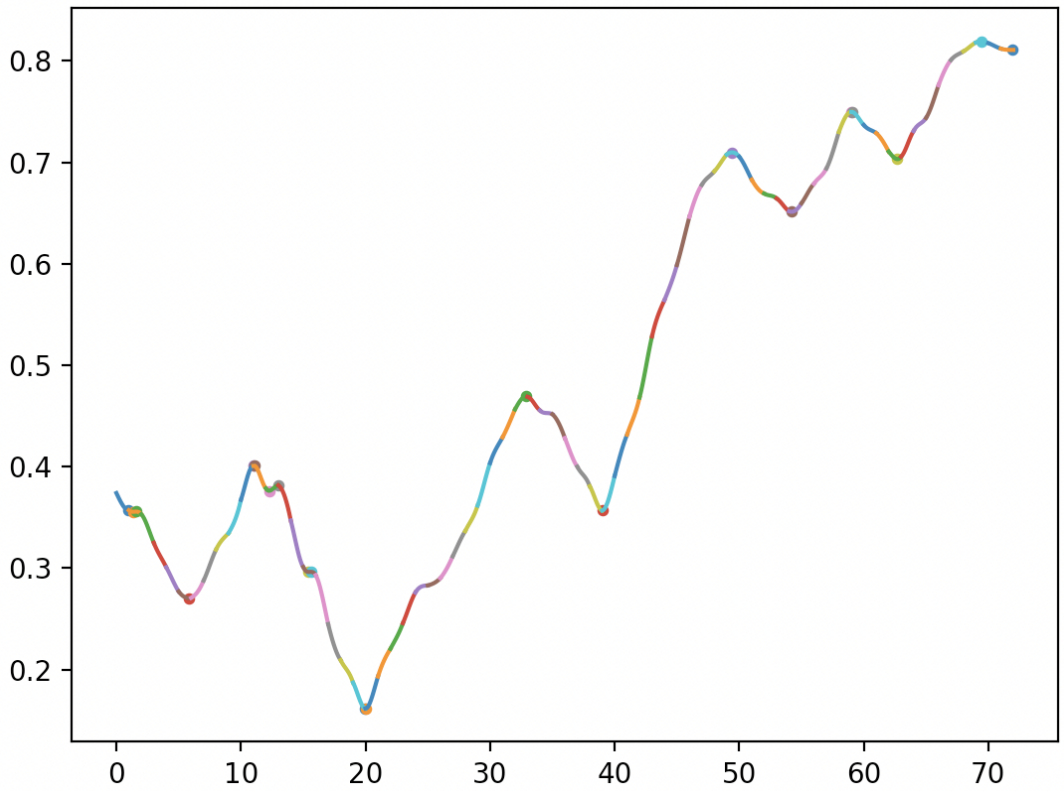


Рис. 17 Точки екстремумів на графіку інтерполяції

Екстремуми позначені точками на графіку сплайна.

Зобразимо тренди як рухи ціни між сусідніми екстремумами:

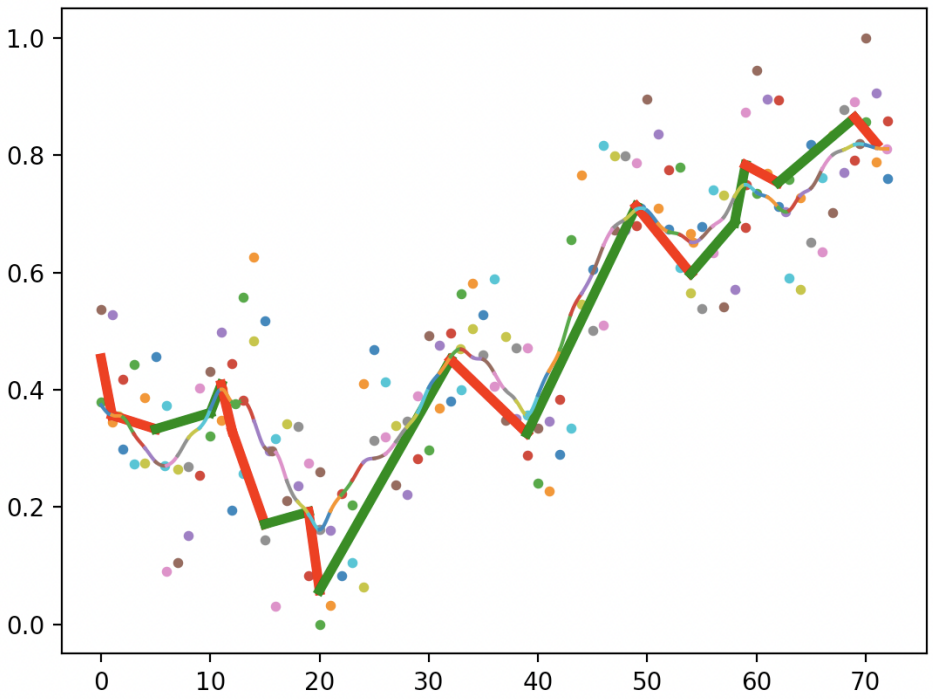


Рис. 18 Графік інтерполяції з зображеними трендами

Бачимо, що отримуємо невеликі рухи цін як окремі тренди. Хочемо виділити найсильніші тренди. Для цього використаємо індекс сили тренду описаний раніше - а саме будемо брати інтеграл на відрізку тренду, щоб оцінити його. Зробимо це поділивши відрізок на маленькі кроки, сумуючи усереднене значення індексу на кожному. Сильними вважаємо 10% усіх трендів з найбільшими оцінками.

Виведемо сильні тренди жирними лініями, а слабкі тонкими:

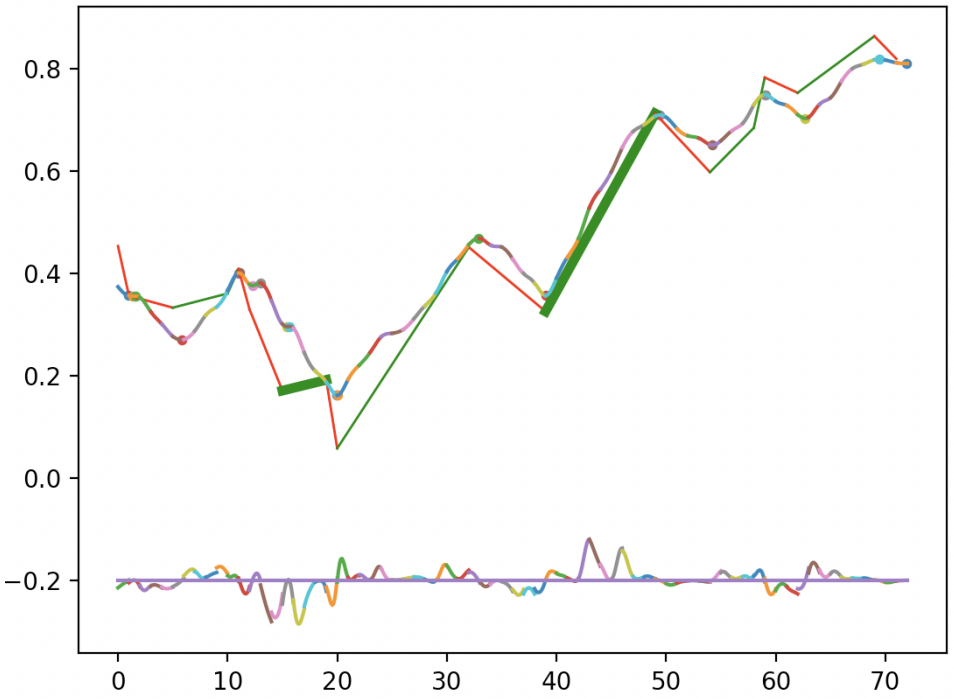


Рис. 19 Розділ трендів на сильні та слабкі

Таке зображення полегшує розуміння графіку.

2.3 Акумулювання

2.3.1 Введення

На екстремумах ціни залишаються пули ліквідності. Вони створені ордерами, які люди ставлять на точках розвороту графіку. Ціна змінюється від однієї ліквідності до іншої.

Рис. 20 Точки ліквідності на свічковому графіку

Часто зняття сильної ліквідності супроводжується імпульсами у напрямку руху. Це відбувається через те, що спрацьовують ордери інших учасників ринку. Чим більша ліквідність – тим більший результуючий імпульс. Таким чином, акумулюючись, ціна накопичує собі ліквідність, а великі гравці в цей час накопичують позицію (Фази B, C, D в вище описаній моделі Вайкоффа).

На цьому прикладі відбувається акумуляція зі звуженням діапазону цін:

Рис. 21 Акумуляція зі звуженням діапазону цін

Цим самим з обох сторін залишається зона з великою кількістю ліквідності. А потім стається пробій. Така формація називається трикутник.

На цьому прикладі відбувається акумуляція без звуження діапазону:

Рис. 22 Акумуляція без звуження діапазону цін

Також накопичується ліквідність, яка потім провокує імпульс. Будемо шукати такі формації на ринку.

2.3.2 Пошук акумулювань

Наша мета отримати промені, які будуть дотикатися до нашої множини в найбільшій кількості точок з певною похибкою.

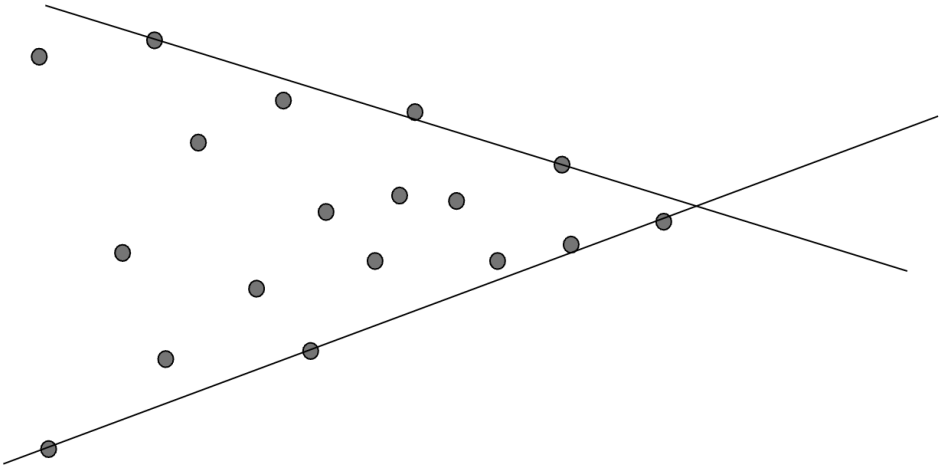


Рис. 23 Схематичне зображення шуканого акумулювання

Горизонтальна складова таких променів має спільний напрям зі шкалою часу. Всі точки мають лежати з однієї сторони такого променя з певною похибкою. З таких променів конструюємо зону акумуляції.

Додамо всі точки екстремумів на площину, та будемо проводити промінь через кожні дві. Для кожного променя рахуємо кількість точок на ньому та перевіряємо, щоб усі інші були з однієї сторони. Вважаємо, що точка лежить на промені, якщо відстань від точки до променя менша за обрану похибку d.

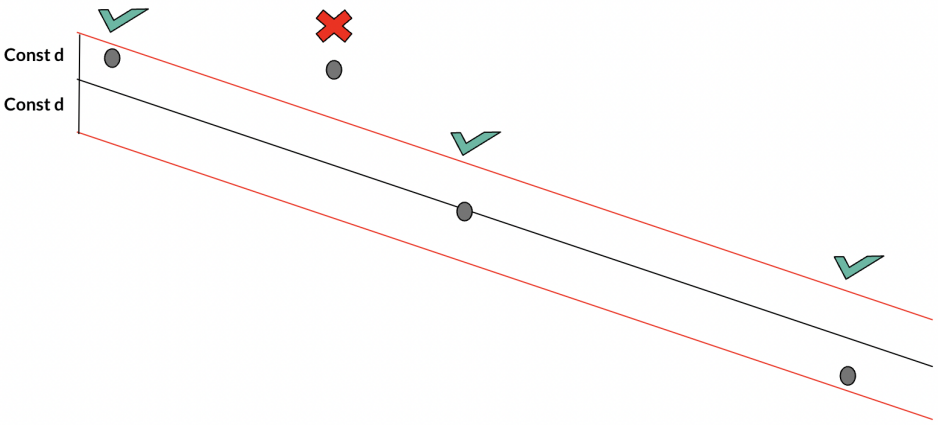


Рис. 24 Зображення знаходження точки в околі прямої

Оцінку променя будемо рахувати за допомогою ядра Гауса

$$G(\Delta y) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\Delta y^2}{2d^2}}$$

Як сума таких значень по всіх точках на промені.

Додамо також систему валідації:

- Вимагаємо не менше двох дотиків до кожного променя
- Вимагаємо щоб між сусідніми дотиками до променя була відстань не більше половини всього відрізку

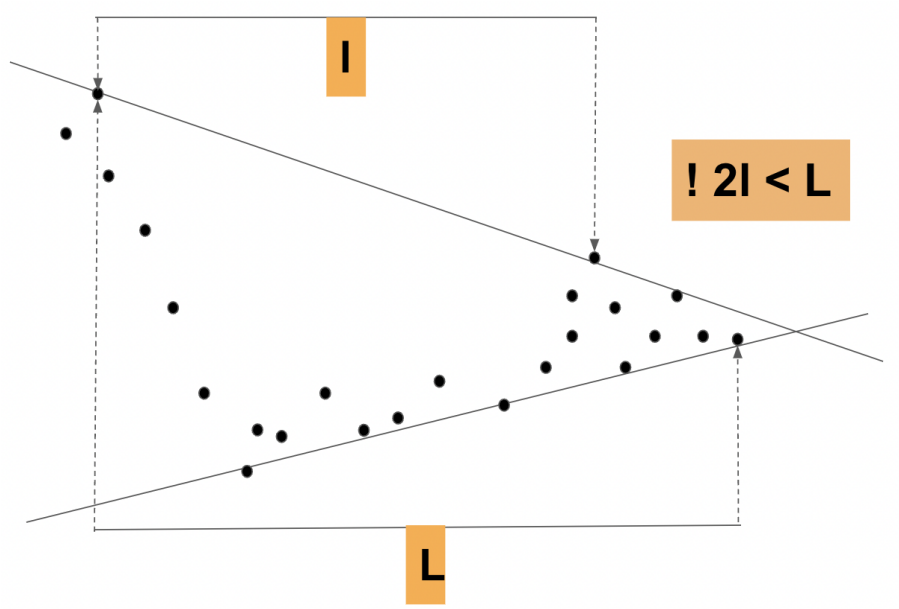


Рис. 25 Ілюстрація другої умови

- Вимагаємо, щоб дотики до верхнього і нижнього променів чергувались. Тобто якщо графік торкається одного променя, відходить на іншу половину трикутника, і потім повертається до нього ж, то ми не розглядаємо таку комбінацію.

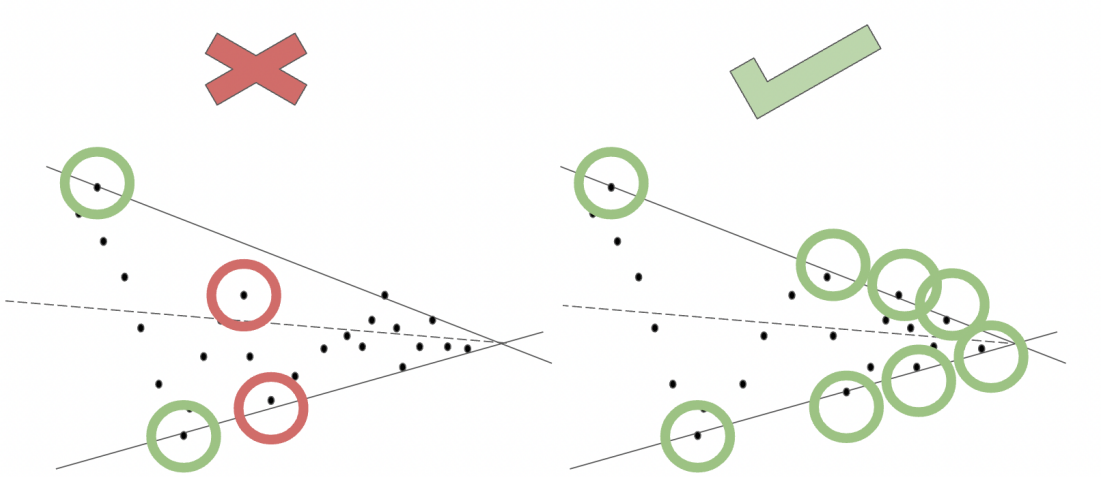


Рис. 26 Ілюстрація третьої умови

- Вимагаємо, щоб акумуляція починалася з сильного тренду - тобто початок променів шукаємо в певному проміжку навколо кінців сильних трендів.
- Вимагаємо, щоб сумарний об’єм кожного руху в бік тренда був більшим за об’єм послідовної корекції.

Розглянемо отримані результати:

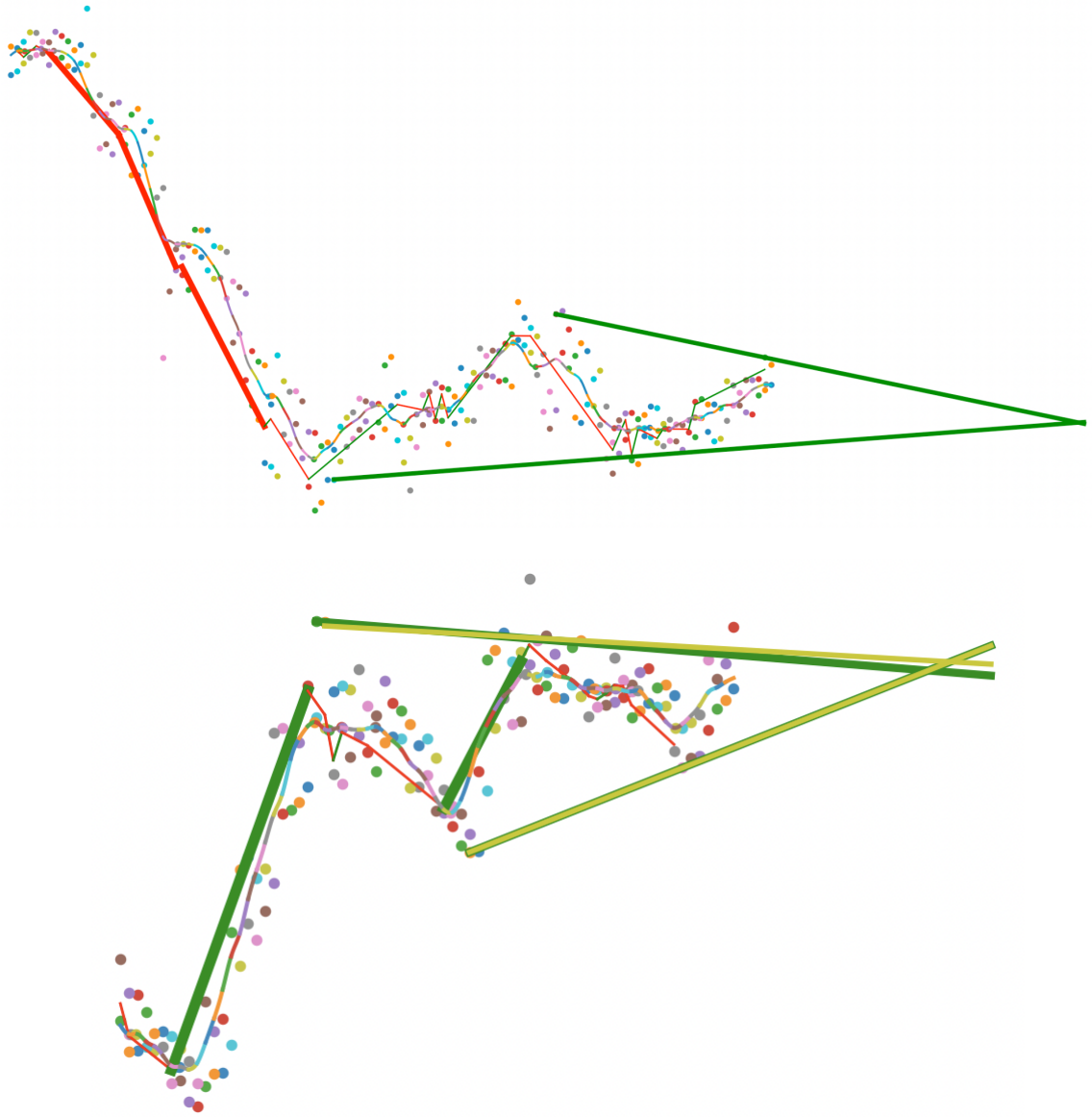


Рис. 27 Два приклади пошуку сильних трендів та акумуляцій разом

На першому прикладі бачимо сильний рух вниз, і після нього знайдена акумуляція. На другому - два сильні імпульси вгору, на яких базується акумуляція.

Таким чином похідна функції ціни стала фундаментом для подальшого аналізу фаз ринку.

ДОПОМІЖНІ ЗАСТОСУНКИ

Завантажувач даних

Для тестування алгоритмів я використовую реальні дані про ціну монет. Їх я отримую з криптовалютної біржі Binance за допомогою бібліотеки python-binance. Застосунок має такі параметри: дати початку і кінця періоду, TF, назва монети. Записує в файл для кожної одиниці TF екстремуми цін, відкриття за закриття свічки та об’єм.

Зображувач результатів

Для оцінки результатів роботи алгоритмів я зображую їх на графіку. Графіки будую за допомогою бібліотеки matplotlib. Зображувач приймає точки екстремумів свічок, дотичні промені, тренди, результат симуляції, введений індикатор і зображує це все на одному графіку. Таким чином зручно робити висновки щодо майбутнього монет.

ВИСНОВКИ

Висока волатильність криптовалюти відкриває чудові можливості до заробітків. В наш час криптовалюта неперервно набирає популярність серед нових користувачів та інвесторів. Багато грошей входять у ринок і це чудовий час діяти. Але з іншого боку, криптовалюта – це високі ризики втрат. Тому важливо робити детальний аналіз ринку перед будь-якими діями.

У ході дослідження я узагальнив похідні для функції ціни та показав варіанти їх використання, а саме:

- побудував симуляцію Монте-Карло, яка робить імовірнісний прогноз на рух ціни,
- створив індекс сили тренду, використовуючи похідну інтерполяції,
- розділив рух ціни на окремі тренди, знайшовши екстремуми.

Також було реалізовано пошук акумуляцій, що доповнює описані методи до практичного використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Investopedia (https://www.investopedia.com/terms/m/montecarlosimulation.asp)
2. Coingecko (https://www.coingecko.com)
3. Binance academy (https://academy.binance.com/ru/articles/the-wyckoff-method-explained)
4. Wikipedia (https://en.wikipedia.org/wiki/Markov_chain)
5. Investopedia (https://www.investopedia.com/articles/07/montecarlo.asp)
6. Algotrading101 (https://algotrading101.com/wiki/stochastic-calculus)
7. Kiel University (https://www.tf.uni-kiel.de/matwis/amat/comp_math/kap_1/backbone/r_se16.html)
8. Python Binance Docs (https://python-binance.readthedocs.io/en/latest)

ДОДАТКИ

main.cpp – реалізація усіх алгоритмів описаних у ході дослідження

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <math.h>
#include <random>

using namespace std;

string outputFolder, ticker, tf;

class Candle{
public:
    static const int parametersNum = 13;
    double index, openTime, openPrice, highPrice, lowPrice, closePrice, volume,
           closeTime, assetVolume, tradesCount, baseVolume, quoteVolume, ignore;
    Candle(vector<double>data){
        index = data[0];
        openTime = data[1];
        openPrice = data[2];
        highPrice = data[3];
        lowPrice = data[4];
        closePrice = data[5];
        volume = data[6];
        closeTime = data[7];
        assetVolume = data[8];
        tradesCount = data[9];
        baseVolume = data[10];
        quoteVolume = data[11];
        ignore = data[12];
    }
};

class Point{
public:
    int X;
    double Y;
    Point(int X_, double Y_){
        X = X_;
        Y = Y_;
    }
    Point(){}

    static bool cw (const Point& a, const Point& b, const Point& c) {
        return a.X*(b.Y-c.Y)+b.X*(c.Y-a.Y)+c.X*(a.Y-b.Y) < 0;
    }
```

```
    static bool ccw (const Point& a, const Point& b, const Point& c) {
        return a.X*(b.Y-c.Y)+b.X*(c.Y-a.Y)+c.X*(a.Y-b.Y) > 0;
    }

    static double getCoef(const Point& a, const Point& b){
        return (b.Y - a.Y) / (b.X - a.X);
    }
};

class Line{
public:
    double A, B, C;
    Line(){}
    Line(double a, double b, double c){
        A = a;
        B = b;
        C = c;
    }
    Line(Point a, Point b){
        A = a.Y - b.Y;
        B = b.X - a.X;
        C = -A * a.X - B * a.Y;
    }
    Line(Line a, Point pt){
        A = a.A;
        B = a.B;
        C = -A * pt.X - B * pt.Y;
    }

    void print() const{
        cout << "LINE:\n";
        cout << A << ' ' << B << ' ' << C << '\n';
    }

    Point getX(double Y) const{
        return Point((-B * Y - C) / A, Y);
    }

    Point getY(int X) const{
        return Point(X, (-A * X - C) / B);
    }

    static double dist(Line l, Point p){
        return fabs(l.getY(p.X).Y - p.Y);
    }

    bool checkDir(){
        return A * B < 0;
    }
```

```
bool operator==(const Line& b) const{
        return A == b.A &&
               B == b.B &&
               C == b.C;
    }
};

class Ray{
public:
    int from, br;
    Line dir;
    Ray(){}
    Ray(int fr, Line d, int b){
        from = fr;
        dir = d;
        br = b;
    }
    bool operator<(const Ray& b) const{
        return from < b.from;
    }

    bool operator==(const Ray& b) const{
        return from == b.from &&
        dir == b.dir &&
        br == b.br;
    }

    void print(bool upp = true){
        //cout << "--RAY--\n";
        cout << -from << '\n';
        if(upp)
            cout << -dir.A << ' ' << dir.B << ' ' << dir.C << '\n';
        else cout << -dir.A << ' ' << -dir.B << ' ' << dir.C << '\n';
    }
};

class Triangle{
public:
    Ray upp, low;
    double score;

    Triangle(){
        score = -1;
    }

    Triangle(Ray u, Ray l, double s){
        upp = u;
        low = l;
        score = s;
    }
```

```
    bool operator<(const Triangle& b) const{
        return score > b.score; // reversed order
    }

    bool operator==(const Triangle& b) const {
        return score == b.score &&
        upp == b.upp &&
        low == b.low;
    }
};

class Trend{
public:
    Point from, to;
    int type; // 1 - side, 2 - up, 3 - down
    bool strong, big;

    Trend(int a, int b, double pa, double pb, int tp, bool str, bool bg){
        big = bg;
        strong = str;
        type = tp;
        from = Point(a, pa);
        to = Point(b, pb);
    }

    void print(){
        cout << from.X << ' ' << from.Y << ' ' << to.X << ' ' << to.Y;
        cout << ' ' << type << ' ' << strong << ' ' << big << '\n';
    }
};

struct SplineTuple {
    double a, b, c, d, x;
};

class Builder{
private:
    vector<Point> upper_, lower_;
    vector<Trend> trends;
    vector<Triangle> trian;
    vector<pair<double, Ray> > upTangs, lowTangs;
    vector<double> volume, clp, opp;
    const double dC = 0.002, pad = 0.70, minPad = 5, epsP = 15,
                 shadowCut = 1, minCandleToCut = 0.07, neighPtsC = 9;
    const int minPointsOnRay = 2, takeTop = 3, maxBreaks = 2;
    double eps;
    int num;
```

```
bool checkForPads_(const Ray& ray, const vector<Point>& pts){
    int prev = pts[0].X;
    double dP = pad * num, maxPad = 0;
    for(int i = 0;i < num;i++){
        if(pts[i].X > ray.from)break;
        if(pts[i].Y - ray.dir.getY(pts[i].X).Y > -eps){
            maxPad = max(maxPad, 1.0 * pts[i].X - prev);
            if(pts[i].X - prev > dP)
                return false;
            prev = pts[i].X;
        }
    }
    if(ray.from - prev > dP)// || maxPad < minPad)
        return false;
    return true;
}

int countAdj_(const vector<Point>& pts, const Line& l, const Point& to, int br, int cf){
    int cnt = 0;
    int prev = pts[0].X - minPad - 10;
    for(int i = 0;i < num;i++){
        if(pts[i].X > to.X)break;
        auto ds = Line::dist(l, pts[i]);
        if(ds < eps && abs(pts[i].X - prev) > minPad) {
            ++cnt;
            prev = pts[i].X;
        }else{
            if(pts[i].Y - l.getY(pts[i].X).Y > eps) {
                --br;
                if(br < 0 || cf * clp[i] - l.getY(pts[i].X).Y > eps)return 0;
                ++cnt;
                prev = pts[i].X;
            }
        }
    }
    return cnt;
}

double score_(const vector<Point>& pts, const Line& l, const Point& to){
    double res = 0;
    int prev = -10;
    for(int i = 0;i < num;i++){
        if(pts[i].X > to.X)break;
        double td = Line::dist(l, pts[i]);
        if(td < eps) {
            if(abs(pts[i].X - prev) > minPad) {
                prev = pts[i].X;
                res += exp(-td * td / eps / eps / 2) / sqrt(2 * acos(-1));
            }else{
                res += exp(-td * td * neighPtsC / eps / eps / 2) / sqrt(2 * acos(-1));
            }
        }
    }
    return res;
}
```

```
pair<double, Ray> getRay_(const Point& a, const Point& b, const vector<Point>& pts,
                          int br = 0, int cf = 1, bool dbg = 0){
    auto line = Line(a, b);
    if(!line.checkDir()){
        return {-1, Ray()};
    }
    vector<Point>cl;
    for(int i = 0;i < num;i++){
        if(pts[i].X > b.X)break;
        if(Line::dist(line, pts[i]) < eps) {
            cl.push_back(pts[i]);
        }
    }
    double best = -1;
    Ray res;
    for(auto pt: cl){
        auto curLine = Line(line, pt);
        int adj = countAdj_(pts, curLine, b, br, cf);
        if(adj < minPointsOnRay)continue;
        auto scr = score_(pts, curLine, b);
        if(scr > best) {
            auto start = curLine.getY(b.X);
            auto cur = Ray(start.X, curLine, br);
            if(checkForPads_(cur, pts)) {
                res = cur;
                best = scr;
            }
        }
    }
    return {best, res};
}

void getTangs_(const vector<Point>& pts, vector<pair<double, Ray> >& rays, int cf){
    eps = 0.015;
    rays.clear();
    if(num == 1)return;
    for (int i = 1; i < num; ++i) {
        int cur = i, curBr = i, bs[2] = {0, 0};
        for(int j = i - 1;j >= 0;j--){
            auto temp = Line(pts[j], pts[i]);
            auto adj1 = countAdj_(pts, temp, pts[i], 0, cf);
            auto adj2 = countAdj_(pts, temp, pts[i], maxBreaks, cf);
            if(adj1 > bs[0]){
                bs[0] = adj1;
                cur = j;
            }
            if(adj2 > bs[1]){
                bs[1] = adj2;
                curBr = j;
            }
        }
```

```
            if(cur != i) {
                auto curRay = getRay_(pts[cur], pts[i], pts, 0, cf);
                if (curRay.first > 0) {
                    rays.push_back(curRay);
                }
            }

            if(curBr != i) {
                auto curRay = getRay_(pts[curBr], pts[i], pts, maxBreaks, cf);
                if (curRay.first > 0) {
                    rays.push_back(curRay);
                }
            }
        }
    }

    void addTangs(vector<pair<double, Ray> >& up, vector<pair<double, Ray> >& lw, pair<int, int>& from, int to){
        to++;
        while(from.first < lowTangs.size() && lowTangs[from.first].second.from <= to){
            lw.push_back(lowTangs[from.first]);
            from.first++;
        }
        while(from.second < upTangs.size() && upTangs[from.second].second.from <= to){
            up.push_back(upTangs[from.second]);
            from.second++;
        }
    }

    int getCorrection(int from){
        int cur = from;
        while(cur > 0 && trends[cur - 1].type != trends[from].type) {
            --cur;
        }
        return -trends[cur].to.X;
    }

    bool valid(Triangle tr, bool upp){// upp 1(uptrend)/0(downtrend)
        //return 1;
        if(upp && tr.upp.from < tr.low.from) return false;
        if(!upp && tr.upp.from > tr.low.from) return false;
        if(!upp && tr.upp.br > 0)return false;
        if(upp && tr.low.br > 0)return false;
        double awayTh = 2, maxHalfCandles = 0.5, narrowing = 0.8;
        int trFrom = max(tr.upp.from, tr.low.from);
        if((tr.low.dir.getY(trFrom).Y + tr.upp.dir.getY(trFrom).Y) * narrowing <
                tr.low.dir.getY(lower_[0].X).Y + tr.upp.dir.getY(upper_[0].X).Y)
            return false;
        int cnt[2], halfCnt = 0, maxHalf = (num + max(tr.upp.from, tr.low.from)) * maxHalfCandles;
        bool upHalf;
        bool cur;
        int i = 0;
        double mn = -lower_[0].Y, mx = upper_[0].Y, curMn = 1, curMx = 0;
        while(upper_[i].Y - tr.upp.dir.getY(upper_[i].X).Y < -eps &&
            lower_[i].Y - tr.low.dir.getY(lower_[i].X).Y < -eps){
            mn = min(mn, -lower_[i].Y);
            mx = max(mx, upper_[i].Y);
            ++i;
        }
```

```
cur = (lower_[i].Y - tr.low.dir.getY(lower_[i].X).Y > -eps);
cnt[0] = cnt[1] = 0;
double vol = 0, c = 0, prevv = -1, shadowTh = 3;
bool away = 0;
for(;upper_[i].X <= tr.upp.from || lower_[i].X <= tr.low.from;i++){
    double lpr = min(opp[i], clp[i]) + lower_[i].Y, upr = upper_[i].Y - max(opp[i], clp[i]);
    if(volume[i] < 0 && !upp){
        vol -= volume[i];
        ++c;
    }
    if(volume[i] > 0 && upp){
        vol += volume[i];
        ++c;
    }

    curMn = min(curMn, -lower_[i].Y);
    curMx = max(curMx, upper_[i].Y);

    double lowY = -tr.low.dir.getY(lower_[i].X).Y,
           uppY = tr.upp.dir.getY(upper_[i].X).Y,
           curY = clp[i];

    if(upper_[i].X <= tr.upp.from && upper_[i].Y - tr.upp.dir.getY(upper_[i].X).Y > -eps &&
           upr * shadowTh < uppY - lowY){
        if(!cur){
            if(away){
                if(upp && curMn > mn) {
                    return false;
                }
                if(curMn <= mn){
                    ++cnt[0];
                    ++cnt[1];
                }
            }
            vol = c = 0;
            curMn = 1; curMx = 0;
        }else{
            if (c > 0) {
                if (prevv > 0 && !upp && prevv > vol / c) {
                    return false;
                }
                prevv = vol / c;
                vol = c = 0;
            }
            curMn = 1; curMx = 0;
            cnt[0]++;
            if(volume[i] < 0){vol = -volume[i]; c = 1;};
            away = false;
            cur = false;
        }
    }
```

```
            if(lower_[i].X <= tr.low.from && lower_[i].Y - tr.low.dir.getY(lower_[i].X).Y > -eps &&
                    lpr * shadowTh < uppY - lowY){
                if(cur){
                    if(away){
                        if(curMx < mx && !upp) {
                            return false;
                        }
                        if(curMx >= mx){
                            ++cnt[0];
                            ++cnt[1];
                        }
                    }
                    vol = c = 0;
                    curMn = 1; curMx = 0;
                }else{
                    if(c > 0) {
                        if (prevv > 0 && upp && prevv > vol / c) {
                            return false;
                        }
                        prevv = vol / c;
                        vol = c = 0;
                    }
                    curMn = 1; curMx = 0;
                    cnt[1]++;
                    if(volume[i] > 0){vol = volume[i]; c = 1;}
                    away = false;
                    cur = true;
                }
            }

            if(cur && awayTh * fabs(curY - lowY) > fabs(uppY - lowY))away = 1;
            if(!cur && awayTh * fabs(curY - uppY) > fabs(uppY - lowY))away = 1;

            bool curUpHalf = (2 * fabs(curY - lowY) > fabs(uppY - lowY));

            if(halfCnt > 0 && curUpHalf == upHalf){
                ++halfCnt;
            }else{
                upHalf = curUpHalf;
                halfCnt = 1;
            }

            if(halfCnt > maxHalf){
                return false;
            }

            mn = min(mn, -lower_[i].Y);
            mx = max(mx, upper_[i].Y);
        }
        cnt[!cur]++;
        return cnt[0] >= minPointsOnRay && cnt[1] >= minPointsOnRay;
    }
```

```
public:
    Builder(const vector<Candle>& data, vector<Trend> trends_){
        trends = trends_;
        reverse(trends.begin(), trends.end());
        cout << "SZ " << data.size() << '\n';
        double mn = data[0].openPrice, mx = data[0].openPrice;
        for(auto el: data){
            mx = max(mx, el.closePrice);
            mn = min(mn, el.closePrice);
        }
        num = data.size();
        opp.resize(num);
        clp.resize(num);
        volume.resize(num);
        upper_.resize(num);
        lower_.resize(num);
        int ii = 0;
        for(auto el: data){
            double cmx = max(el.openPrice, el.closePrice), cmn = min(el.openPrice, el.closePrice),
                   sz = cmx - cmn;
            if(sz > (mx - mn) * minCandleToCut) {
                if (el.highPrice - cmx > shadowCut * sz) {
                    //el.highPrice = cmx;
                }
                if (cmn - el.lowPrice > shadowCut * sz) {
                    //el.lowPrice = cmn;
                }
            }
            clp[ii] = el.closePrice;
            opp[ii] = el.openPrice;
            volume[ii] = el.openPrice < el.closePrice ? el.volume : -el.volume;
            upper_[ii] = Point(-el.openTime, el.highPrice);
            lower_[ii++] = Point(-el.openTime, -el.lowPrice);
        }
        reverse(opp.begin(), opp.end());
        reverse(clp.begin(), clp.end());
        reverse(volume.begin(), volume.end());
        reverse(upper_.begin(), upper_.end());
        reverse(lower_.begin(), lower_.end());
    }

    void build(){
        getTangs_(upper_, upTangs, 1);
        getTangs_(lower_, lowTangs, -1);
        vector<pair<double, Ray> > up, lw;
        pair<int, int> from = {0, 0};
        double mx = -1, mn = 2;
        int extInd = num;
        for(int i = 0;i < trends.size();i++){
            auto trend = trends[i];
            if(trend.type == 2 && trend.from.Y >= mn)
                trend.strong = 0;
            if(trend.type == 3 && trend.from.Y <= mx)
                trend.strong = 0;
            while(extInd-- > trend.from.X){
                mx = max(mx, upper_[num - 1 - extInd].Y);
                mn = min(mn, -lower_[num - 1 - extInd].Y);
            }
```

```
            if(!trend.strong || trend.type == 1)
                continue;
            int to = -trend.from.X;
            addTangs(up, lw, from, to);
            auto corrTo = getCorrection(i);
            int cur;
            if(trend.type == 2){//uptrend
                cur = from.second - 1;
                if(lw.empty() || cur <= 0)continue;
                if(upTangs[cur].second.from < corrTo)continue;
                while(cur > 0 && upTangs[cur - 1].second.from >= corrTo){
                    --cur;
                }
                for(;cur < from.second && cur < upTangs.size();cur++){
                    int pInd = num - 1 + upTangs[cur].second.from;
                    if(upper_[pInd].Y - upper_[pInd+1].Y < -eps){
                        continue;
                    }
                    Triangle curRes;
                    for(auto ray: lw){
                        if(ray.first + upTangs[cur].first < curRes.score ||
                                (ray.second.from + num) * 2.5 < upTangs[cur].second.from + num)continue;
                        auto curTr = Triangle(upTangs[cur].second, ray.second, ray.first + upTangs[cur].first);
                        if(valid(curTr, 1))curRes = curTr;
                    }
                    if(curRes.score > 0)trian.push_back(curRes);
                }
            }else{//downtrend
                cur = from.first - 1;
                if(up.empty() || cur <= 0)continue;
                if(lowTangs[cur].second.from < corrTo)continue;
                while(cur > 0 && lowTangs[cur - 1].second.from >= corrTo){
                    --cur;
                }
                for(;cur < from.first && cur < lowTangs.size();cur++){
                    int pInd = num - 1 + lowTangs[cur].second.from;
                    if(lower_[pInd].Y - lower_[pInd+1].Y < -eps){
                        continue;
                    }
                    Triangle curRes;
                    for(auto ray: up){
                        if(ray.first + lowTangs[cur].first < curRes.score ||
                           (ray.second.from + num) * 2.5 < lowTangs[cur].second.from + num)continue;
                        auto curTr = Triangle(ray.second, lowTangs[cur].second, ray.first + lowTangs[cur].first);
                        if(valid(curTr, 0))curRes = curTr;
                    }
                    if(curRes.score > 0)trian.push_back(curRes);
                }
            }
        }
        sort(trian.begin(), trian.end());
        cout << "TRIAN SIZE " << trian.size() << '\n';
        unique(trian.begin(), trian.end());
        cout << "TRIAN SIZE " << trian.size() << '\n';
    }
```

```
    void output(){
        auto trianFile = outputFolder + "Results/triangles.out";
        freopen(trianFile.c_str(), "w", stdout);
        int to = min(int(trian.size()), takeTop);
        cout << to << '\n';
        for(int i = 0;i < to;i++) {
            trian[i].upp.print();
            trian[i].low.print(false);
        }
        fclose(stdout);

        auto scoreFile = outputFolder + "Results/scores.out";
        freopen(scoreFile.c_str(), "w", stdout);
        cout << to << '\n';
        for(int i = 0;i < to;i++)
            cout << trian[i].score << '\n';
        fclose(stdout);


        auto pointsFile = outputFolder + "Results/points.out";
        freopen(pointsFile.c_str(), "w", stdout);
        cout << num * 2 << '\n';
        for(int i = 0;i < num;i++){
            cout << -lower_[i].X << ' ' << -lower_[i].Y << '\n';
        }
        for(int i = 0;i < num;i++){
            cout << -upper_[i].X << ' ' << upper_[i].Y << '\n';
        }
        fclose(stdout);
    }
};

class MonteCarlo{
private:
    const double iter = 0.2, conf = 0.8;
    const int sims = 100, div = 100;
    vector<double> points_, stat;
    double m, var, mn, mx, seg, start, drift;
    default_random_engine generator_;
    normal_distribution<double> distribution_;
    vector<vector<double> > allScenarios;
    int cnt_, steps;

    double step(double s){
        double e = distribution_(generator_);
        s *= exp(drift + sqrt(var) * e);
        return s;
    }
```

```
    double generate(const int& num){
        auto s = start;
        for(int i = 0;i < steps;i++){
            s = step(s);
            allScenarios[i][num] = s;
        }
        return s;
    }

public:
    MonteCarlo(const vector<Candle>& data){
        distribution_ = normal_distribution<double>(0, 1);
        cnt_ = data.size();
        mn = data[0].openPrice;
        mx = data[0].openPrice;
        m = var = 0;
        int nm = 0;
        points_.resize(cnt_);
        stat.resize(div);
        for(int i = 0;i < div;i++)stat[i] = 0;
        for(auto el: data) {
            points_[nm] = log(el.closePrice / el.openPrice);
            m += points_[nm];
            if(el.openPrice < el.closePrice){
                mx = max(mx, el.closePrice);
                mn = min(mn, el.openPrice);
            }else{
                mx = max(mx, el.openPrice);
                mn = min(mn, el.closePrice);
            }
            ++nm;
        }
        seg = mx - mn;
        mx += seg;
        mn -= seg;
        seg *= 3;
        start = data.back().closePrice;
        m /= cnt_;
        for(int i = 0;i < cnt_;i++){
            var += (m - points_[i]) * (m - points_[i]);
        }
        var /= (cnt_ - 1);
        steps = iter * cnt_;
        allScenarios.assign(steps, vector<double>(sims));
        drift = m - var / 2;
        cout << "VAR: " << var << '\n';
        cout << "MEAN: " << m << '\n';
    }
```

```
    void simulate(){
        for(int i = 0;i < sims;i++){
            auto s = generate(i);
            int ind = (s - mn) / (seg / div);
            if(ind < 0)ind = 0;
            ++stat[ind];
        }
        for(int i = 0;i < steps;i++) {
            sort(allScenarios[i].begin(), allScenarios[i].end());
        }
        double mean = 0;
        int shift = (start - mn) / (seg / div);
        for(int i = 0;i < div;i++){
            mean += stat[i] * (i - shift);
        }
        mean /= sims;
        auto change = mean * seg / div / start;
        cout << "MONTE CARLO CHANGE EXP " << change * 100 << "%\n";
    }

    void output(){
        int ignore = sims * (1 - conf) / 2;
        auto minsFile = outputFolder + "Results/mins.out";
        freopen(minsFile.c_str(), "w", stdout);
        cout << steps << '\n';
        cout << start << '\n';
        for(int i = 0;i < steps;i++)
            cout << allScenarios[i][ignore] << ' ';
        cout << '\n';
        fclose(stdout);
        auto maxsFile = outputFolder + "Results/maxs.out";
        freopen(maxsFile.c_str(), "w", stdout);
        cout << steps << '\n';
        cout << start << '\n';
        for(int i = 0;i < steps;i++)
            cout << allScenarios[i][sims - 1  - ignore] << ' ';
        cout << '\n';
        fclose(stdout);
        auto midsFile = outputFolder + "Results/mids.out";
        freopen(midsFile.c_str(), "w", stdout);
        cout << steps << '\n';
        cout << start << '\n';
        for(int i = 0;i < steps;i++)
            cout << allScenarios[i][sims / 2] << ' ';
        cout << '\n';
        fclose(stdout);
    }
};
```

```
class Wyckoff{
private:
    const double breakPoint = 0.5, noTrend = 1.3, breakOut = 1.2, minTrend = 0.03,
                 minVolDif = 1, minSide = 0.05;
    double avgVol;
    vector<double> op, cl, vol;
    vector<Trend> trends_;
    int num;
    int a, b, c; // trend: start, end, max move back

    double getAvgVol(int from, int to){
        double res = 0;
        int cnt = 0;
        bool upp = (op[from] < cl[to]);
        for(int i = from;i <= to;i++) {
            if(upp && op[i] > cl[i])
                continue;
            if(!upp && op[i] < cl[i])
                continue;
            ++cnt;
            res += vol[i];
        }
        res /= cnt;
        return res;
    }

    void Add(int a, int b, int type){
        if(a > b)return;
        bool strong = getAvgVol(a, b) > avgVol * minVolDif,
            big = b - a > num * minTrend;
        //if(!strong && !big)return;
        trends_.push_back(Trend(a, b, op[a], cl[b], type, strong, big));
    }

    void goSide(int from){
        int cur = from;
        double mx = max(op[a], cl[b]), mn = min(op[a], cl[b]);
        while(cur < num){
            if(cl[cur] > op[cur]){
                if((mx - mn) * breakOut < cl[cur] - mn){
                    Add(a, cur, 1);
                    startTrend(cur);
                    return;
                }
            }else {
                if((mx - mn) * breakOut < mx - cl[cur]){
                    Add(a, cur, 1);
                    startTrend(cur);
                    return;
                }
            }
            ++cur;
        }
        Add(a, cur - 1, 1);
    }
```

```
void goUp(int from){
        int cur = from;
        while(cur < num){
            if(cl[cur] > op[cur]){
                if(cl[cur] > cl[b]){
                    b = c = cur;
                }
            }else{
                if(cl[cur] < cl[c]){
                    c = cur;
                    if((cl[c] - op[a]) / (cl[b] - op[a]) < breakPoint){
                        Add(a, b, 3);
                        a = b + 1;
                        b = c;
                        goDown(cur + 1);
                        return;
                    }
                }
            }
            if(max(minSide * num, (b - a) * noTrend) < cur - b){
                Add(a, b, 3);
                a = b + 1;
                b = c;
                goSide(cur + 1);
                return;
            }
            ++cur;
        }
        Add(a, b, 3);
        Add(b + 1, cur - 1, 1);
    }

    void goDown(int from){
        int cur = from;
        while(cur < num){
            if(cl[cur] < op[cur]){
                if(cl[cur] < cl[b]){
                    b = c = cur;
                }
            }else{
                if(cl[cur] > cl[c]){
                    c = cur;
                    if((cl[c] - op[a]) / (cl[b] - op[a]) < breakPoint){
                        Add(a, b, 2);
                        a = b + 1;
                        b = c;
                        goUp(cur + 1);
                        return;
                    }
                }
            }
```

```
            if(max(minSide * num, (b - a) * noTrend) < cur - b){
                Add(a, b, 2);
                a = b + 1;
                b = c;
                goSide(cur + 1);
                return;
            }
            ++cur;
        }
        Add(a, b, 2);
        Add(b + 1, cur - 1, 1);
    }

    void startTrend(int from){
        if(from >= num)
            return;
        a = b = c = from;
        if(op[from] < cl[from]){
            goUp(from + 1);
        }else{
            goDown(from + 1);
        }
    }

public:
    Wyckoff(vector<Candle> data) {
        num = data.size();
        op.resize(num);
        cl.resize(num);
        vol.resize(num);
        for(int i = 0;i < num;i++) {
            cl[i] = data[num - 1 - i].openPrice;
            op[i] = data[num - 1 - i].closePrice;
            vol[i] = data[num - 1 - i].volume;
        }
        avgVol = getAvgVol(0, num - 1);
    }

    void build(){
        startTrend(0);
        reverse(trends_.begin(), trends_.end());
        for(int i = 0;i < trends_.size();i++){
            swap(trends_[i].from, trends_[i].to);
            trends_[i].from.X = num - 1 - trends_[i].from.X;
            trends_[i].to.X = num - 1 - trends_[i].to.X;
        }
    }
```

```
    vector<Trend> trends(){
        return trends_;
    }

    void output(){
        auto outputFile = outputFolder + "results/trends.out";
        freopen(outputFile.c_str(), "w", stdout);
        cout << trends_.size() << '\n';
        for(int i = 0; i < trends_.size(); i++)
            trends_[i].print();
        fclose(stdout);
    }
};

class SplineInterpolation{
    vector<SplineTuple> spline;
    bool vol;

    void smooth(vector<double>& v){
        int steps = 3;
        for(int i = 0;i < v.size();i++){
            int cnt = 1;
            double sum = cnt * v[i];
            for(int j = max(0, i - steps);j < i;j++)
                ++cnt, sum += v[j];
            for(int j = i + 1;j <= i + steps && j < v.size();j++)
                cnt += 1, sum += 1 * v[j];
            v[i] = sum / cnt;
        }
    }
public:
    SplineInterpolation(const vector<Candle>& cs, bool vol=0):vol(vol) {
        vector<double>xs, ys;
        for(auto el: cs){
            xs.push_back(xs.size());
            if(vol) ys.push_back(el.volume);
            else ys.push_back(el.closePrice);
            cout << xs.back() << ' ' << ys.back() << '\n';
        }
        if(!vol)smooth(ys);
        int n = xs.size() - 1;
        std::vector<double> a(ys.begin(), ys.end());
        std::vector<double> b(n), d(n), h(n), alpha(n);
        std::vector<double> c(xs.size(), 0.0);

        for (int i = 0; i < n; ++i) {
            h[i] = xs[i + 1] - xs[i];
            alpha[i] = (a[i + 1] - a[i]) / h[i];
        }
```

```
        std::vector<double> mu(n), l(n + 1, 1.0), z(n + 1, 0.0);

        for (int i = 1; i < n; ++i) {
            mu[i] = h[i - 1] / (h[i - 1] + h[i]);
            l[i] = 2.0 * (xs[i + 1] - xs[i - 1]) - h[i - 1] * mu[i];
            z[i] = (3.0 * (alpha[i] - alpha[i - 1]) - h[i - 1] * z[i - 1]) / l[i];
        }

        // Backward substitution
        for (int j = n - 1; j >= 0; --j) {
            c[j] = z[j] - mu[j] * c[j + 1];
            b[j] = (a[j + 1] - a[j]) / h[j] - h[j] * (c[j + 1] + 2.0 * c[j]) / 3.0;
            d[j] = (c[j + 1] - c[j]) / (3.0 * h[j]);
        }

        for (int i = 0; i < n; ++i) {
            spline.push_back({a[i], b[i], c[i], d[i], xs[i]});
        }
    }

    vector<SplineTuple> get_tuples(){
        return spline;
    }

    double get(double x) {
        if (spline.empty()) {
            return 0.0;  // No data points
        }
        const SplineTuple* s;

        if (x <= spline.front().x) {
            s = &spline.front();
        } else if (x >= spline.back().x) {
            s = &spline.back();
        } else {
            size_t lo = 0, hi = spline.size() - 1;
            while (hi - lo > 1) {
                size_t mid = lo + (hi - lo) / 2;
                if (x < spline[mid].x) {
                    hi = mid;
                } else {
                    lo = mid;
                }
            }
            s = &spline[lo];
        }

        double dx = x - s->x;
        return s->a + s->b * dx + s->c * dx * dx + s->d * dx * dx * dx;
    }
```

```
    double get_der(double x) {
        if (spline.empty()) {
            return 0.0;  // No data points
        }
        const SplineTuple* s;

        if (x <= spline.front().x) {
            s = &spline.front();
        } else if (x >= spline.back().x) {
            s = &spline.back();
        } else {
            size_t lo = 0, hi = spline.size() - 1;
            while (hi - lo > 1) {
                size_t mid = lo + (hi - lo) / 2;
                if (x < spline[mid].x) {
                    hi = mid;
                } else {
                    lo = mid;
                }
            }
            s = &spline[lo];
        }

        double dx = x - s->x;
        return s->b + 2 * s->c * dx + 3 * s->d * dx * dx;
    }

    void show(){
        auto splineFile = outputFolder;
        if(vol) splineFile += "Results/spline_vol.out";
        else splineFile += "Results/spline.out";
        freopen(splineFile.c_str(), "w", stdout);
        cout << spline.size() << '\n';
        for(auto el: spline){
            std::cout << el.d << ' ' << el.c << ' '<< el.b << ' '<< el.a << '\n';
        }
        cout << '\n';
    }
};

class DervTrends{
private:
    vector<Trend> trends_;
    vector<double> zeros;
    vector<Candle> candles;
    SplineInterpolation* spline;
```

```
    vector<double> get_zeros(double a, double b, double c, double from, double to){
        vector<double>zeros;
        double step = (to - from) / 100.0;
        for(double x = from; x <= to; x += step){
            double l = x, r = x + step;
            if(a * l * l + b * l + c == 0)
                zeros.push_back(l);
            else if ((a * l * l + b * l + c) * (a * r * r + b * r + c) < 0){
                while(r - l > step / 10000){
                    double m = (r + l) / 2;
                    if((a * l * l + b * l + c) * (a * m * m + b * m + c) > 0)
                        l = m;
                    else r = m;
                }
                zeros.push_back(r);
            }
        }
        return zeros;
    }

public:
    DervTrends(SplineInterpolation* spline, vector<Candle> candles): spline(spline), candles(candles){
        auto splines = spline -> get_tuples();
        int cnt = 0;
        for(auto el: splines){
            auto ext = get_zeros(3 * el.d, 2 * el.c, el.b, 0, 1);
            for(auto zr: ext)
                zeros.push_back(zr + cnt);
            ++cnt;
        }
    }

    void build(){
        for(int i = 1;i < zeros.size();i++){
            double a = zeros[i - 1], b = zeros[i];
            double pa = candles[int(a)].closePrice, pb = candles[int(b)].closePrice;
            int tp = pa < pb ? 2 : 3;
            trends_.push_back(Trend(a, b, pa, pb, tp, false, false));
        }

    }

    void output_zeros(){
        auto extsFile = outputFolder + "Results/exts.out";
        freopen(extsFile.c_str(), "w", stdout);
        cout << zeros.size() << '\n';
        for(auto zero: zeros){
            cout << zero << ' ' << spline->get(zero) << '\n';
        }
    }

    void output(){
        auto outputFile = outputFolder + "results/dtrends.out";
        freopen(outputFile.c_str(), "w", stdout);
        cout << trends_.size() << '\n';
        for(int i = 0; i < trends_.size(); i++)
            trends_[i].print();
        fclose(stdout);
    }
```

```
    vector<Trend> trends(){
        return trends_;
    }

    void score_trends(SplineInterpolation* price, SplineInterpolation* vol){
        auto outputFile = outputFolder + "results/dtscores.out";
        freopen(outputFile.c_str(), "w", stdout);
        vector<double> scores;
        cout << "SCORES\n";
        for(auto& el: trends_){
            int steps = 1000;
            double from = el.from.X, to = el.to.X;
            double step = (to - from) / steps;
            long double sum = 0;
            for(double cur = from;cur < to;cur += step){
                sum += step * price -> get_der(cur) * vol -> get(cur);
            }
            scores.push_back(fabs(sum));
        }
        auto sorted = scores;
        sort(sorted.begin(), sorted.end());
        reverse(sorted.begin(), sorted.end());
        cout << trends_.size() << ' '<<   sorted[trends_.size() / 10] <<    '\n';
        for(int i = 0;i < trends_.size();i++){
            if(scores[i] > sorted[trends_.size() / 10])
                trends_[i].big = trends_[i].strong = true;
        }
        cout << '\n';
    }
};

vector<Candle> testRead(bool btc){
    vector<double>cur(Candle::parametersNum);
    vector<Candle>res;
    int cnt = 0;
    double mx = -1, mn = -1,
        mxv = -1, mnv = -1;

    while(cin >> cur[2] >> cur[3] >> cur[4] >> cur[5] >> cur[6]){
        if(cur[2] == -1)
            return res;
        if(mx < cur[3] || mx < 0)mx = cur[3];
        if(mn > cur[4] || mn < 0)mn = cur[4];
        if(mxv < cur[6] || mxv < 0)mxv = cur[6];
        if(mnv > cur[6] || mnv < 0)mnv = cur[6];
        cur[1] = cnt++;
        res.push_back(Candle(cur));
    }

    for(int i = 0;i < res.size();i++){
        res[i].openPrice = (res[i].openPrice - mn) / (mx - mn);
        res[i].closePrice = (res[i].closePrice - mn) / (mx - mn);
        res[i].highPrice = (res[i].highPrice - mn) / (mx - mn);
        res[i].lowPrice = (res[i].lowPrice - mn) / (mx - mn);
        res[i].volume = (res[i].volume - mnv) / (mxv - mnv);
        cout << res[i].volume << '\n';
    }

    return res;
}
```

```
void doWork(string outp, string tckr, string tfr){
    outputFolder = outp;
    ticker = tckr;
    tf = tfr;
    auto sampleData = outputFolder + "SampleData/" + ticker + tf + ".txt";
    freopen(sampleData.c_str(),"r",stdin);
    auto candles = testRead(false);
    auto spline = new SplineInterpolation(candles);
    auto spline_vol = new SplineInterpolation(candles, true);
    auto simulation = new MonteCarlo(candles);
    auto wyckoff = new Wyckoff(candles);
    auto derv_trends= new DervTrends(spline, candles);
    wyckoff -> build();
    derv_trends -> build();
    auto builder = new Builder(candles, wyckoff->trends());
    simulation -> simulate();
    builder -> build();
    spline -> show();
    spline_vol -> show();
    builder -> output();
    simulation -> output();
    wyckoff -> output();
    derv_trends -> score_trends(spline, spline_vol);
    derv_trends -> output();
    derv_trends -> output_zeros();
}

int main() {
    doWork("~/Projects/ChartAnalyser/Output/", "BTCUSDT", "4h");
    return 0;
}
```

painter.py – реалізація зображувача результатів

```
import matplotlib.pyplot as plt
import numpy as np

resDir = "~/Projects/ChartAnalyser/Output/Results"


fP = open(resDir + "/points.out", "r")
nP = int(fP.readline())
nCand = int(nP / 2)

for i in range(nP):
    a, b = [float(el) for el in fP.readline().split()]
    plt.scatter(a, b, s=10)


fTrian = open(resDir + "/triangles.out", "r")
nTrian = int(fTrian.readline())

col = ['g', 'y', 'r']
width = [3, 2, 1]

for i in range(nTrian):
    left = float(fTrian.readline())
    a, b, c = [float(el) for el in fTrian.readline().split()]
    x = np.linspace(left, nCand * 1.4, 100)
    y = (-a * x - c) / b
    plt.plot(x, y, col[i], linewidth=width[i])
    left = float(fTrian.readline())
    a, b, c = [float(el) for el in fTrian.readline().split()]
    x = np.linspace(left, nCand * 1.4, 100)
    y = (-a * x - c) / b
    plt.plot(x, y, col[i], linewidth=width[i])


fT = open(resDir + "/dtrends.out", "r")
nT = int(fT.readline())

for i in range(nT):
    a, b, c, d, e, f, g = [float(el) for el in fT.readline().split()]
    if f == 0:
        #continue
        f = 0
    wd = 4
    if f == 0:
        wd = 1
```

```
    col = 'y'
    if e == 2:
        col = 'g'
    if e == 3:
        col = 'r'
    plt.plot([a, c], [b, d], col, linewidth=wd)


fMn = open(resDir + "/mins.out", "r")
nMn = int(fMn.readline())
start = float(fMn.readline())
nCand = int(nP / 2)
mins = [float(el) for el in fMn.readline().split()]
mins.insert(0, start);
x = [i for i in range(nCand, nCand + nMn + 1)]
plt.plot(x, mins, 'm')

fMx = open(resDir + "/maxs.out", "r")
nMx = int(fMx.readline())
start = float(fMx.readline())
maxs = [float(el) for el in fMx.readline().split()]
maxs.insert(0, start);
x = [i for i in range(nCand, nCand + nMn + 1)]
plt.plot(x, maxs, 'm')

fMd = open(resDir + "/mids.out", "r")
nMd = int(fMd.readline())
start = float(fMd.readline())
nCand = int(nP / 2)
mids = [float(el) for el in fMd.readline().split()]
mids.insert(0, start);
x = [i for i in range(nCand, nCand + nMd + 1)]
plt.plot(x, mids, 'm')

fSpline = open(resDir + "/spline.out", "r")
nSpline = int(fSpline.readline())

splines = []
```

```
for i in range(nSpline):
    a, b, c, d = [float(el) for el in fSpline.readline().split()]
    splines.append([a, b, c])
    print('----\n', c)
    x = np.linspace(i, i + 1, 10)
    y = a * (x - i) ** 3 + b * (x - i) ** 2 + c * (x - i) + d
    yd = 3 * a * (x - i) ** 2 + 2 * b * (x - i) + c
    print(3 * a + 2 * b + c)

    plt.plot(x, y)
    #plt.plot(x, yd)

fSpline = open(resDir + "/spline_vol.out", "r")
nSpline = int(fSpline.readline())

vols = []

for i in range(nSpline):
    a, b, c, d = [float(el) for el in fSpline.readline().split()]
    vols.append([a, b, c, d])
    x = np.linspace(i, i + 1, 10)
    y = (a * (x - i) ** 3 + b * (x - i) ** 2 + c * (x - i) + d) / 4
    #plt.plot(x, y)

for i in range(len(vols)):
    x = np.linspace(i, i + 1, 10)
    y = (vols[i][0] * (x - i) ** 3 + vols[i][1] * (x - i) ** 2 + vols[i][2] * (x - i) + vols[i][3]) * \
        (3 * splines[i][0] * (x - i) ** 2 + 2 * splines[i][1] * (x - i) + splines[i][2]) * 2 - 0.2
    plt.plot(x, y)

x = np.linspace(0, nSpline, 10)
y = 0 * x - 0.2
plt.plot(x, y)

fExts = open(resDir + "/exts.out", "r")
nExts = int(fExts.readline())

for i in range(nExts):
    a, b = [float(el) for el in fExts.readline().split()]
    plt.scatter(a, b, s=10)

plt.show()
```

price_loader.py – реалізація завантажувача даних

```python
from binance.client import Client

client = Client()

ticker = 'BTCUSDT'
tf = '4h'
begin = '08 May, 2022'
end = '23 May, 2022'

sampleDir = "~/Projects/ChartAnalyser/Output/SampleData/"

bars = client.get_historical_klines(ticker, tf, begin, end)
btc = client.get_historical_klines("BTCUSDT", tf, begin, end)

with open(sampleDir + ticker + tf + ".txt", "w") as o:
    for line in bars:
        for i in range(1, 6):
            print(line[i], end=" ", file=o)
        print('', file=o)
```

ВІДГУК
на випускну кваліфікаційну роботу бакалавра
«Прогнозування ціни біткоїна засобами математичного аналізу»
студента 4-го курсу кафедри обчислювальної математики
факультету комп'ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка
Сидоренка Нікіти Сергійовича

У роботі розглядається курс біткоїна, Продемонстровано симуляцію методу Монте-Карло для отримання імовірнісного прогнозу, а також сплайн інтерполяцію 3го степеня. Отримані результати супроводжуються лінійним та «свічковим» графіками коливань курсу та їх наближенням. Виконується чисельний аналіз показників, зроблено висновки опираючись на історію ціни певної криптовалюти.

Методи та ідеї, розглянуті у роботі, мають практичне застосування та відображають ідею згладжування функції - в кожній точці обчислено нове значення як середнє арифметичне, що дозволяє виконати аналіз коливань курсу з точки зору усереднених показників.

Студент якісно та самостійно виконав роботу, досяг точних результатів кількома підходами. Продемонстровано належний рівень знань та кваліфікації. Роботу можу оцінити на «відмінно».

Асистент кафедри обчислювальної
математики факультету комп'ютерних
наук та кібернетики
Київського національного університету
імені Тараса Шевченка,
доктор філософії

Андрій
ТИМОШЕНКО

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра:

«Прогнозування ціни біткоїна засобами математичного аналізу»

студента 4-го курсу кафедри обчислювальної математики

факультету комп'ютерних наук та кібернетики

Київського національного університету імені Тараса Шевченка

Сидоренка Нікіти Сергійовича

Рецензована робота присвячена дослідженню курсу криптовалюти біткоїн за допомогою симуляції Монте-Карло, яка дозволяє виконати імовірнісний прогноз на рух, та сплайн інтерполяції. За наявними показниками курсу також виконано пошук трендів. Виконано графічну побудову з фактичними і апроксимованими показниками для кожного методу. Додатково розглянуто фази ринку відповідно до законів Вайкоффа.

Робота має практичне застосування та допомагає зробити короткострокове передбачення з обмеженням на ризик. Результати є наочними та точними.

Студент виконав роботу якісно та заслуговує на оцінку «відмінно», а також на присвоєння кваліфікації бакалавра.

Професор кафедри обчислювальної
математики факультету комп'ютерних
наук та кібернетики
Київського національного університету
імені Тараса Шевченка,
доктор фізико-математичних наук

Дмитро КЛЮШИН

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

СИСТЕМА ЗАПОБІГАННЯ ТА ВИЯВЛЕННЯ АКАДЕМІЧНОГО ПЛАГІАТУ

Довідка про оригінальність кваліфікаційної роботи за освітнім рівнем бакалавр

Експертна оцінка роботи науковим керівником :

Робота студента 4-го курсу Сидоренка Нікіти Сергійовича «Прогнозування ціни біткоїна засобами математичного аналізу» виконана самостійно, обсяг цитувань та запозичень становить 0.9% та не перевищує норму.

Науковий керівник: _____ Тимошенко А.А.
(підпис) (ПІБ)

Оператор: _____ Оноцький В.В.
(підпис) (ПІБ)